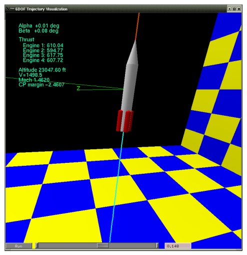
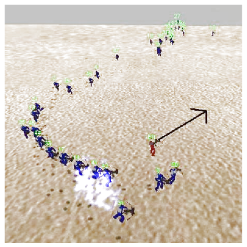
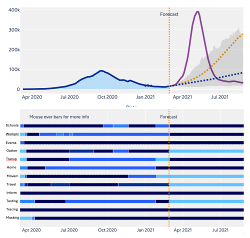
Neuroevolution of Behavior

Risto Miikkulainen

September 23, 2024

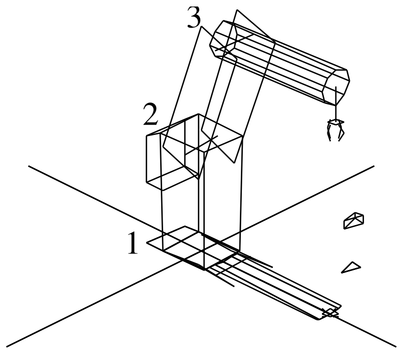
Introduction to Neuroevolution of Behavior

- Neuroevolution aims to construct agents that behave intelligently in simulated or real environments.
- Behavior is optimized at multiple levels:
  - Control tasks: locomotion for robots, production in bioreactors.
  - Behavioral strategies: navigation, gameplay, cognitive tasks.
  - Decision strategies: business, healthcare, societal decisions.

Neuroevolution for Control

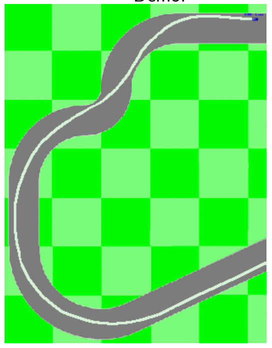
- Neuroevolution has been applied to various control tasks, demonstrating creative solutions.
- Agents evolve to compensate for challenges such as hardware failures.
  - E.g. controlling a robotic arm when a motor fails:



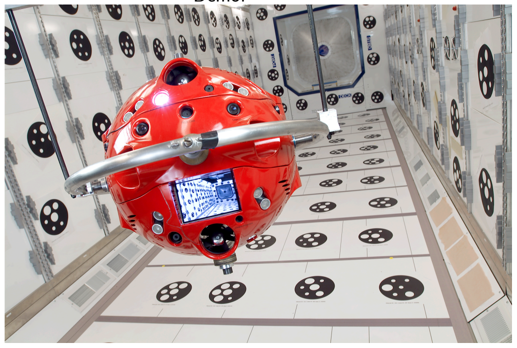
Creative Problem Solving

- Neuroevolution can find solutions not immediately obvious to human designers.
  - Driving a race cars by maximizing speed instead of minimizing distance.
  - Stopping spacecraft by rotating it around.

Demo:

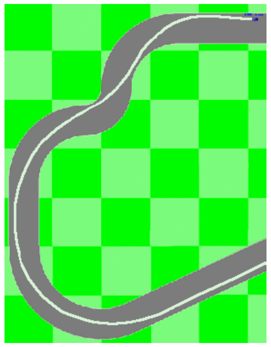
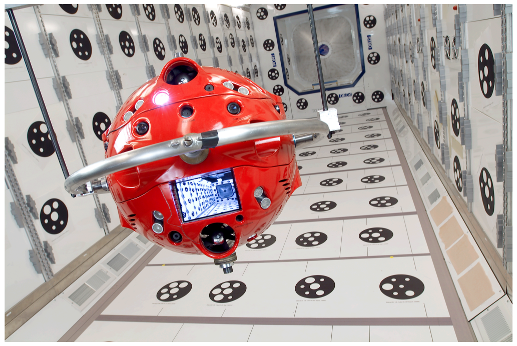


Demo:



Challenge: Robustness

- Robust control is difficult:
  - Environments can be dynamic, nonlinear, and noisy.
  - Conditions can change over time (e.g., sensor failure, obstacles, ice...).
- Neural networks can handle noise, nonlinear effects, and partial observability.
- Evolution needs to see enough such variation to be effective.

