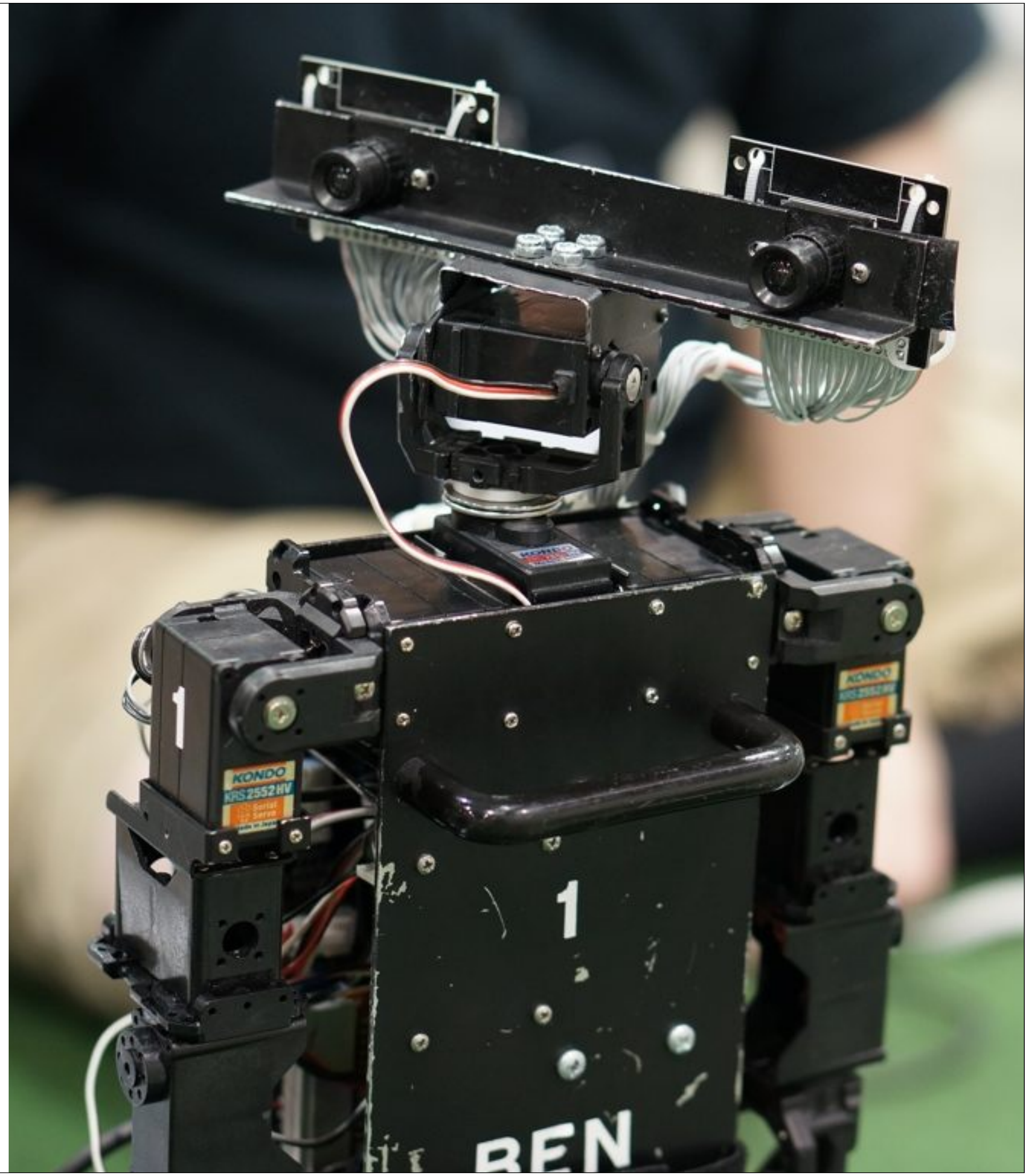
Online Training of Robots and Multirobot Teams

Sean Luke
陸子荀
Department of
Computer Science
George Mason University



About Me

- Associate Professor
  Department of Computer Science
  George Mason University

- **Interests**
  Multiagent Systems
  Machine Learning
  Multirobotics
  Stochastic Optimization and Evolutionary Computation
  Simulation

- **Software (and Hardware)**
  ECJ *Evolutionary Computation Toolkit*
  MASON *Multiagent Simulation Toolkit*
  RoboPatriots and FlockBots *Robot Architectures*

My Current Multiagent Systems Problem

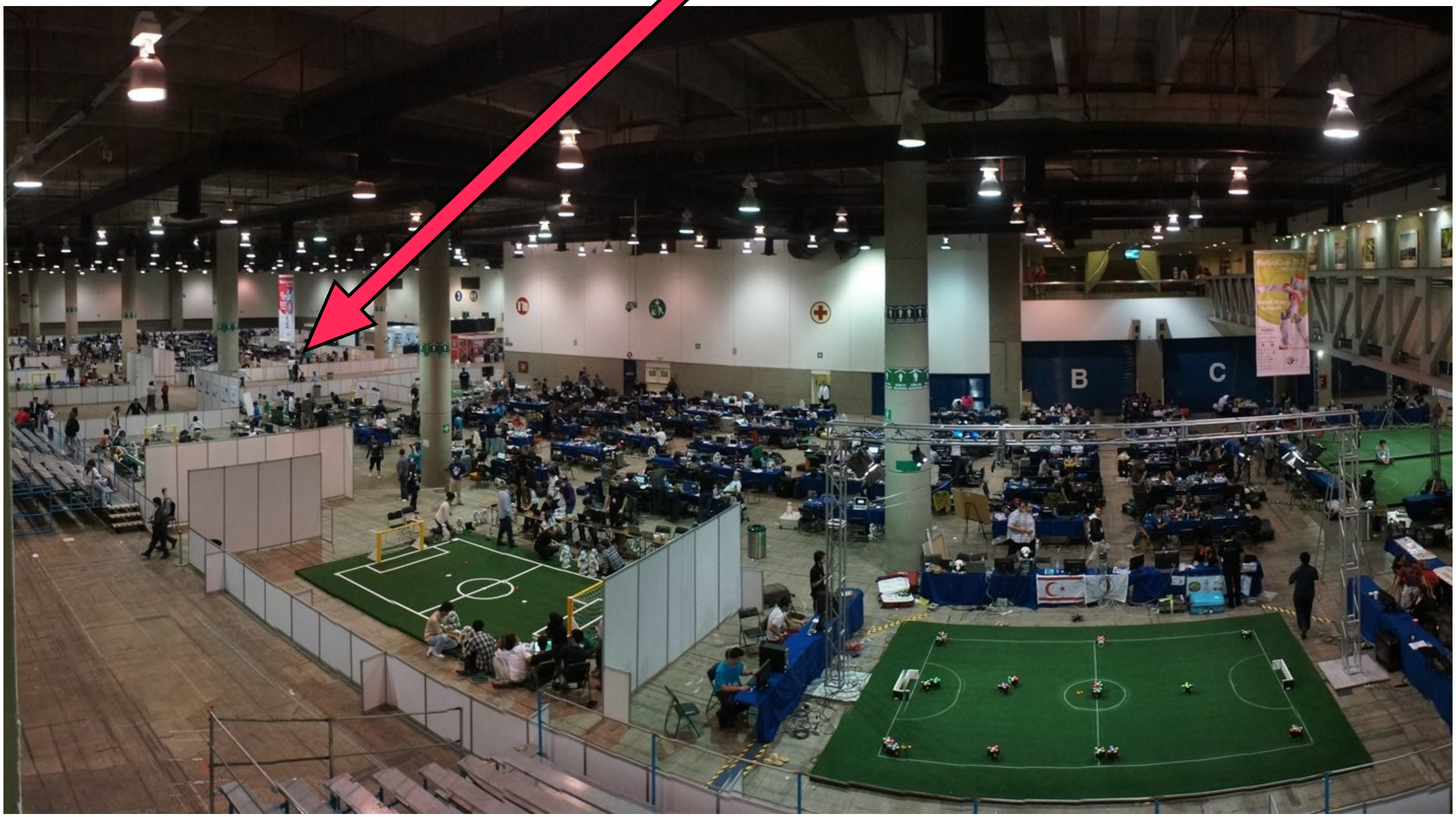
Topics in This Talk

- **RoboCup**
- **Multiagent and Multi-robot Systems**
- **Pheromone-based Robotics:** An Example of Emergent Behavior
- **HiTAB:** Single-Agent and Single-Robot Training
- **Unlearning:** Dealing with noise in single-agent training
- **Behavioral Bootstrapping:** training a flat (leaderless) swarm
- **M-HiTAB:** Hierarchical Multiagent and Multi-Robot Training

RoboCup 2012 Mexico City

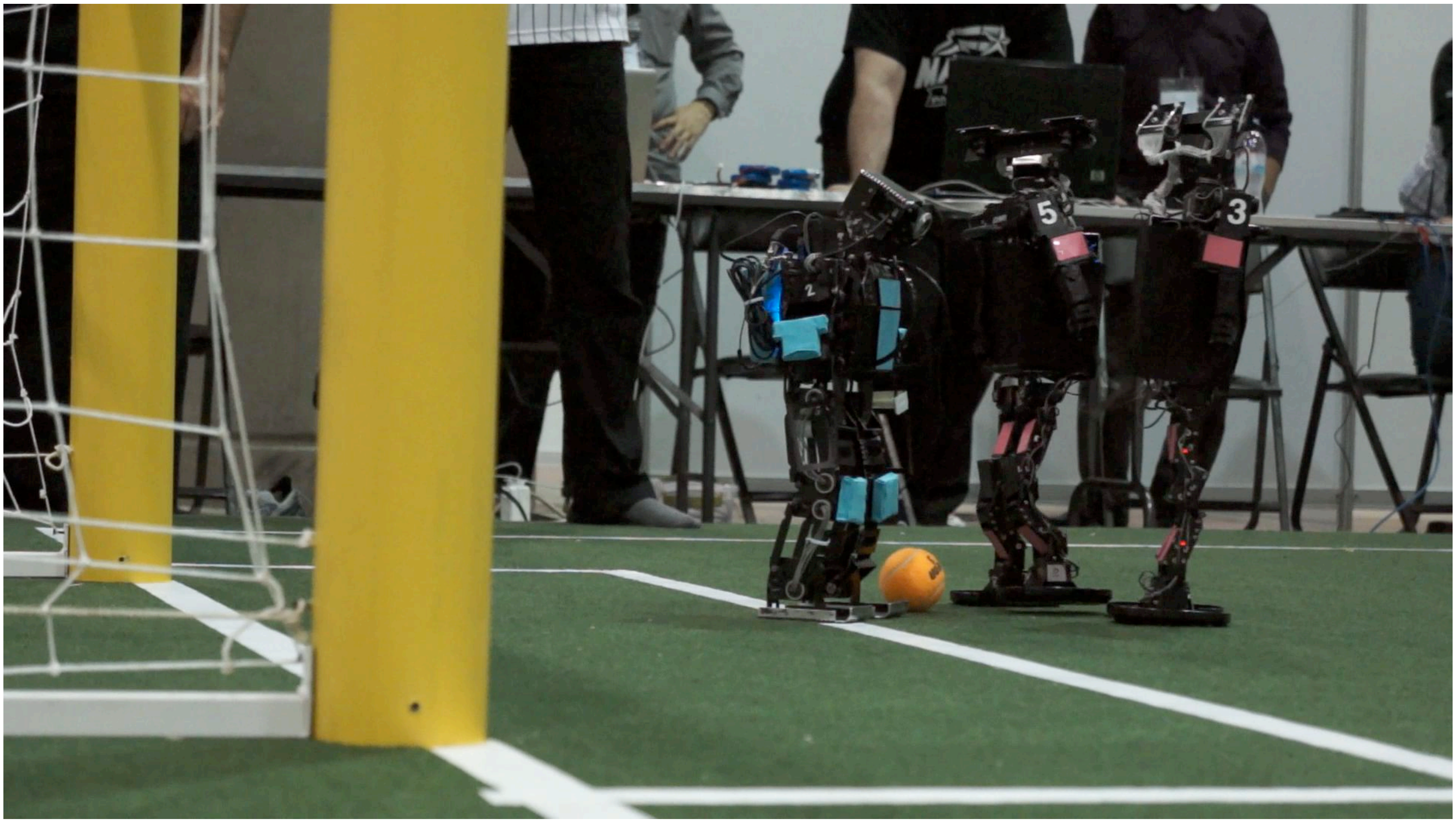
RoboCup 2012

George Mason University




MARKOVITO
TEAM

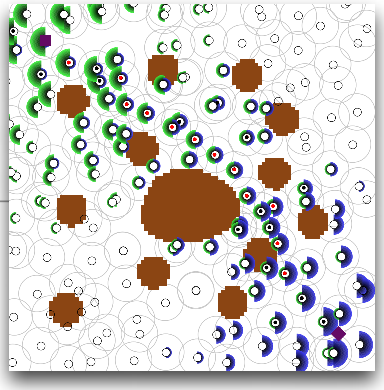
RoboCup 2012 GMU: Pink Osaka: Blue



A Multiagent System (or MAS)

- **Agent:** an autonomous entity which iteratively manipulates its environment in response to feedback received from the environment.
- **Multiagent System:** a system of ... you know ... multiple agents.
    - Agent interaction
    - Emergence
- **Distributed Systems Problem:** given multiple processors and resources under your control, solve a given task.
- **Multiagent Systems Problem:** given multiple agents with **major constraints on communication or mutual knowledge**, solve a given task.