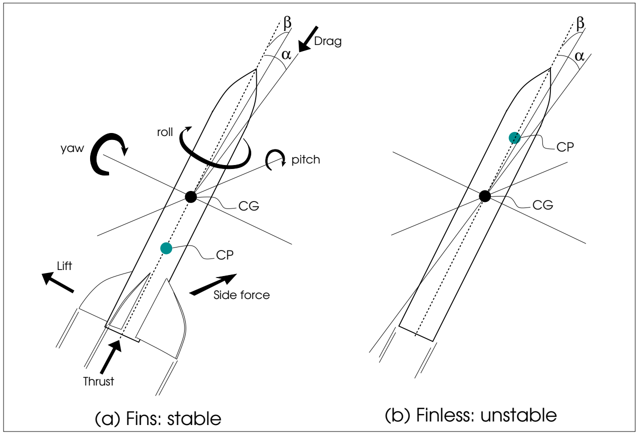



Example: Controlling a Finless Rocket

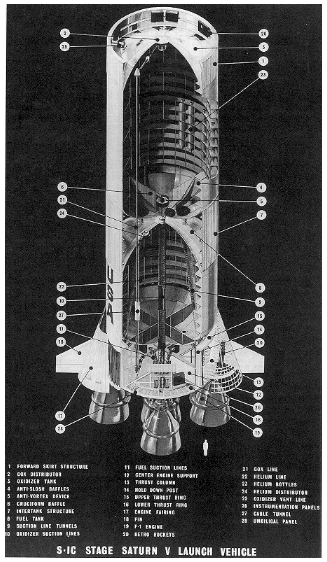
- Task: Stabilize a finless version of the Interorbital Systems RSX-2 sounding rocket
  - Scientific measurements in the upper atmosphere
  - 4 liquid-fueled engines with variable thrust
  - Without fins will fly much higher for same amount of fuel

Rocket Stability

- Drag from fins pulls the Center of Pressure (CP) behind Center of Gravity (CG)
- Without fins, need active control.

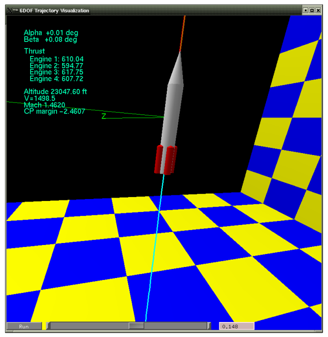


Active Rocket Guidance



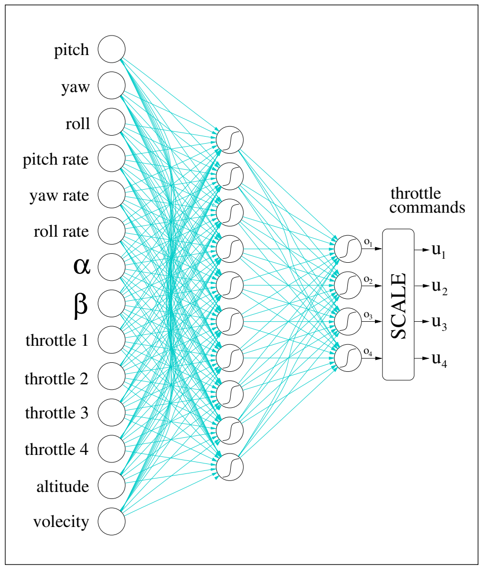
- Used on large scale launch vehicles (Saturn, Titan)
- Typically based on classical linear feedback control
- High level of domain knowledge required
- Expensive, heavy

Simulation Environment: JSBSim



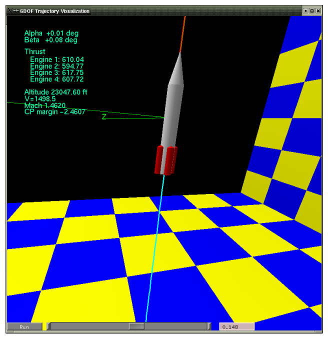
- General rocket simulator
- Models complex interaction between airframe, propulsion, aerodynamics, and atmosphere
- Used by IOS in testing their rocket designs
- Accurate geometric model of the RSX-2

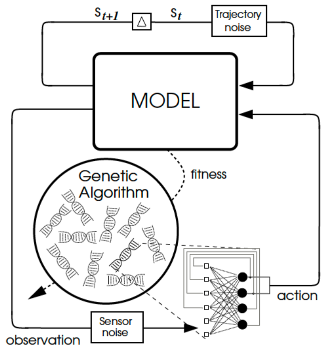
Rocket Guidance Network



Idea: Adding Noise to Encourage Robust Control

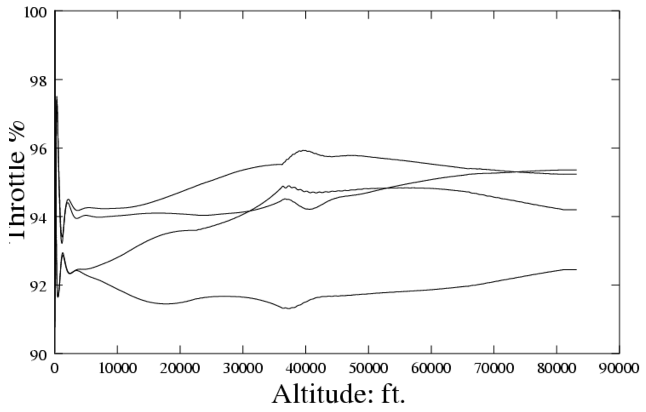
- One approach to robust control is adding trajectory noise.
- Trajectory noise forces the controller into situations where it must recover.
- This method is more effective than sensor noise because it doesn't confuse the agent.





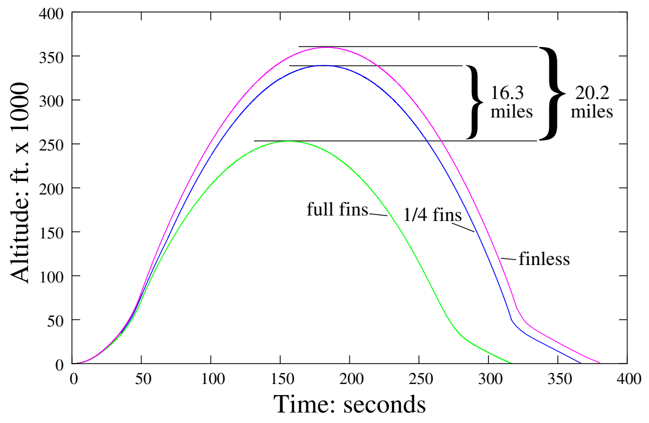
Results: Control Policy

- Accurate control in the beginning.
- Flies through atmospheric disturbance later.



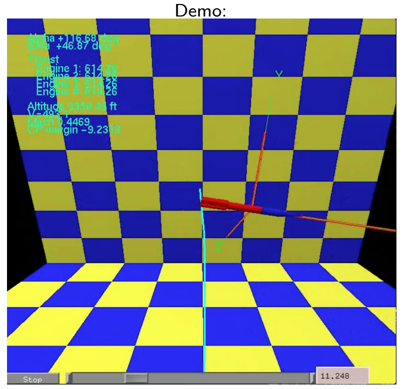
Results: Apogee

- Flies 20 miles higher without fins!
  (much of it coasting in thin air)



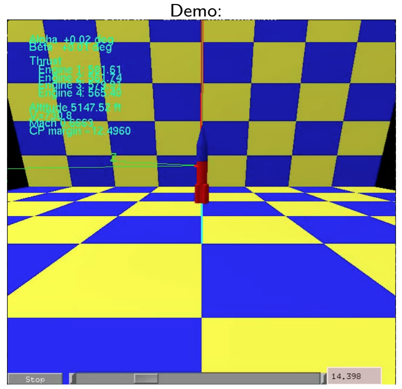
Finless Rocket Control Demo

Demo:



No active stabilization

Demo:

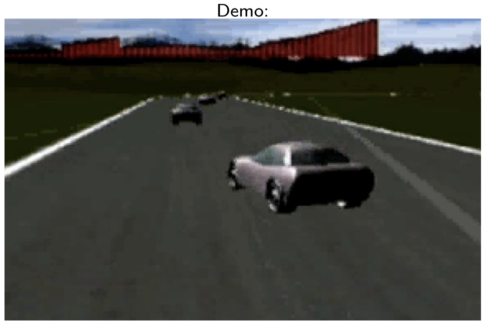


Evolved active stabilization

Challenge: Generalizing to Novel Situations

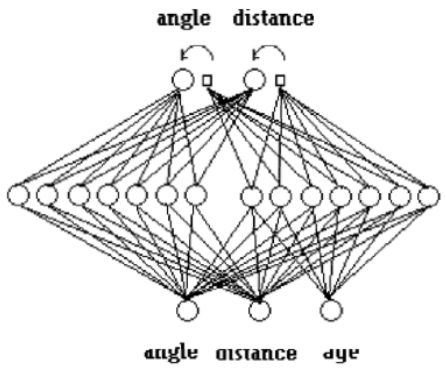
- Even with robust control, handling significant changes remains a challenge.
  - Training on every possible scenario is not feasible.
  - Need to come up with systematic approaches to extrapolate.