Why Develop / Simulate MAS?



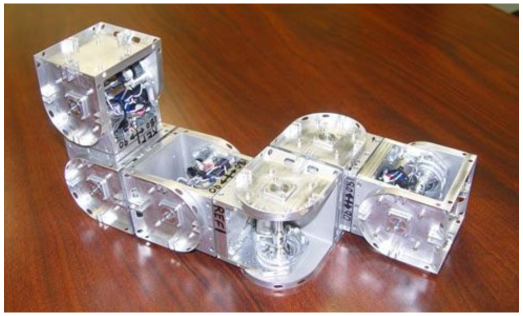
- **Science:** MAS **models** can help us make predictions and test hypotheses when it would be impossible, immoral, or unrealistic to perform real-world tests.
  - Biology, Physics, **Social Sciences**
  - *Goal:* accurate replication of existing phenomena
- **Engineering:** MAS **methods** help us test new techniques or inventions.
  - Games, Animation, Networked Agents, **Multirobotics**
  - *Goal:* optimization or demonstration of new methods

Multiagent Systems (for Engineering)

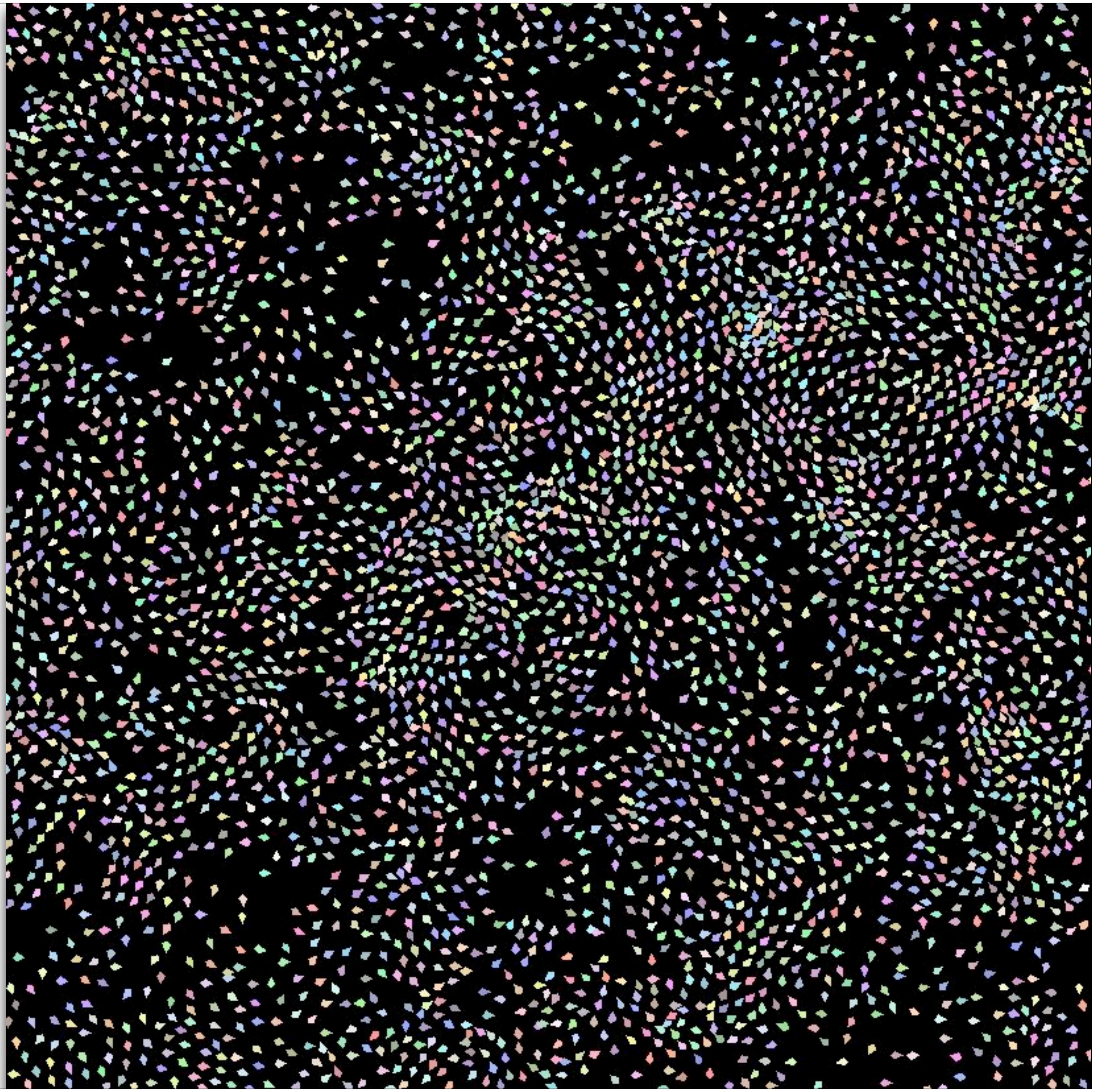
- **Agent or Robot <u>Teams</u>**
  Small Numbers,
  Often Heterogeneous
  Lots of Communication/Interaction
  Global Communication

- **Agent or Robot <u>Swarms</u>**
  Large Numbers

- **<u>Modular</u> Robots**
  A Robot Consists of *Modules* (the “Agents”)
  Moderate Numbers, Usually Homogeneous
  Communication via Internal Network
  Is this really a multiagent system?



Multiagent Systems Are Very Complex



The Multiagent Systems Design Space is **Big**

- **Factors in the complexity of a Multiagent Systems Design:**
    - Number of Agents
    - Complexity of Agent Behavior and Capability
    - Heterogeneity of Agents
    - Degree of Agent Interaction
    - Communication Complexity
    - Designing Robust and Cost-Effective Designs
- **This becomes very complicated very quickly**

Tradeoffs (in Multirobotics)

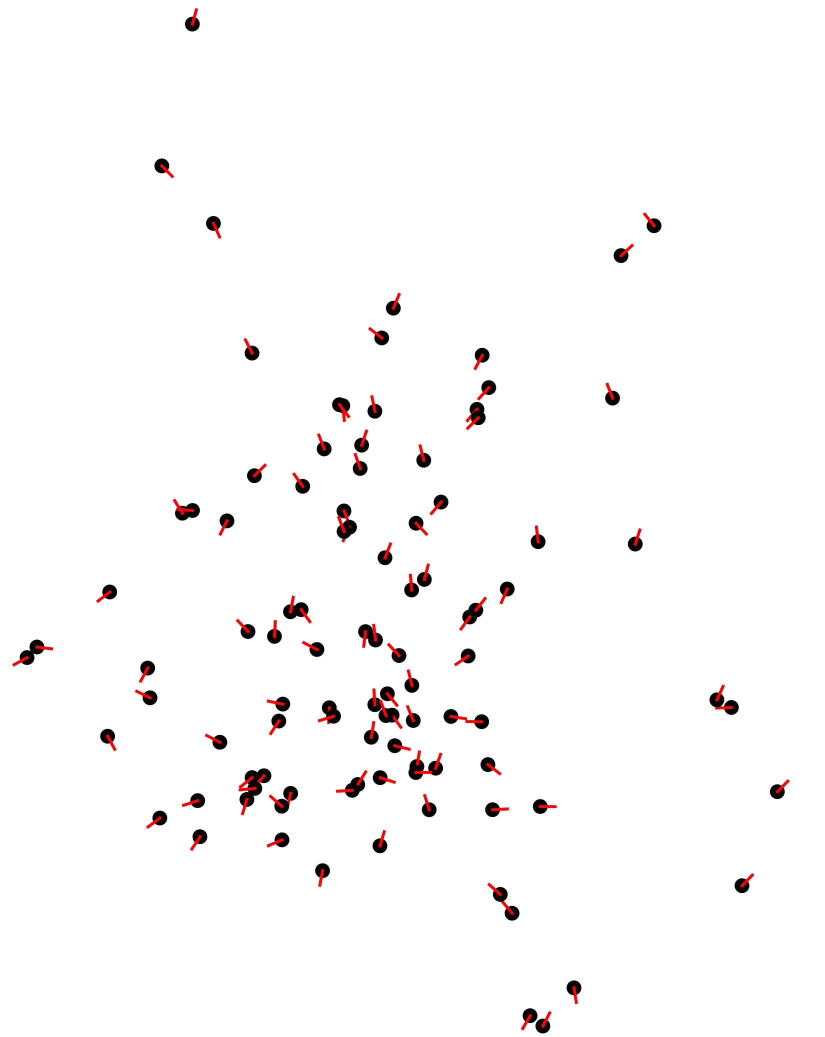
- **Agent or Robot <u>Teams</u>**
  Small Numbers (often 2 or 3!)

- **Agent or Robot <u>Swarms</u>**
  Homogeneous
  Little Communication/Interaction
  Local Communication
  Very Simple Behaviors

- **The more agents,
  the simpler they get!**

Emergent Behavior

- Simple **Micro-Level** Behaviors
- Complex Emergent **Macrophenomena**
- Can you **Predict** the Macrophenomena given the Micro-level Behaviors?
- Complexity Theorists **Love** Emergence
- Multiagent / Multirobot Designers **Hate** Emergence



Can you predict this?

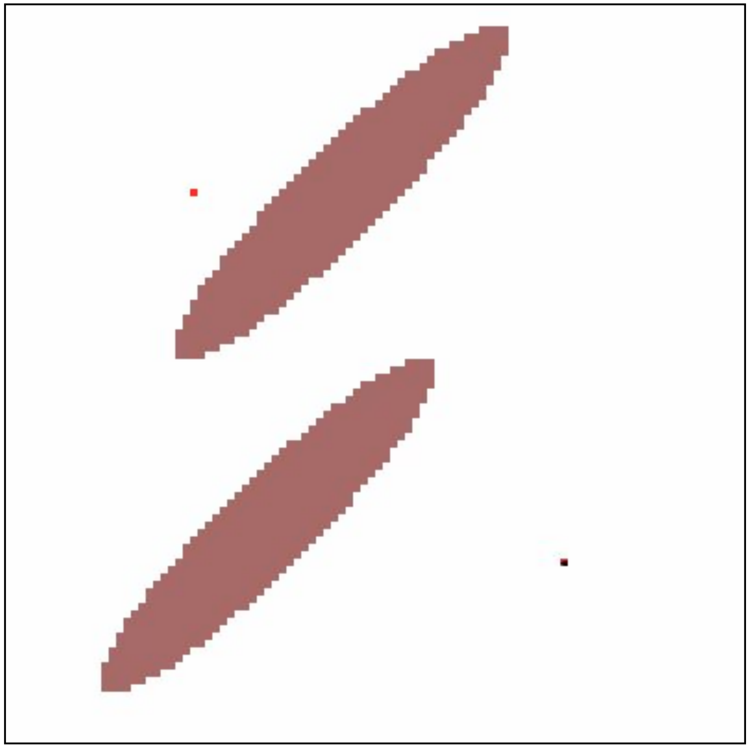
Example: Ant Pheromone Foraging

- **Most ant pheromone literature uses a single pheromone**
  (Biologically plausible, but bad algorithms)

- **We use multiple pheromones**
  2 in this example: *Food* and *Nest*

- **Each ant *follows* one pheromone but *updates* another.**

- **Each ant is in a *state*, which determines which pheromones it follows / updates.**