Demo:



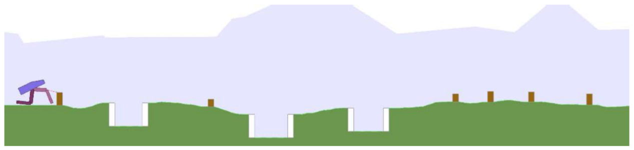
Idea 1: Teacher Networks for Enhanced Learning

- Teacher networks generate learning targets for controllers that learn via backprop.
- Teachers are evolved based on how well the controller performs after training.
- E.g. in creating a controller that forages for food:
  - With extra input for the age of the controller.
  - Optimal training inputs do not correspond to correct targets!
  - Instead, they create maximally effective learning experiences



Idea 2: Coevolution of Problems and Solutions

- In some cases, problems and solutions can be coevolved together, encouraging robust behavior.
- E.g. POET: coevolution of obstacle courses and runners.
- It starts with simple obstacle courses and gradually complexifies them as agents evolve better behaviors.
- This process leads to more general and robust solutions

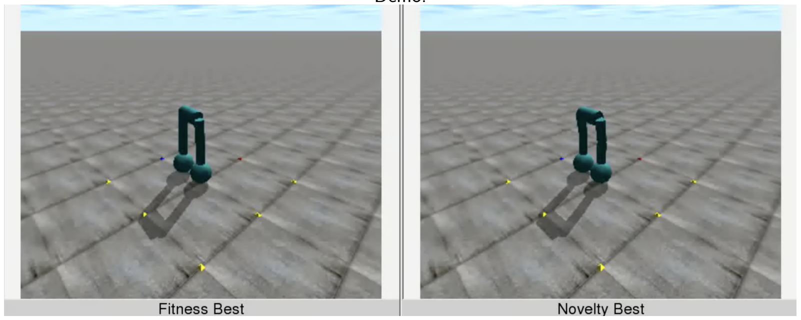


https://youtu.be/D1WWhQY9N4g?si=tmSrFmD8GNeNvA6L

Idea 3: Novelty Search

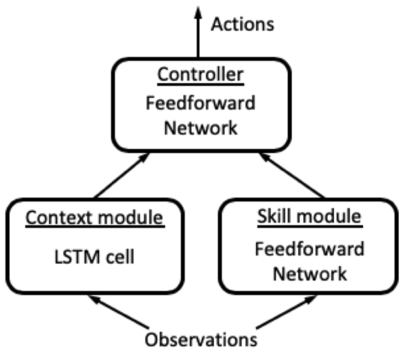
- Novelty search rewards diversity in behavior rather than just goal achievement.
- This method encourages exploration, leading to more generalized and robust solutions.
- Example: Novelty search discovered a dynamic, fast bipedal walk, while fitness-based search failed.

Demo:



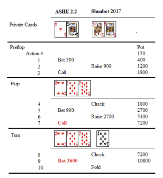
Idea 4: Modeling the Context Explicitly

- The system can be designed with three components:
  - Skill network: Takes actions.
  - Context network: Models the environment.
  - Decision network: Uses context to modulate skill actions.
- This allows the controller to adapt actions based on the environment.

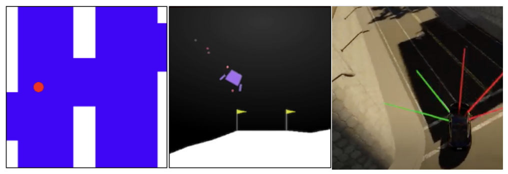


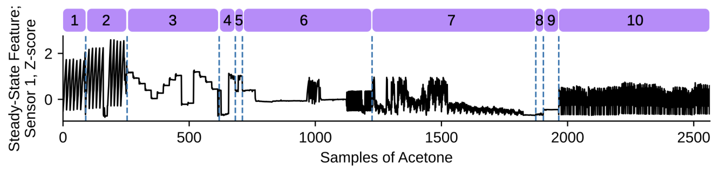
Context in Various Domains

- Opponent modeling in poker
  - Learn basic game play against canonical opponents
  - Track play by novel opponents; modulate play accordingly
  - Can generalize to much better opponents
- Context+Skill in physical games
  - Evaluated in FlappyBall, LunarLander, CARLA
- Tracking continuously changing environments
  - E.g. modeling sensor drift in odor recognition

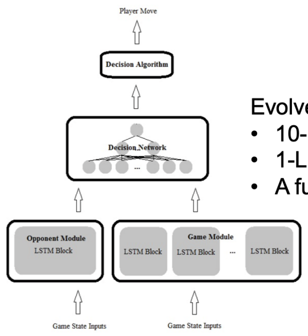


		ASHE 2.2	Slumbot 2017	
Private Cards				
Preflop				Pot
	Action #			150
	1	Bet 300		400
	2		Raise 900	1200
	3	Call		1800
Flop				
	4		Check	1800
	5	Bet 900		2700
	6		Raise 2700	5400
	7	Call		7200
Turn				
	8		Check	7200
	9	Bet 3600		10800
	10		Fold	





Adapting to Novel Opponents in Poker



Evolve weights of poker-playing NN

- 10-LSTM Game Module integrates over each game
- 1-LSTM Opponent Module integrates over each opponent
- A fully connected Decision Network makes moves