Example: Ant Pheromone Foraging

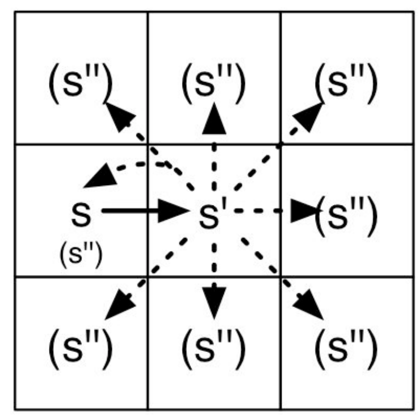
States:	**Follow Pheromone:**	**Update Pheromone:**
Looking for Food	Food	Nest
Looking for Nest	Nest	Food

- **Following:**
  An ant is in state s'
  Go to square s" with highest pheromone $U_p(s")$

$$s' = \operatorname*{argmax}_{s'' \in S''} U_p(s'')$$

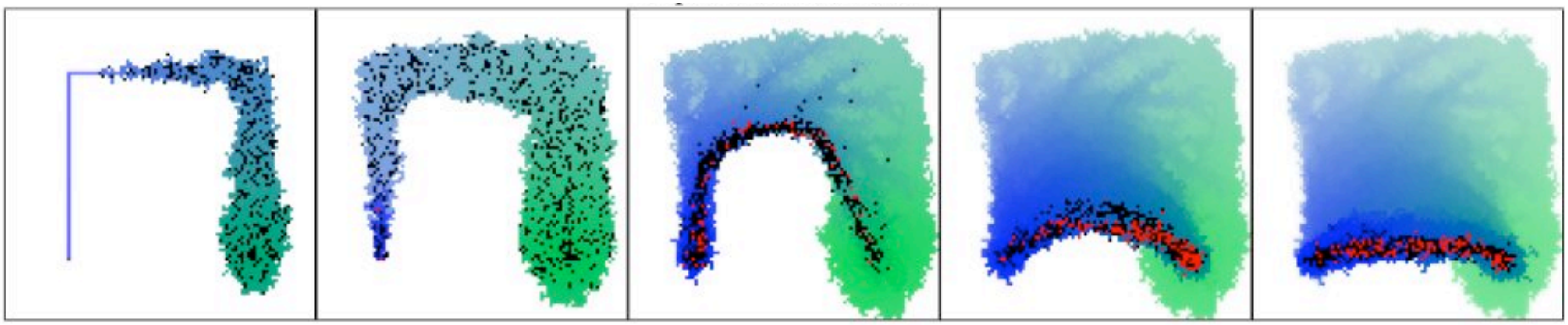
- **Updating:**
  An ant is in state s'
  Update $U_p(s')$
  Reward R(s') is received only if at nest / food

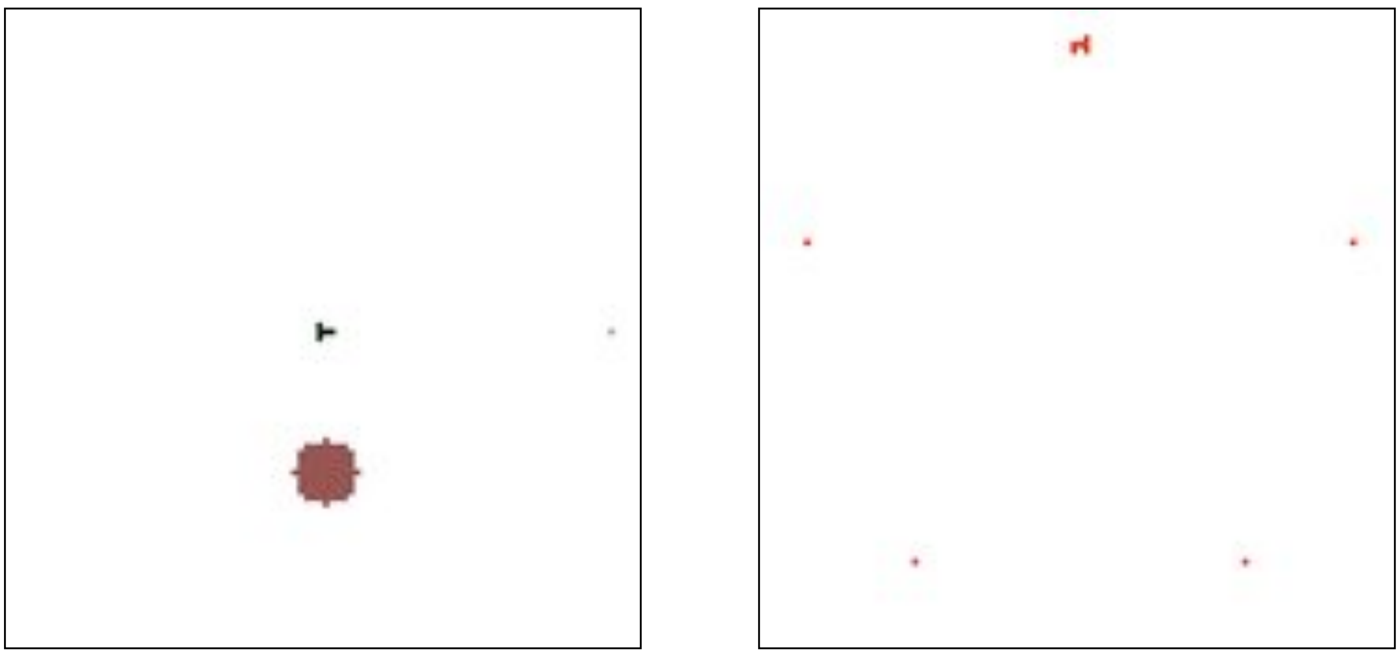
$$U_p(s') = R(s') + \gamma \max_{s'' \in S''} U_p(s'')$$