TEXAS
The University of Texas at Austin

Cognizant

Adapting to Novel Opponents in Poker (2)

Opponent	Evolved LSTM	Slumbot 2017
Scared Limper	999	702
Calling Machine	46114	2761
Hothead Maniac	42333	4988
Candid Statistician	9116	4512
Random Switcher	8996	2102
Loose Aggressive	20005	2449
Tight Aggressive	509	284
Half-a-Pro	278	152
Slumbot 2017	19	

Adapts strategy dynamically according to opponent

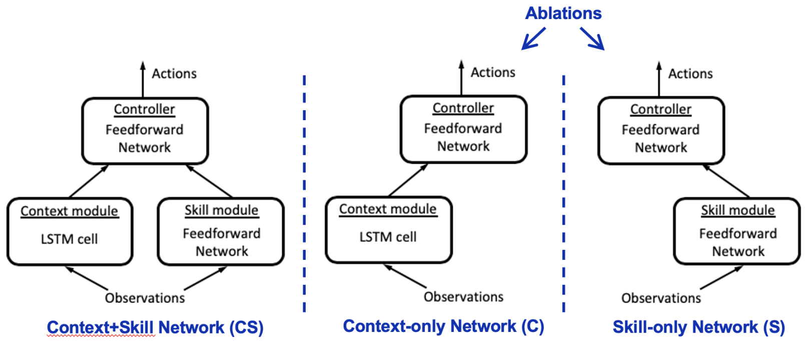
- Exploits weaknesses better than Slumbot (in mBB)
- Ties against Slumbot (although evolved with only weak opponents)

Can cope robustly with novel game play

TEXAS
The University of Texas at Austin

Cognizant

Adapting to Changing Worlds in Physical Games

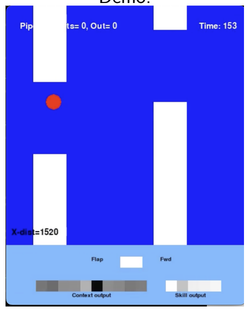


TEXAS
The University of Texas at Austin

Cognizant

The FlappyBall Domain

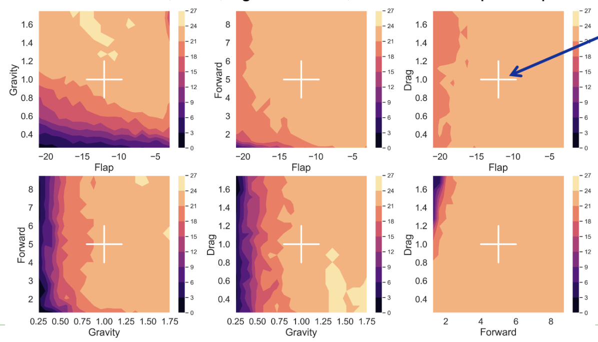
Demo:



- Extension of Flappy Bird: FlapFwd, Drag
- **Inputs**: 6 numerical state values
  - Vertical position, distance to next pipe
  - Horizontal and vertical velocity
  - Height of the upper and lower pipe
- **Outputs**: select FlapUp, FlapFwd, glide
- **Objectives**:
  - Safety: Don't hit pipes, ceiling, ground
  - Performance: Fly fast
- **Task Variation:**
  - Strength of Gravity, Drag, FlapUp, FlapFwd

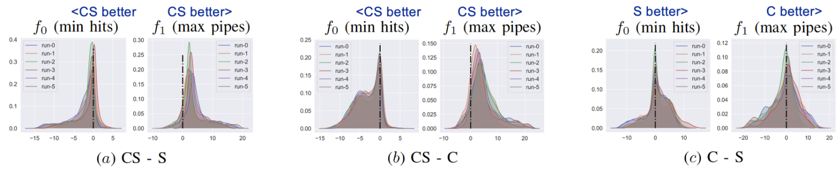
Illustration of Extrapolation

FB Performance; CS-S; Lighter is better; variation across pairs of parameters



- Training tasks distributed on the white cross
- Testing tasks distributed outside the cross: Require interpolation and significant extrapolation

Generalization in FlappyBall

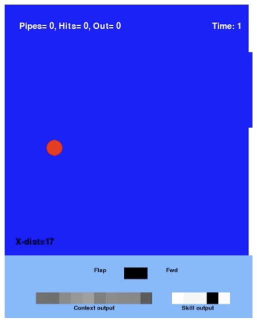


- Best networks from 5 independent evolutionary runs evaluated in new tasks
  - Effect of Gravity, Drag, FlapUp, FlapFwd varied +/- 75% (instead of +/-20% during evolution)
  - All parameters varied simultaneously; 10,000 tasks created randomly
- CS performs better than S and C in both objectives
- S is better than C in safety, the same in performance