- **Form of multi-utility value iteration**

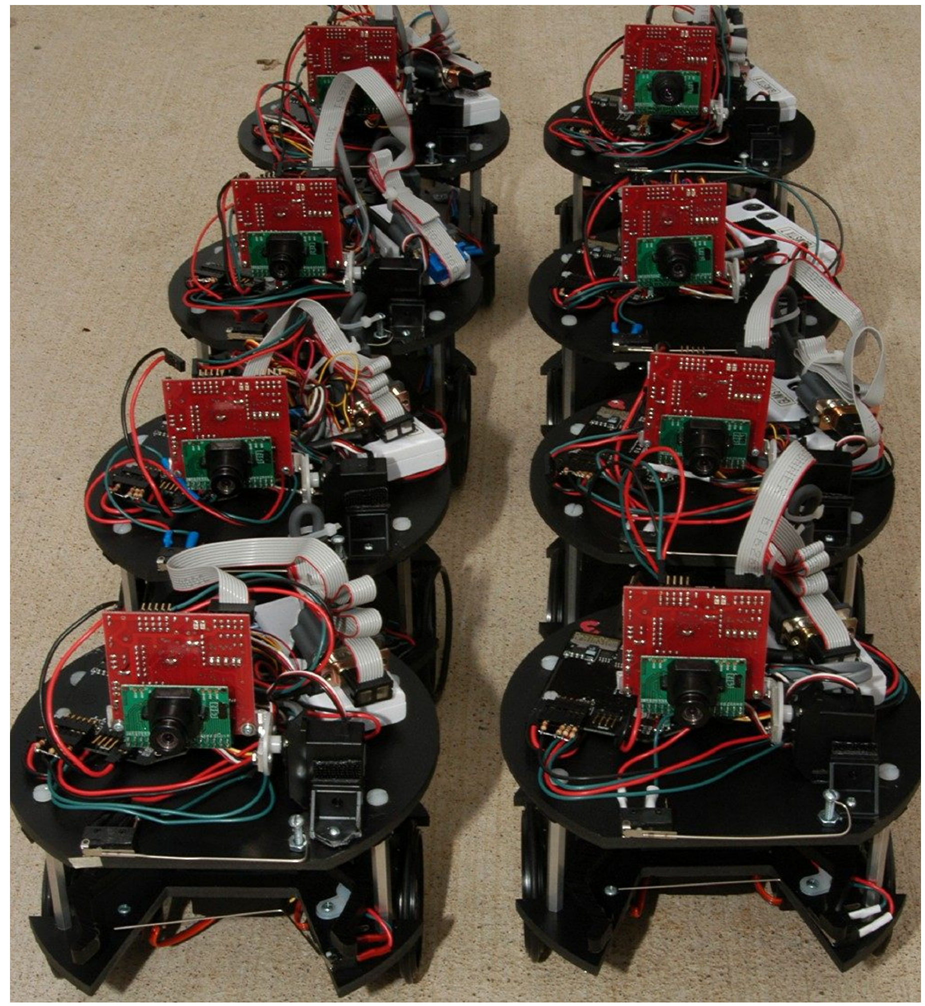
Example: Ant Pheromone Foraging



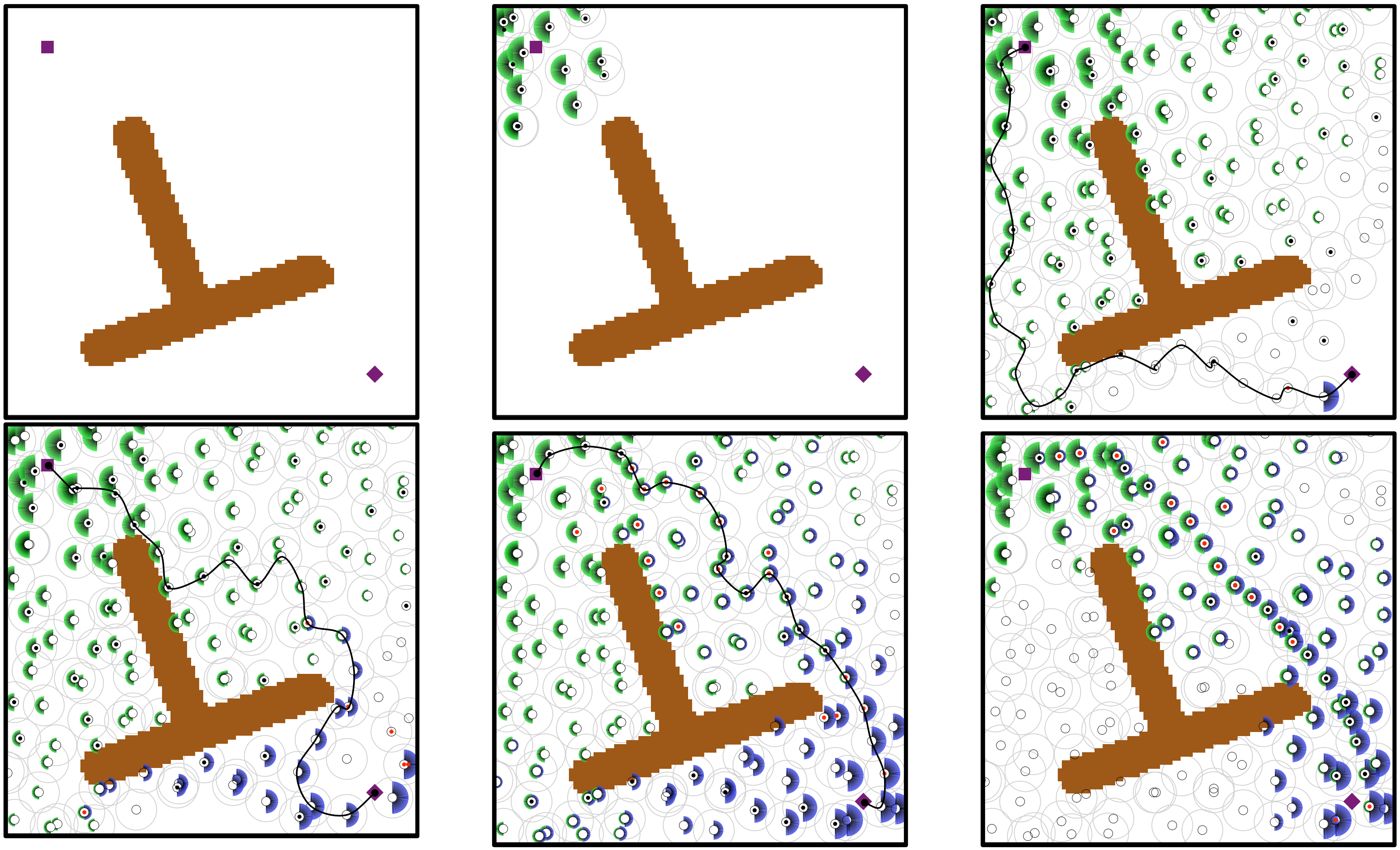


Example: Ant Pheromone Foraging **With Beacons**

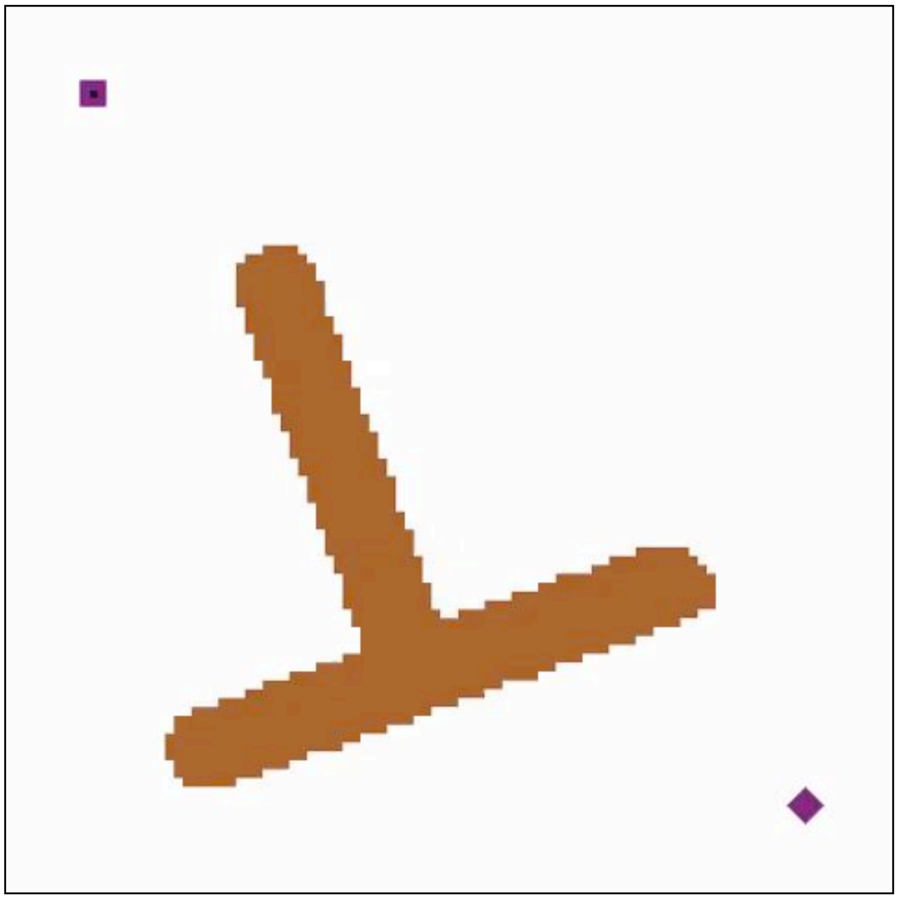
- The **Flockbots**
- Small (15cm diameter) differential drive robots capable of deploying, moving, and removing cans
- Cans contain **Sensor Motes** which act as movable pheromon

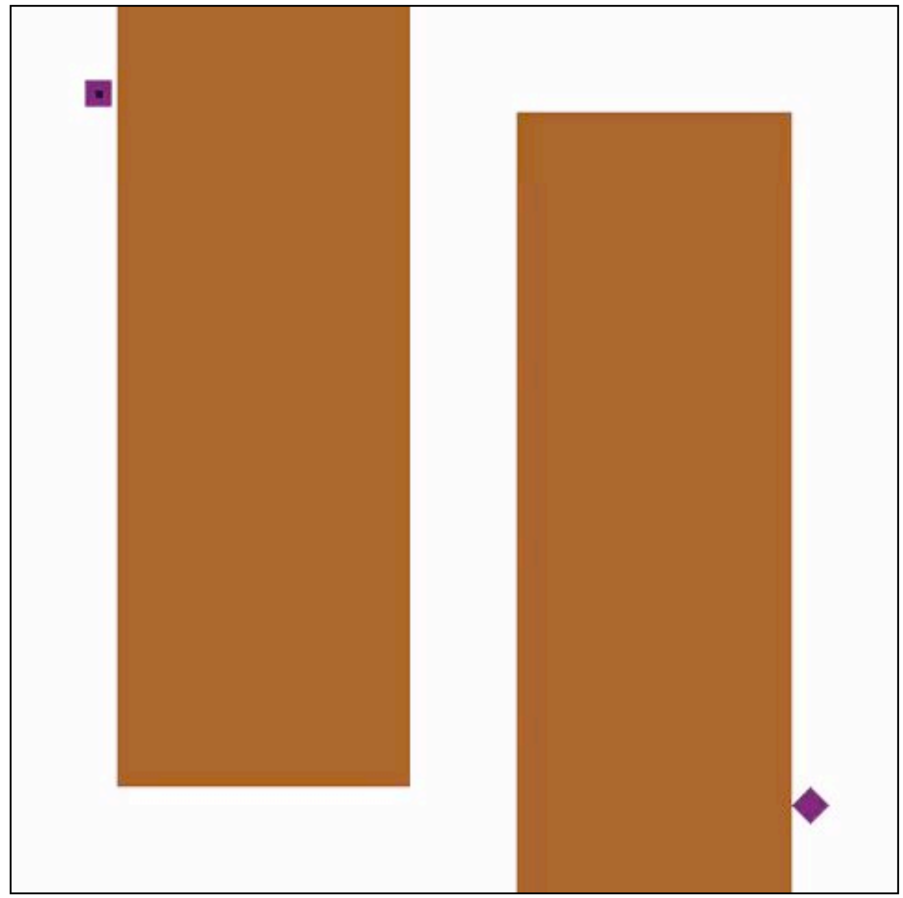


Example: Ant Pheromone Foraging **With Beacons**



Example: Ant Pheromone Foraging **With Beacons**





Agent Learning and Training

- **Machine Learning**
  Given a sample of data drawn from an environment, construct a model which explains the environment.

- **Agent Training**
  An agent is using machine learning, but there is a **trainer** present who observes the agent build and use its model, and suggests corrections.

- **Learning from Demonstration**
  A robot learns to do a task after being given sample data by a human. This is **training** only if the human iteratively updates the sample data to provide corrections or suggestions. It is also **very expesive.**

- **Our Research**
  1. Develop methods to do training of nontrivial **single agent** behaviors.
  2. Develop methods to do training of nontrivial **multiagent** behaviors.

Single and Multi-Agent Training with Few Samples

- **Single-Agent Training Challenge**
  The **Curse of Dimensionality.** The size of the training / learning space can be very large for complex behaviors, but the number of samples is very small.

- **Multi-Agent Training Challenge**
  The **Multiagent Inverse Problem.** Training multiple agents presents a difficult inverse problem which gets worse and worse with more agents, more interactions, and more complex behaviors.

Current Learning from Demonstration Systems

- **Learning Paths or Trajectories**
  Large numbers of samples
  Machine learning is easy

- **Learning Behaviors or Plans**
  Small numbers of samples
  Machine learning is very difficult

- We want to learn **sophisticated** behaviors based on a very **small** number of samples.

HiTAB (Single-Agent Training)

- **Goal**
  Train **complex, stateful** behaviors from a very **small** number of samples in **real time** on simulated agents or robots.

- **Difficulty**
  Curse of dimensionality. Robot behaviors can be complex, but we only have to train on a **small** number of samples.

- **Solution: Behavioral Decomposition**
  Manually break complex behaviors into simpler behaviors. Learn the simpler behaviors. Then learn their composition into the complex behaviors.

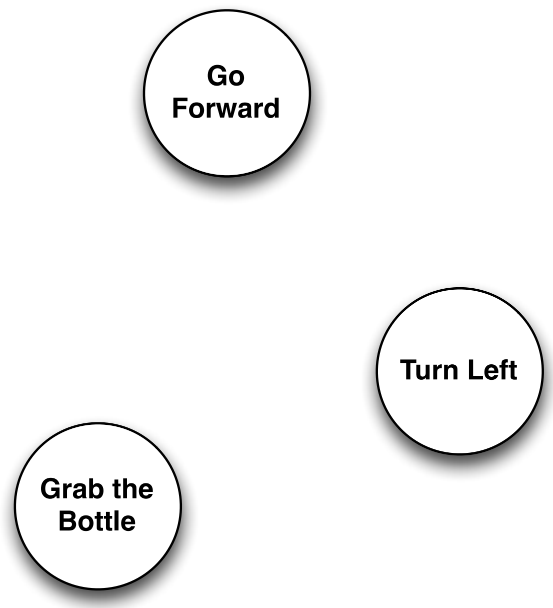
  This projects the complex behaviors' joint space into smaller, simpler spaces that are much easier to learn with few samples.

HiTAB Single-Agent Model

- **Hierarchical Finite-State Automata (HFA)** as **Moore Machines**
  - Each **Behavior** is a **State**
  - **Recursive** Behaviors may themselves be other automata
  - **Transitions** from State to State based on environment **Features**
  - **Parameterizable** "Go to X" rather than "Go to the Ball"
- **Each timestep**
  - Transition function is queried based on current environment features, possibly resulting in a new current state
  - Current state's behavior is pulsed one iteration

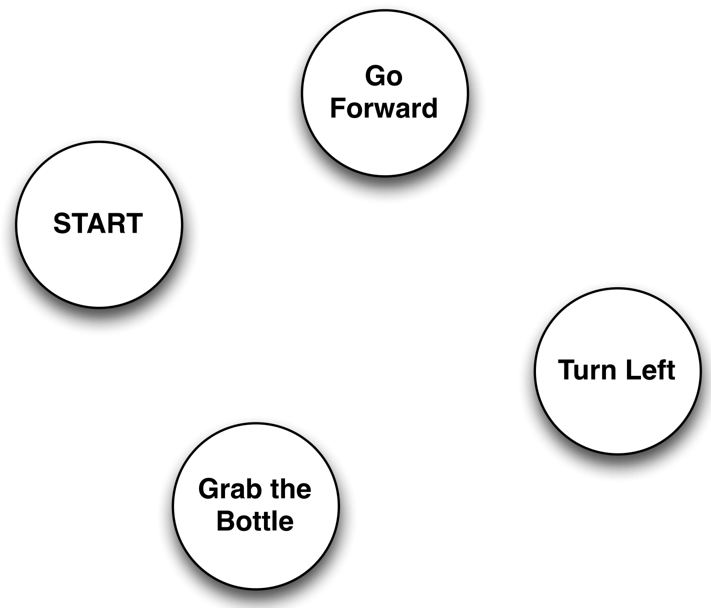
Moore Machines

- A Moore Machine is a **Finite-State Automaton** with:
    - A set of **states** corresponding to **behaviors**



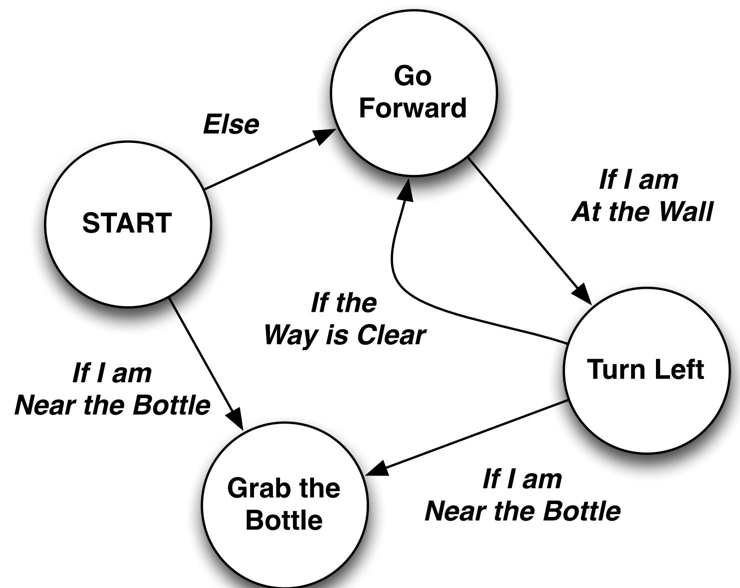
Moore Machines

- A Moore Machine is a **Finite-State Automaton** with:
  - A set of **states** corresponding to **behaviors**
  - A special **START state** (there are no end states)