Example Behaviors in FlappyBall

▶ Extrapolated conditions: F=-7.0, G=0.58, Fwd=8.75, D=0.58

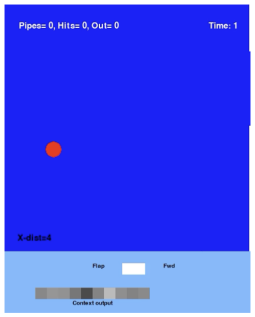
Demo:



Context+Skill Network (CS)

Pipes = 21
Hits = 0

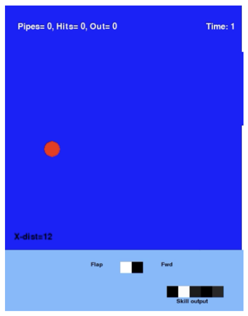
Demo:



Context-only Network (C)

Pipes = 15
Hits = 6

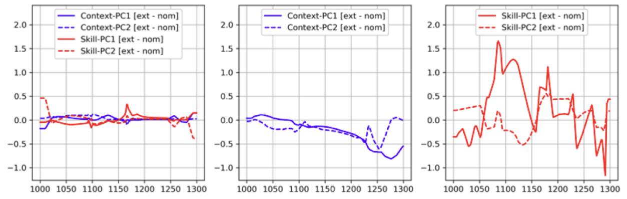
Demo:



Skill-only Network (S)

Pipes = 16
Hits = 5

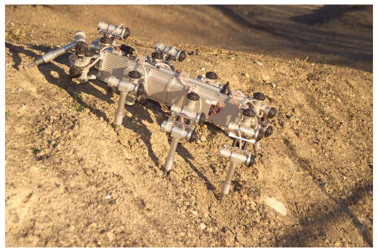
Modulation by Context

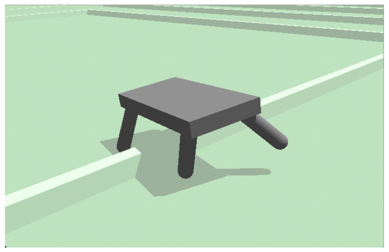


- Output of Context and Skill modules mapped to 2D with PCA
- Difference in an extrapolated task and the nominal task plotted
- Differences are smaller in CS than in C-only and S-only
  - Decision network needs to deal with less variance
  - Easier to generalize
  - CS evolves to make new tasks look more familiar
- Allows coping robustly in novel situations

Multilegged Walking

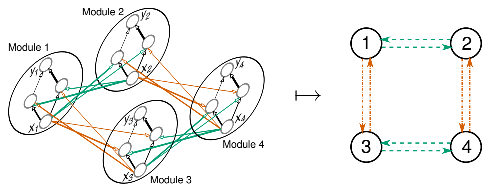
- Navigate rugged terrain better than wheeled robots
- Controller design is more challenging
  - Leg coordination, robustness, stability, fault-tolerance, ...
- Hand-design is generally difficult and brittle
- Large design space often makes evolution ineffective





Idea 5: Symmetry Evolution Approach

- Symmetry evolution approach
  - A neural network controls each leg
  - Connections between controllers evolved through symmetry breaking
  - Connections within individual controllers evolved through neuroevolution



Versatile, Robust Gaits

- Symmetric gaits such as trotting and pacing are easier to evolve initially.
- Different gaits on flat ground
  - Pronk, pace, bound, trot
- When facing more complex terrains, symmetry-breaking allows for more adaptive gaits.
  - For example, an agent might switch from a bound gait to a trot to overcome obstacles.
  - This automatic adaptation makes control more robust across various terrains.

Demo:

Different gaits

Demo:

Obstacle field

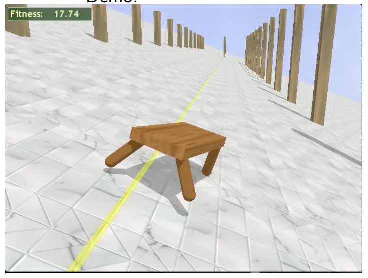
Innovative, Effective Solutions

- As challenges increase, symmetry can be broken to evolve more complex gaits.
- Asymmetric gait on inclines
  - One leg pushes up, others forward
  - Hard to design by hand

Demo:

Handcoded

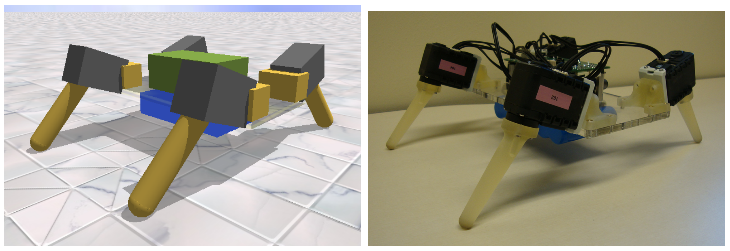
Demo:



Evolved

Challenge: Transferring Solutions to Physical Robots

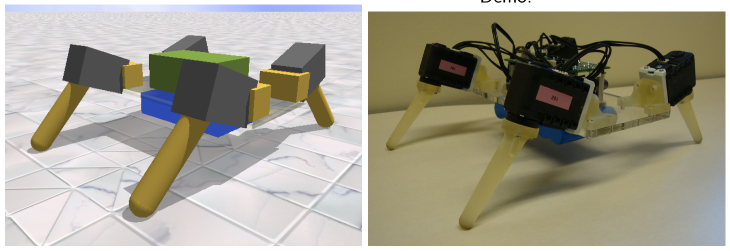
- Simulations are clean and deterministic.
- The real world is noisy, nondeterministic, and includes external factors.
- Transfer from simulation to reality is difficult but critical.


Simulated Real

Transferring to Quadruped Robot

- A robot custom-built at Hod Lipson's lab (Cornell U.)
  - Standard motors, battery, controller board
  - Custom 3D-printed legs, attachments
  - Simulation modified to match
- General, robust transfer
  - Noise to actuators during simulation
  - Generalizes to different surfaces, motor speeds

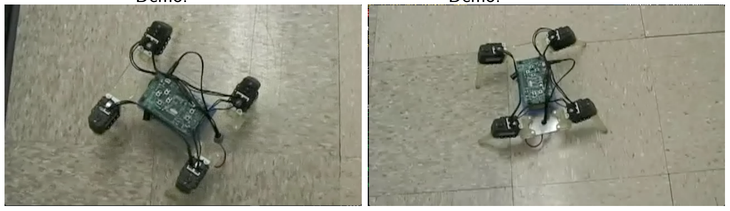
Demo:


Simulated Real

Compensating for Damage

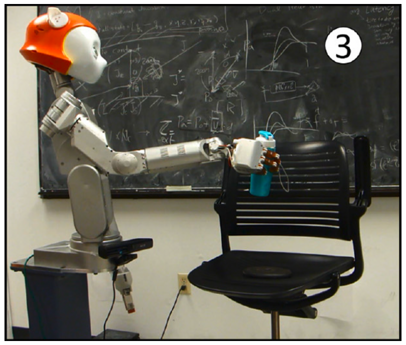
- Neuroevolution evolves controllers that can cope with imperfections and even take advantage of them.
- Example: Evolved asymmetric gait for a four-legged robot with one inoperative leg.
- This shows that neuroevolution transfers well to physical robots and can solve unexpected issues.

Demo: Demo:


Handcoded Evolved

Simulating Physical Challenges in Neuroevolution

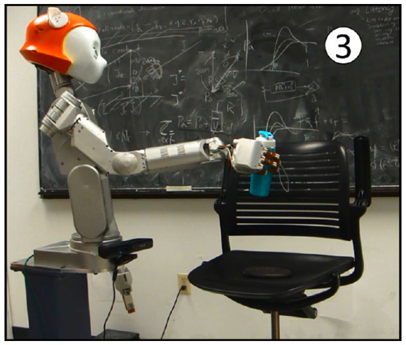
- Simulations can be extended with factors like wind, friction, and uneven terrain.
- Stochastic noise can be added to simulate imperfections in sensors and effectors.



Dreamer robot

Recent Advances in Robotics Simulators

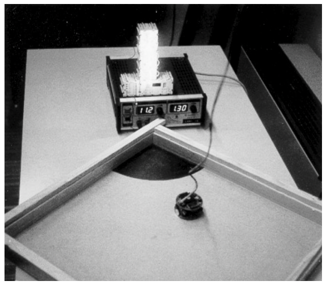
- Modern robotics simulators have become highly accurate, supporting direct transfer to physical robots.
- Example: NEAT with Graspit! simulator for robotic grasping, transferred to the Dreamer robot's Mekahand.
- Controllers can handle sensor inaccuracies, novel objects, and imprecise computation.



Dreamer robot

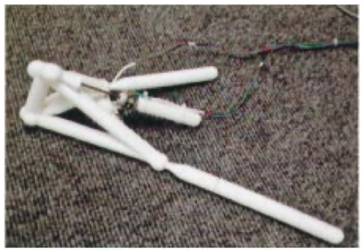
Evolutionary Robotics: Evolving Control in Hardware

- Evolutionary robotics emerged in the 1990s to evolve controllers and sometimes hardware directly.
- Example: Evolving homing behavior in the Khepera mobile robot.
- Neural networks developed an internal topographic map to navigate efficiently.