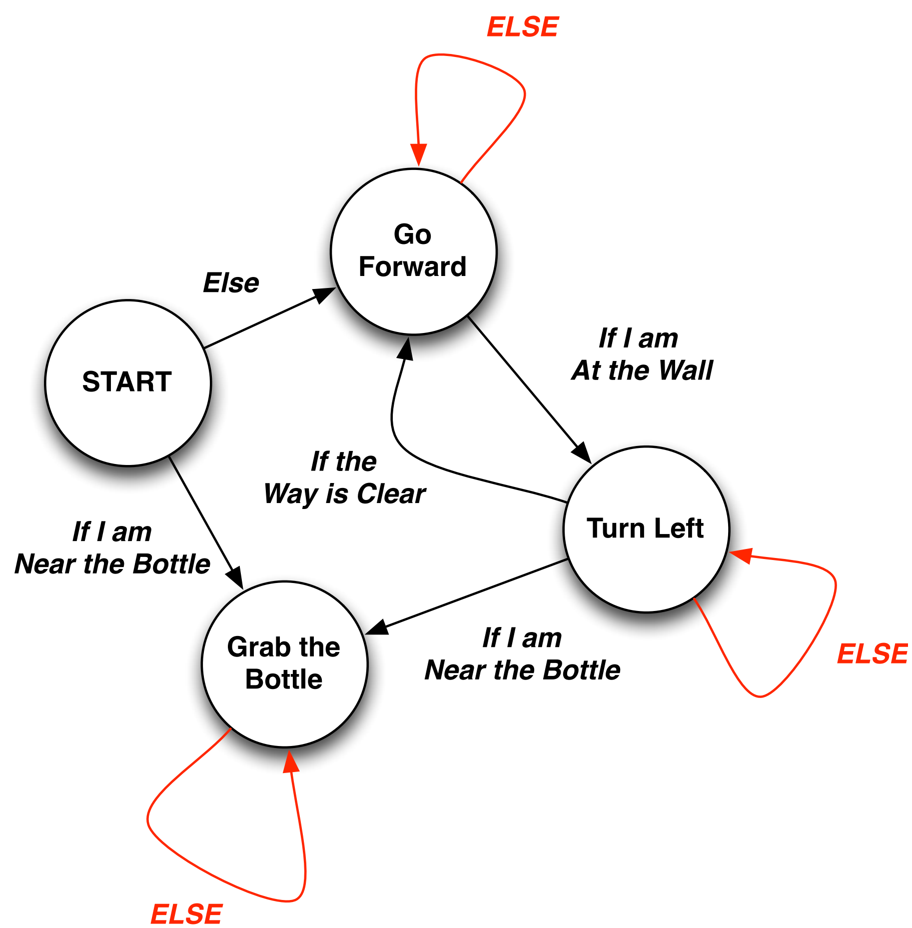
Moore Machines

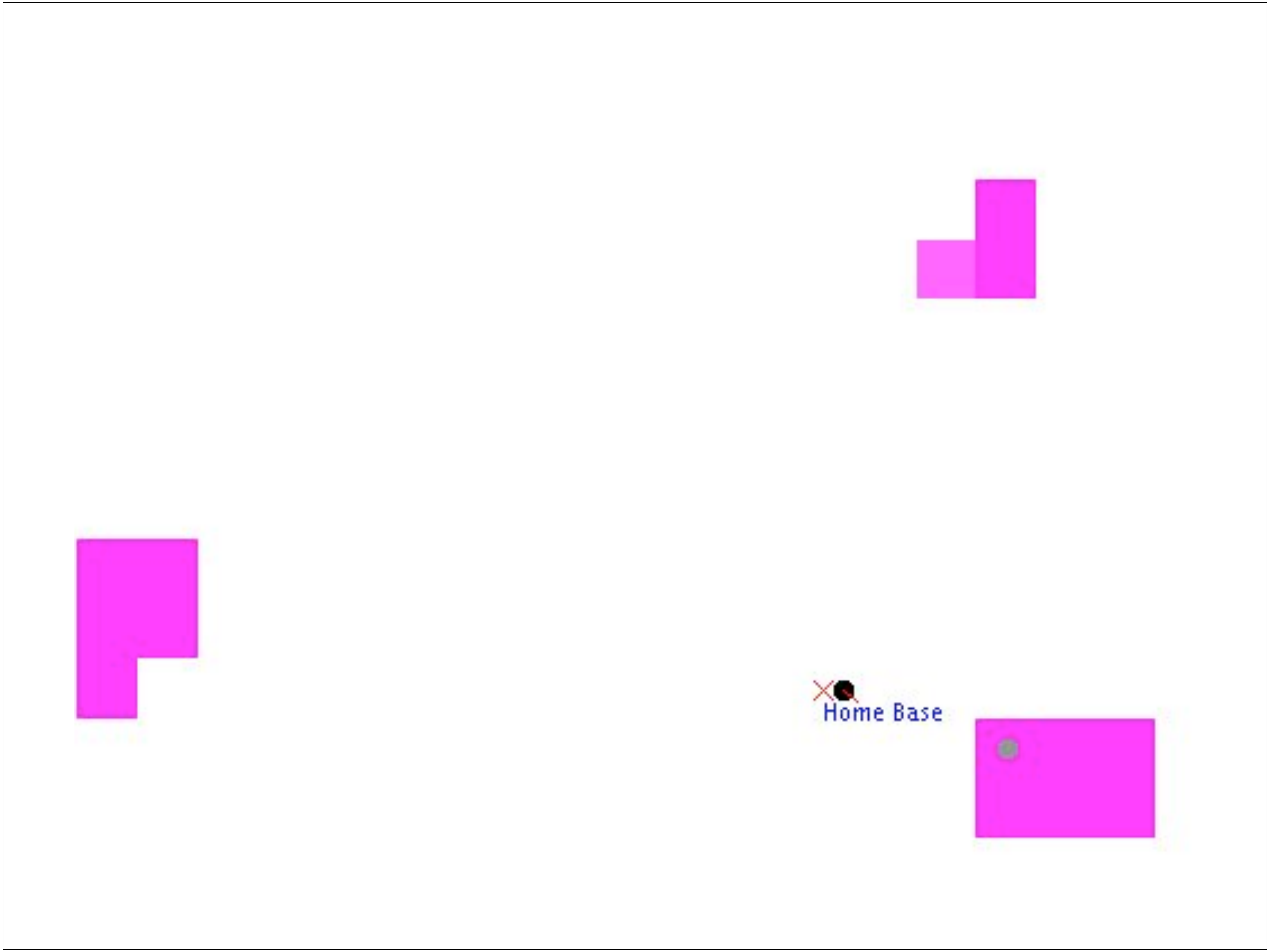
- A Moore Machine is a **Finite-State Automaton** with:
  - A set of **states** corresponding to **behaviors**
  - A special **START state** (there are no end states)
  - A set of **directed edges**
    - All edges leaving a state are called its **transition function**



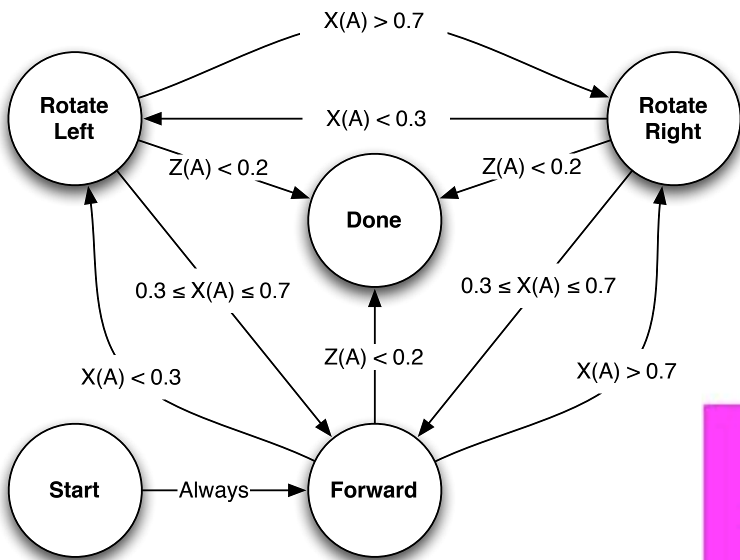
Moore Machines

- A Moore Machine is a **Finite-State Automaton** with:
  - A set of **states** corresponding to **behaviors**
  - A special **START state** (there are no end states)
  - A set of **directed edges**
    - All edges leaving a state are called its **transition function**
  - **No self-edges** (they are implied and mean "else")

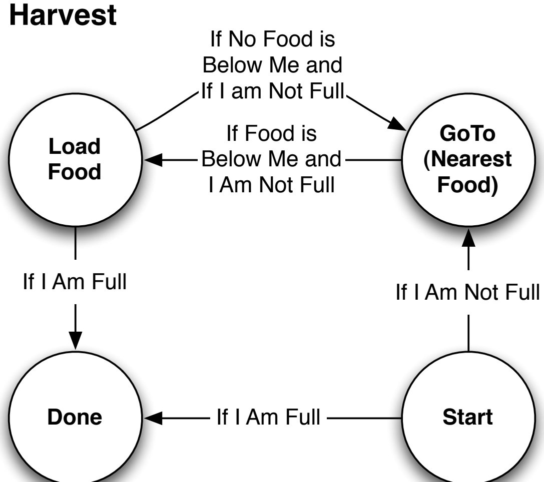



Home Base

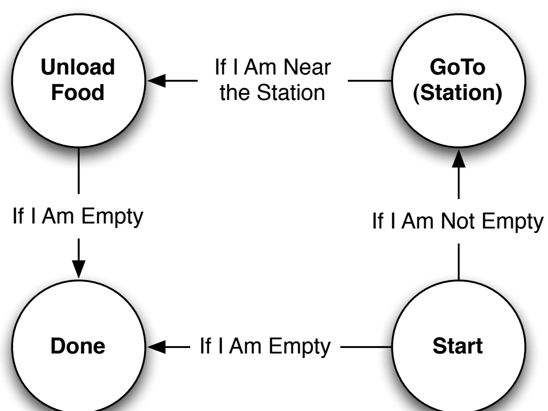
GoTo (A)



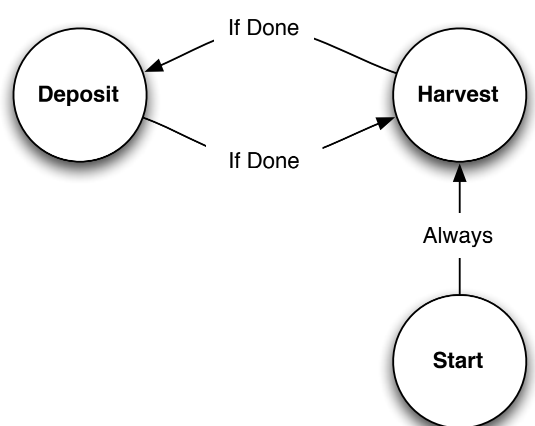
Harvest



Deposit



Forage

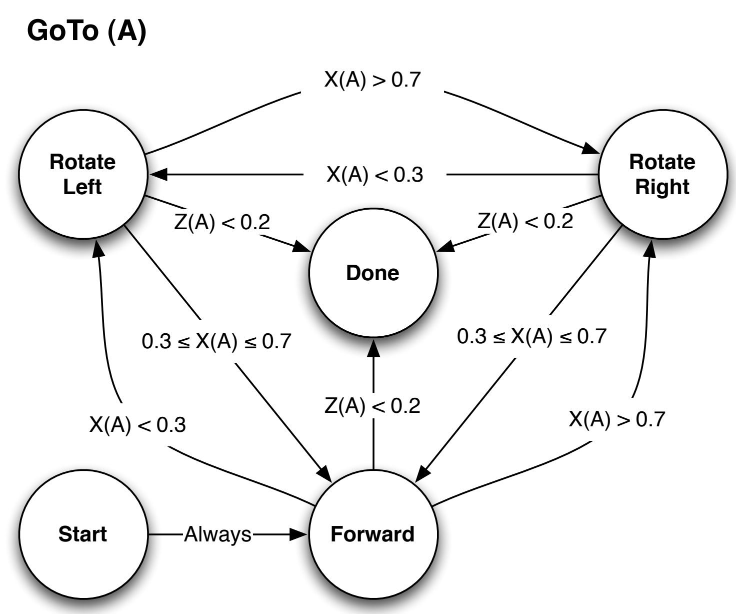


Training a HiTAB Automaton

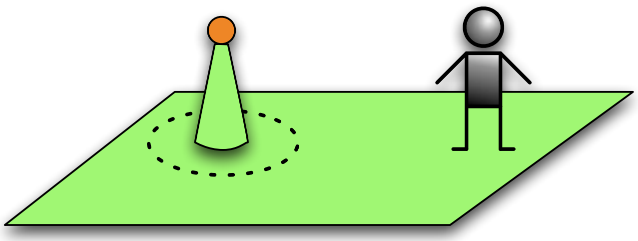
- For each state *s,* we learn the **transition function** *T(s,f)* for edges leaving *s.*
- **Gather Data.** When the user transitions to a new state/behavior, log: [*old behavior, current feature vector, new behavior*]
- **Build *T(s,f)* ⇒ *s'* for each state *s***

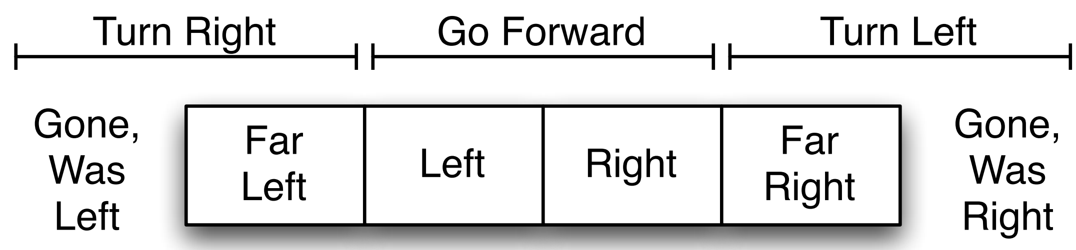
  Gather all samples [*s, f, s'*] starting with *s*
  Reduce to just *f* ⇒ *s'*

  This is just a classification task
- **Delete all unused states, add to library**

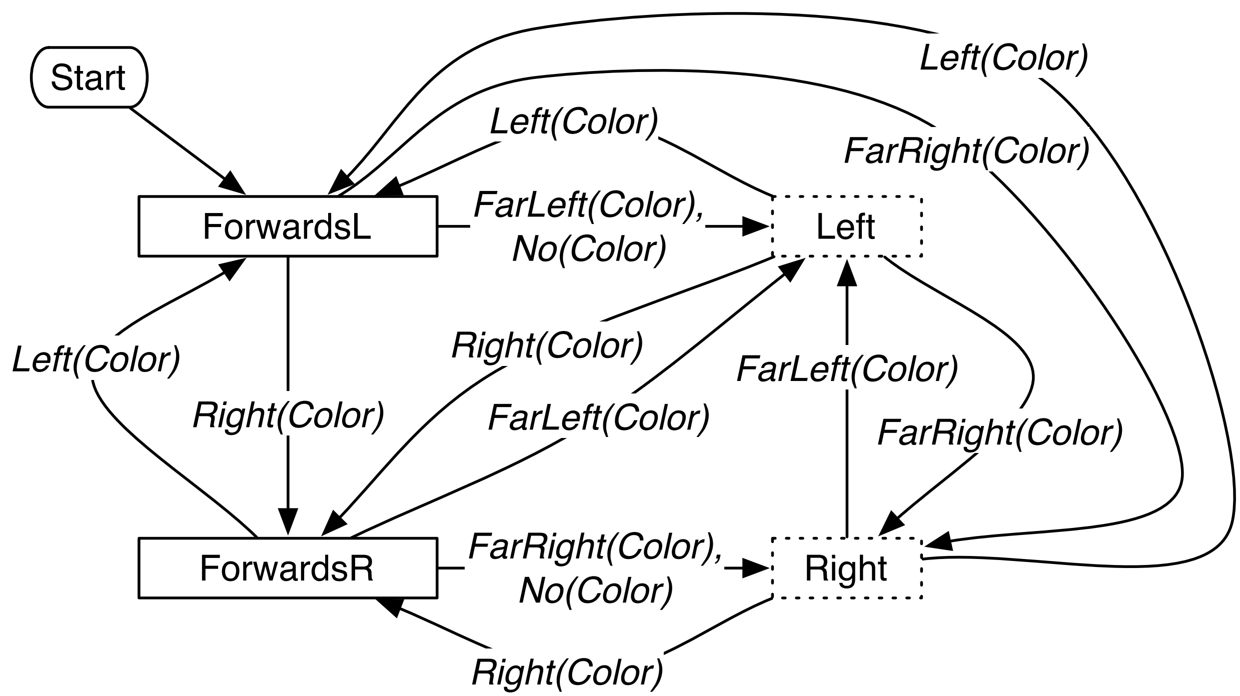


Statefulness Is Important

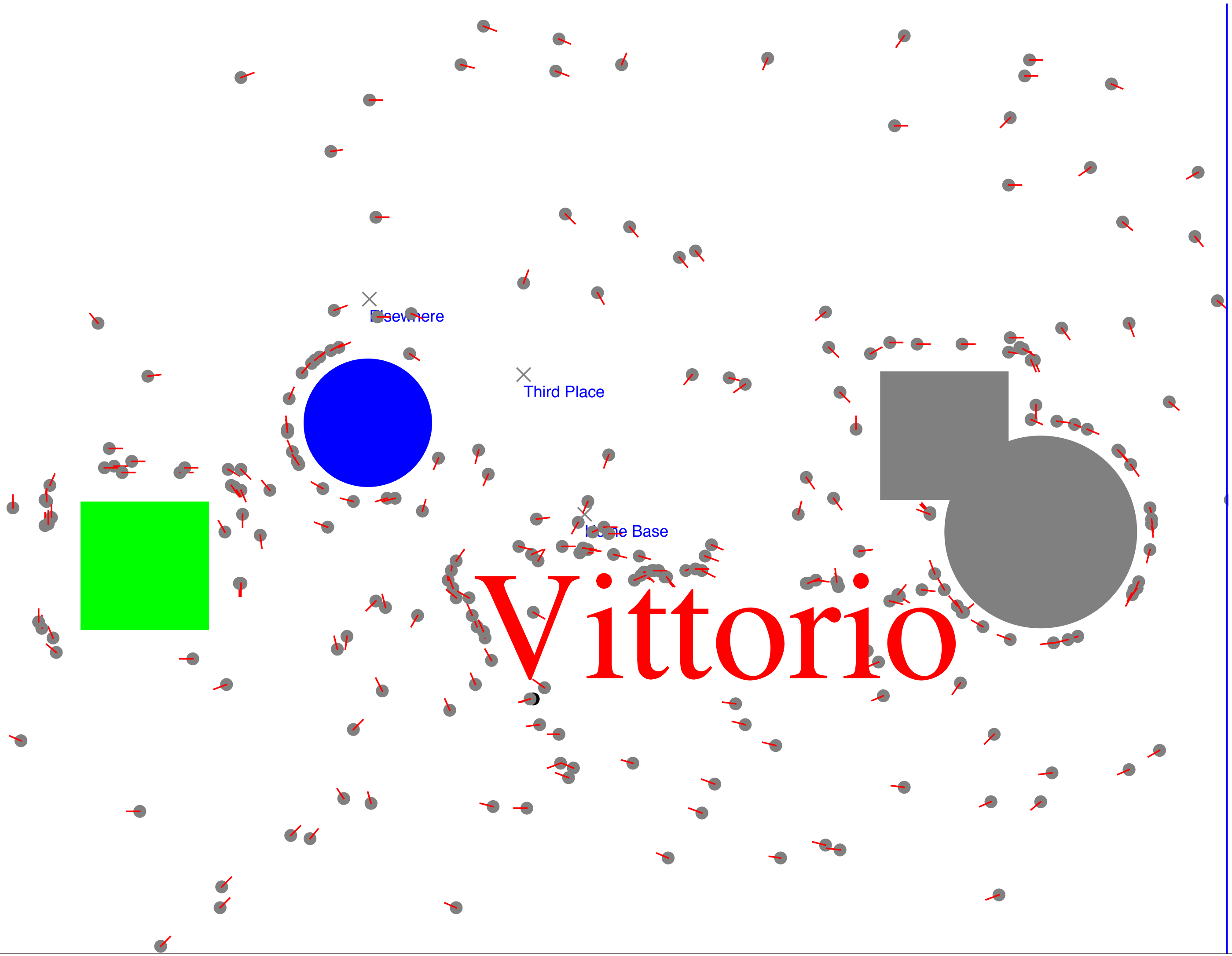




- A **Policy π(*f*)→*a*** is not a sufficiently rich representation to learn many robot behaviors.
- We learn a finite state machine transition function **T(*s*,*f*) →*s***



Demonstration...



Unlearning: Training Despite Noise (IJCAI 2013)

- **Situation: Training**
  When the agent performs its learned behavior incorrectly, the trainer **corrects the behavior.**

- **Problem**
  How do we use the corrective information to update the model?

- **Complication**
  We have a **very small number of samples.** (Samples are precious).

- In typical machine learning (with many samples), we'd just add the corrective samples to our sample set and re-learn the model.

- In **unlearning**, we use the corrective samples to **detect and remove noisy sample data.**

Unlearning

- **We have:**
  S Original sample set (with some noisy samples)
  M Original learned model from S
  C Set of corrective samples

- **We produce:**
  S' Revised sample set (identifying/removing some noisy samples)
  M' Revised learned model from S'

- **Approach**
  Identify the samples B ⊆ S which caused M to misclassify C
  Determine which samples in N ⊆ B are *likely* to be noise
  Remove N from S, producing S'

Identifying Noise in Samples

- **Identifying B requires algorithms customized for your particular model algorithm**
  C4.5, K-NN, SVMs

- **A sample in b∈B caused M to misclassify c∈C for two reasons:**

  1. b is **noisy** *or*
  2. The sample space in S is too **sparse**, so b was inappropriately made responsible too large a region.

- Based on the model M and the algorithm which produced b, we determine if it's **probably** #1 or #2

  - How many other samples are misclassifying c? [if many, it's likely #2]

  - How far is b from c? [if far, it's likely #2]