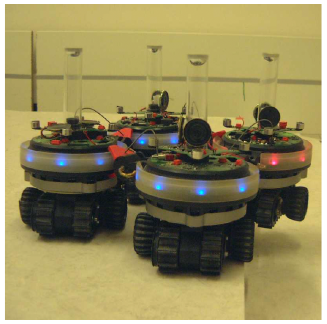
Coevolving Morphology and Control

- Neuroevolution can coevolve both the controllers and the hardware.
- Example: Locomotion starts with eel-like robots and evolves into legged designs.
- This process creates more robust gaits than evolving directly for legged robots.
- GOLEM: Hardware designs and controllers coevolved in simulation, then 3D printed and tested physically.


https://youtu.be/qbUyWZZ_a9g

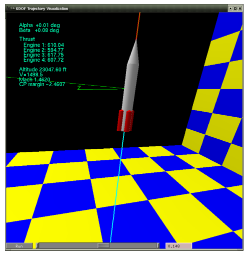
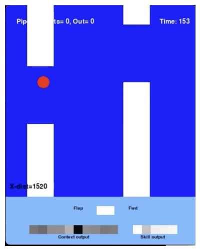
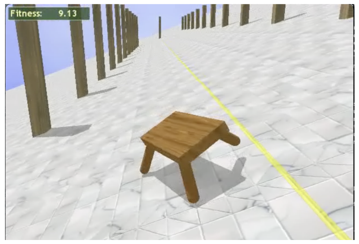
Swarm Robotics: Evolving Collective Behavior

- Swarms of robots exhibit collective behavior that single robots cannot.
- Example: Robots forming a train to traverse gaps that individual robots cannot cross.
- Neuroevolution can evolve both collective and individual behaviors for the swarm.


https://youtu.be/i3ernrkZ91E

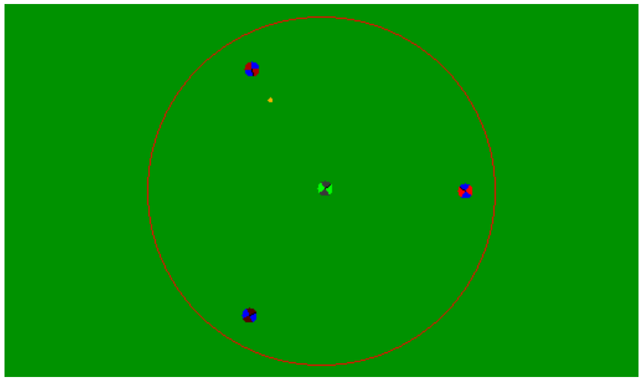
Conclusion: Evolving Robust Control

- Robust control is essential for generalization and adaptability in complex environments.
- Techniques like noise injection, coevolution of controllers with teachers and problems, novelty search, explicit context representation, and symmetry help build this robustness.
- Advanced simulators, noise injection, and coevolution with hardware make transfer possible.

From Low-level Control to High-level Strategy

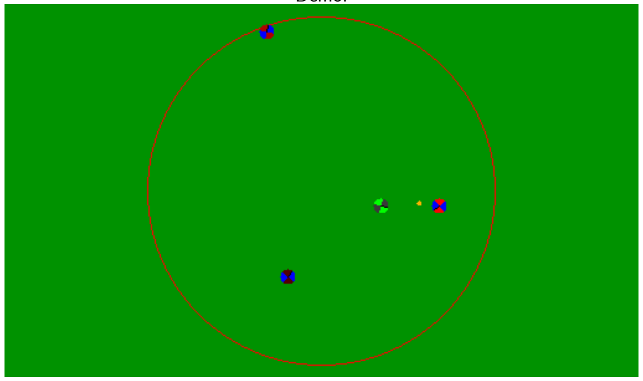
- Low-level control: Adjusting single behaviors (e.g., moving a leg faster).
- High-level strategy: Coordinating multiple behaviors.
- Example: Keepaway soccer: GetOpen, Intercept, Hold, EvaluatePass, Pass
- Challenge: Switching between behaviors effectively.



Direct Evolution

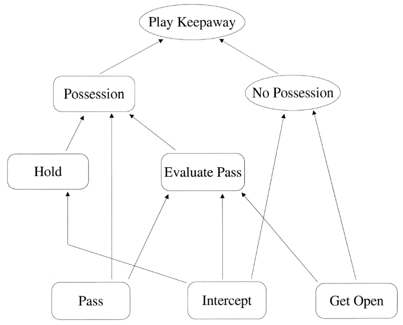
- Mapping sensors directly to actions
- Difficult to separate behaviors
- Ineffective combinations result

Demo:



Coevolution Approach

- Evolve a separate network for each behavior
- A decision tree to decide which network to activate


Play Keepaway
Possession
No Possession
Hold
Evaluate Pass
Pass
Intercept
Get Open