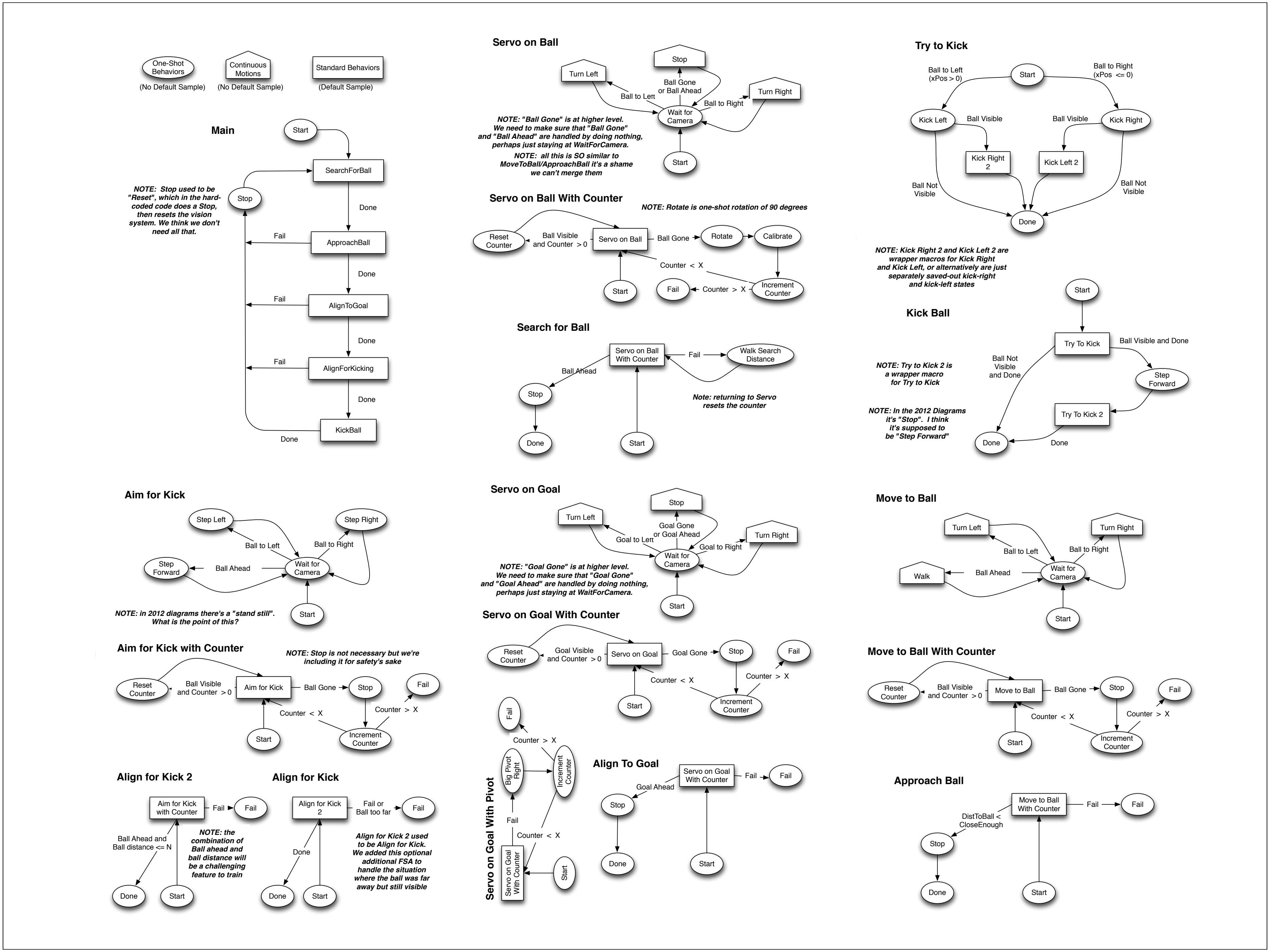
Typical Results

	Noise = 1/5				*Noise = 1/20*				*Noise = 1/100*			
Dataset	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*	*U+C*	*U+C+E*	*Metric*	*Non-Metric*
1-NN												
Iris	0.9553	0.9131	**0.9307**	**0.9255**	0.9553	0.8002	**0.8901**	**0.8601**	0.9553	0.7519	**0.9461**	**0.8490**
Glass	0.6921	0.6707	0.6810	**0.6822**	0.6921	0.6441	**0.6816**	**0.6705**	0.6921	0.5653	**0.6887**	**0.6421**
Wine	0.9533	0.9370	**0.9464**	0.9442	0.9533	0.7998	**0.9506**	**0.8722**	0.9533	0.7566	**0.9520**	**0.8488**
3-NN												
Iris	0.9537	0.9409	0.9468	0.9492	0.9537	0.8887	**0.9361**	**0.9295**	0.9537	0.8539	**0.9370**	**0.9331**
Glass	0.7008	0.6734	**0.6895**	**0.6980**	0.7008	0.6615	**0.6927**	**0.6971**	0.7008	0.6193	**0.6866**	**0.6828**
Wine	0.9615	0.9524	**0.9607**	0.9594	0.9615	0.8895	**0.9511**	**0.9472**	0.9615	0.8548	**0.9462**	**0.9408**
Decision Tree (Unpruned)												
Iris	0.9459	0.8705	**0.8915**	**0.8877**	0.9459	0.8029	**0.8497**	**0.8535**	0.9459	0.8014	**0.8765**	**0.8616**
Glass	0.6701	0.6379	**0.6577**	**0.6572**	0.6701	0.6355	**0.6544**	**0.6514**	0.6701	0.6306	**0.6591**	**0.6492**
Wine	0.9332	0.8321	**0.8638**	**0.8636**	0.9332	0.7375	**0.8103**	**0.7956**	0.9332	0.7206	**0.8365**	**0.8079**
Decision Tree (Pruned)												
Iris	0.9427	0.9135	0.9213	**0.9226**	0.9427	0.8761	**0.9081**	**0.9094**	0.9427	0.8799	**0.9250**	**0.9213**
Glass	0.6711	0.6330	**0.6520**	**0.6529**	0.6711	0.6274	**0.6460**	**0.6426**	0.6711	0.6301	**0.6501**	**0.6496**
Wine	0.9340	0.8591	**0.8811**	**0.8846**	0.9340	0.8185	**0.8749**	**0.8715**	0.9340	0.8093	**0.8892**	**0.8844**
Support Vector Machine												
Iris	0.9102	0.3886	0.4280	**0.9070**	0.9102	0.7389	**0.8649**	**0.8705**	0.9102	0.7374	**0.8695**	**0.8668**
Glass	0.3346	0.3311	0.3163	0.3393	0.3346	0.3329	0.3313	0.3284	0.3346	0.3249	0.3259	0.3350
Wine	0.9329	0.3906	0.3991	**0.9350**	0.9329	0.6400	**0.8828**	**0.8861**	0.9329	0.6544	**0.8834**	**0.8867**

RoboCup 2012

- Use HiTAB to train a humanoid robot team **at the competition**

- **Learn 17 Finite-State Automata**


One-Shot Behaviors
(No Default Sample)
Continuous Motions
(No Default Sample)
Standard Behaviors
(Default Sample)
Main
Start
SearchForBall
Stop
Done
ApproachBall
Fail
AlignToGoal
AlignForKicking
KickBall
NOTE: Stop used to be "Reset", which in the hard-coded code does a Stop, then resets the vision system. We think we don't need all that.
Servo on Ball
Turn Left
Stop
Turn Right
Ball Gone or Ball Ahead
Ball to Left
Ball to Right
Wait for Camera
Start
NOTE: "Ball Gone" is at higher level. We need to make sure that "Ball Gone" and "Ball Ahead" are handled by doing nothing, perhaps just staying at WaitForCamera.
NOTE: all this is SO similar to MoveToBall/ApproachBall it's a shame we can't merge them
Servo on Ball With Counter
NOTE: Rotate is one-shot rotation of 90 degrees
Reset Counter
Ball Visible and Counter > 0
Servo on Ball
Ball Gone
Rotate
Calibrate
Counter < X
Increment Counter
Fail
Counter > X
Start
Search for Ball
Servo on Ball With Counter
Fail
Walk Search Distance
Ball Ahead
Stop
Done
Start
Note: returning to Servo resets the counter
Try to Kick
Start
Ball to Left (xPos > 0)
Ball to Right (xPos <= 0)
Kick Left
Kick Right
Ball Visible
Kick Right 2
Kick Left 2
Ball Not Visible
Done
NOTE: Kick Right 2 and Kick Left 2 are wrapper macros for Kick Right and Kick Left, or alternatively are just separately saved-out kick-right and kick-left states
Kick Ball
Start
Try To Kick
Ball Visible and Done
Ball Not Visible and Done
Step Forward
Try To Kick 2
Done
NOTE: Try to Kick 2 is a wrapper macro for Try to Kick
NOTE: In the 2012 Diagrams it's "Stop". I think it's supposed to be "Step Forward"
Aim for Kick
Step Left
Step Right
Ball to Left
Ball to Right
Step Forward
Ball Ahead
Wait for Camera
Start
NOTE: in 2012 diagrams there's a "stand still". What is the point of this?
Aim for Kick with Counter
NOTE: Stop is not necessary but we're including it for safety's sake
Reset Counter
Ball Visible and Counter > 0
Aim for Kick
Ball Gone
Stop
Fail
Counter > X
Counter < X
Increment Counter
Start
Align for Kick 2
Aim for Kick with Counter
Fail
Ball Ahead and Ball distance <= N
Done
Start
NOTE: the combination of Ball ahead and ball distance will be a challenging feature to train
Align for Kick
Align for Kick 2
Fail or Ball too far
Fail
Done
Start
Align for Kick 2 used to be Align for Kick. We added this optional additional FSA to handle the situation where the ball was far away but still visible
Servo on Goal
Turn Left
Stop
Turn Right
Goal Gone or Goal Ahead
Goal to Left
Goal to Right
Wait for Camera
Start
NOTE: "Goal Gone" is at higher level. We need to make sure that "Goal Gone" and "Goal Ahead" are handled by doing nothing, perhaps just staying at WaitForCamera.
Servo on Goal With Counter
Reset Counter
Goal Visible and Counter > 0
Servo on Goal
Goal Gone
Stop
Fail
Counter < X
Counter > X
Increment Counter
Start
Servo on Goal With Pivot
Fail
Counter > X
Big Pivot Right
Increment Counter
Fail
Counter < X
Servo on Goal With Counter
Start
Align To Goal
Servo on Goal With Counter
Fail
Goal Ahead
Stop
Done
Start
Move to Ball
Turn Left
Turn Right
Ball to Left
Ball to Right
Walk
Ball Ahead
Wait for Camera
Start
Move to Ball With Counter
Reset Counter
Ball Visible and Counter > 0
Move to Ball
Ball Gone
Stop
Fail
Counter < X
Counter > X
Increment Counter
Start
Approach Ball
Move to Ball With Counter
Fail
DistToBall < CloseEnough
Stop
Done
Start

Simple “Flat” Swarms with HiTAB

- **Homogenous Case:** Every agent uses the same behavior.
  *This is **not just parallel:** the agents interact.*

- **Heterogeneous Case:** Agents belong to disjoint classes. Only agents in the same class use the same behavior.

- If the **interesting** behaviors require interaction, how do you train agents simultaneously?

- Example: to passing behaviors, you must teach **two** robots at the same time how to coordinate passing and receiving.

Behavioral Bootstrapping

- If you have **multiple agents that must be trained <u>simultaneously</u>**
- ... and you only have **one trainer ... ?**

- **Homogeneous Case**

1. Set all agents to empty behaviors (doing nothing)
2. Select an Agent and train a **slightly better behavior** in the context of the agents' existing behaviors
3. Distribute this behavior to all the agents
4. Go to 1

Behavioral Bootstrapping

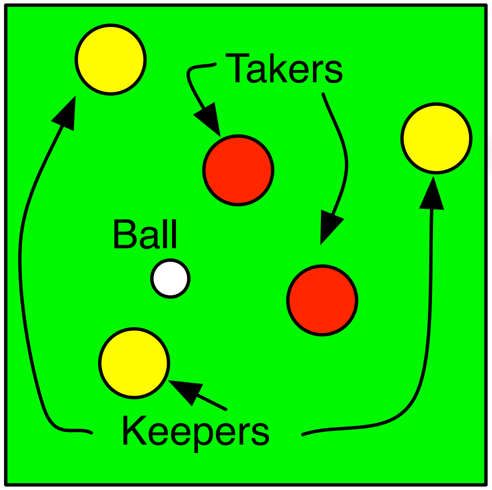
- **Heterogeneous Case** *(2-agent example)*

1. Set both agents to empty behaviors (doing nothing)
2. Select Agent A and train a **slightly better behavior** in the context of Agent B's existing behavior
3. Select Agent B and train a **slightly better behavior** in the context of Agent A's existing behavior
4. Go to 1

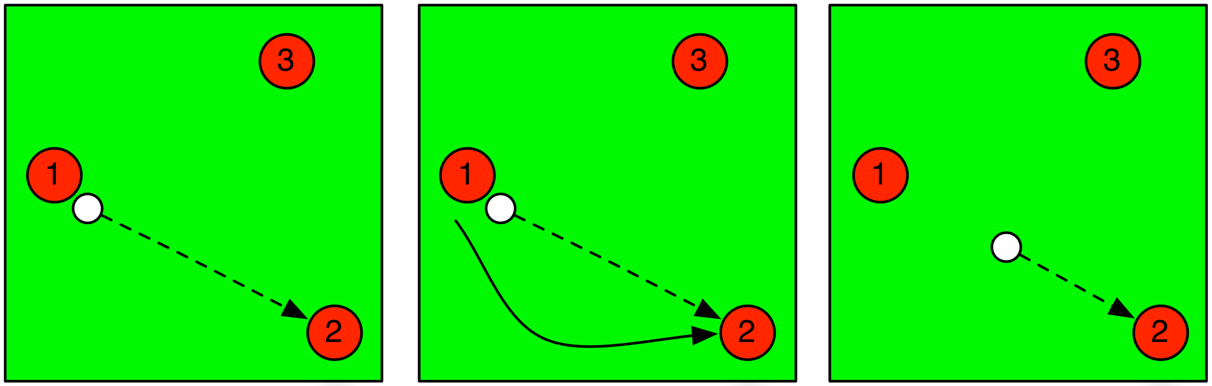
Behavioral Bootstrapping: **Keepaway Soccer**

- **Three Keepers, Two Takers**
  The Keepers have control of the ball
  The Takers are trying to take the ball

  The Takers are hard-coded
  We are training the Keepers (Homogeneous)



- **Passing Requires coordination between a passer and a receiver**



Player 1 decides to pass to Player 2 | As Player 1 passes, it also yells to Player 2 | Player 2 stops trying to Get Open and prepares to Receive

Behavioral Bootstrapping: **Keepaway Soccer**

Behavioral Bootstrapping: **Keepaway Soccer**

- **Results**
- **University of Texas, Austin Hard-Coded Team**
  5.6 Seconds On Average (before takers take the ball)
- **George Mason University Bootstrapped Team**
  7 Seconds on Average
  9 Seconds on Average if using “yelling”

Multiagent Training

- **Techniques for Multiagent Training are nearly always optimizers.**
  - Multiagent Reinforcement Learning, Stochastic Optimization
  - Supervised Learning is **extremely rare** for multiagent training. Yet training is a supervised task!

- **User Modeling** The team learns about one another
- **Training** (or **Demonstration)** The team learns to do a task set by you

The MAS Inverse Problem

- **Emergence** Given the micro-behaviors, we can't guess the emergent macro-phenomenon without simulation.
- **The MAS Inverse Problem** Given a desired emergent macro-phenomenon, we can't guess the micro-behaviors **at all.**
- **How this Affects Training:**
  - The trainer can tell the agents "in situation X, the macro-phenomenon should be Y" (when it's dark, storm the castle)
  - To learn, an agent needs to know "in situation X, my micro-behavior should be Z" (when it's dark, stay to the left of Bob)
  - **We can't easily compute the micro-behaviors to achieve the desired macro-phenomena**

Optimization Solves Inverse Problems

- **Training With an Optimizer:**
  - Create a new candidate solution consisting of micro-behaviors.
  - Test in the simulator to observe the resulting macro-phenomenon.
  - Assess the error in the macro-phenomenon.
  - Repeat.

Optimization Solves Inverse Problems

- **Supervised Learning Doesn't Work**
  Multiagent Systems Inverse Problem. The separation between the micro-behaviors and macro-level phenomenon is too large
- **Stochastic Optimization**
  - Simulated Annealing, Hill-Climbing, *etc.:* test one solution at a time
  - Evolutionary Computation: test many solutions at a time
    (very good for multiagent systems
- **Reinforcement Learning**
  - Q-Learning, Policy Search
- **BUT:** optimization requires many trials to gather samples. In robotics, a trial is *very expensive.*

Multi-Agent HiTAB: Training Hierarchies of Swarms

- **Goal**
  Train **complex**, stateful behaviors from a very small number of samples in real time in **arbitrarily large swarms of agents.**

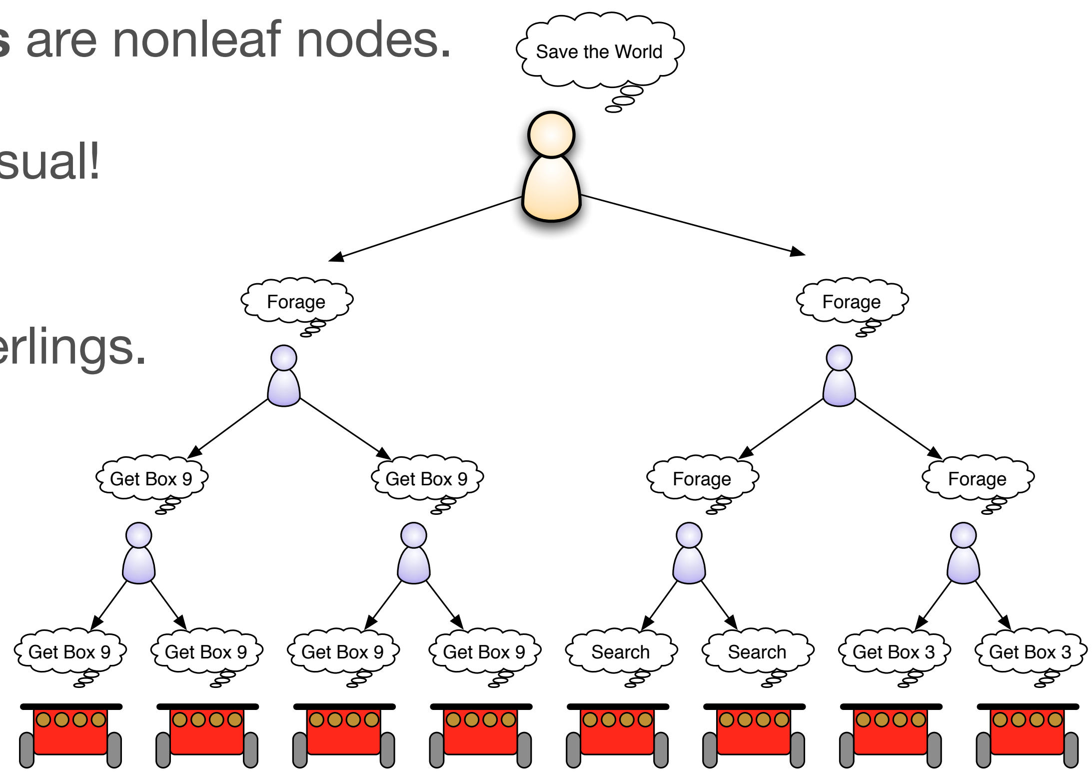
- **Difficulties**
  1. Curse of dimensionality. [like single-agent]
  2. The **Multiagent Inverse Problem**.

- **Solution: Swarm Decomposition**
  Manually break the joint multiagent behaviors into simpler behaviors for smaller sub-swarms.

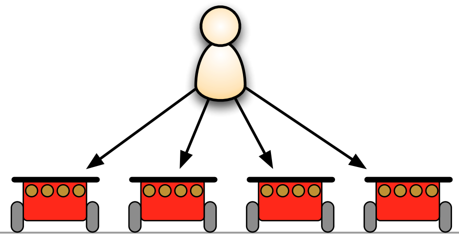
  Train the simpler behaviors on small swarms, then train composed behaviors on larger swarms.

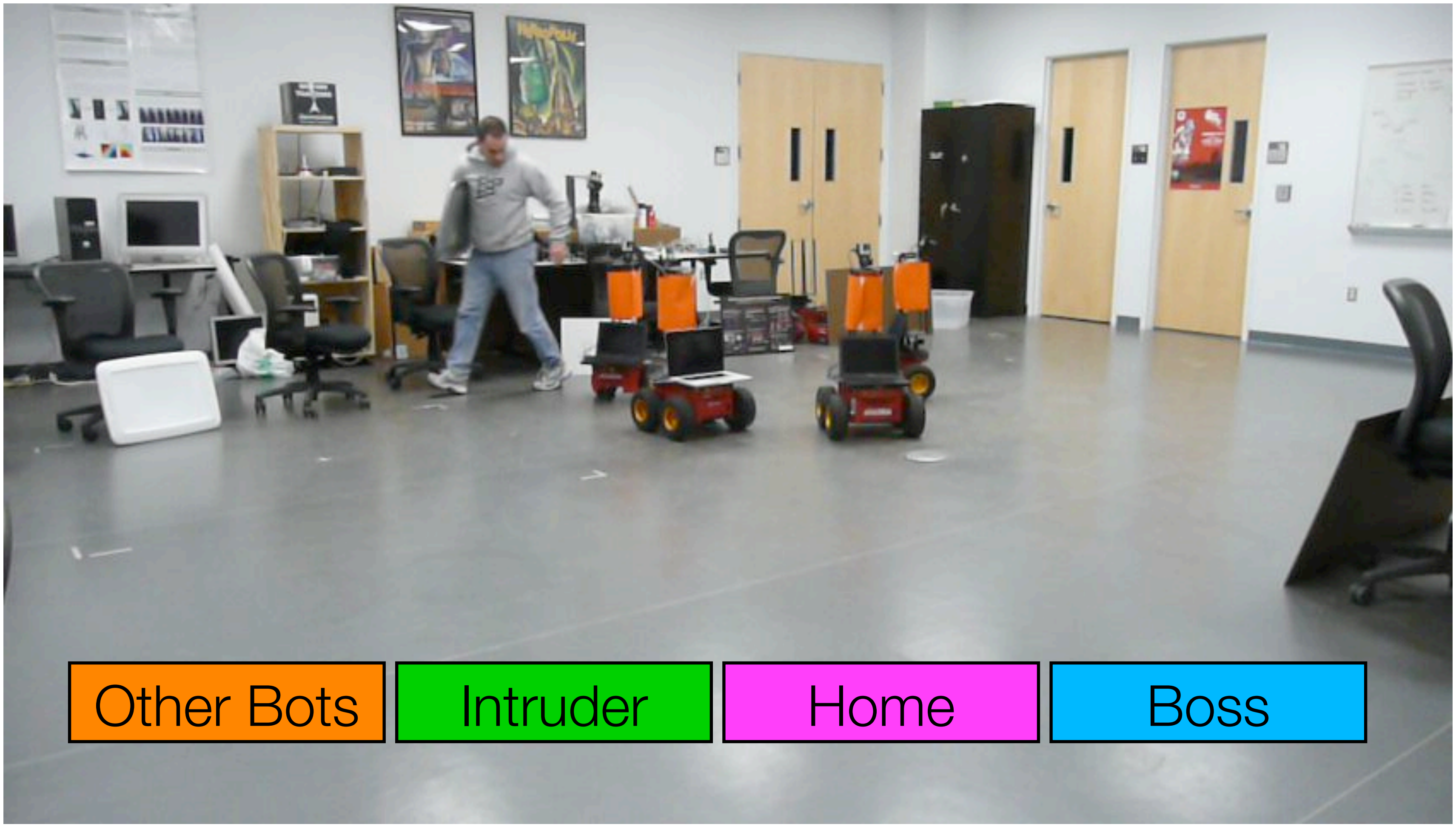
HiTAB Multi-Agent Model

- Decompose the swarm into a **hierarchy of subswarms.**
- **“Regular” (real) agents** are leaf nodes.
- **Controller (“boss”) agents** are nonleaf nodes.
- Train controller agents as usual!
- **Basic Behaviors**
  Top-level behaviors of underlings.
- **Features**
  Statistics about underlings.

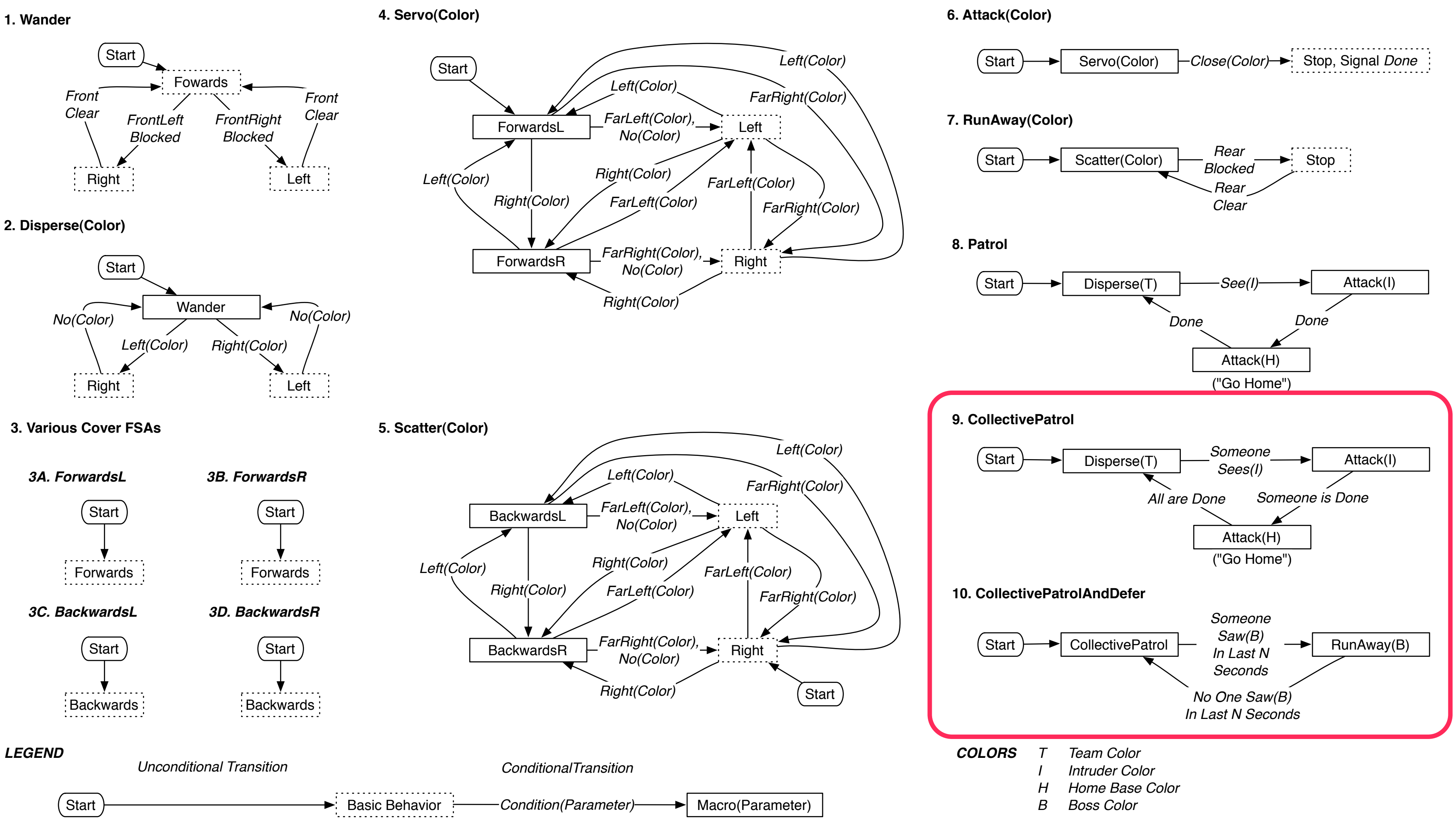


Simple Multiagent Example





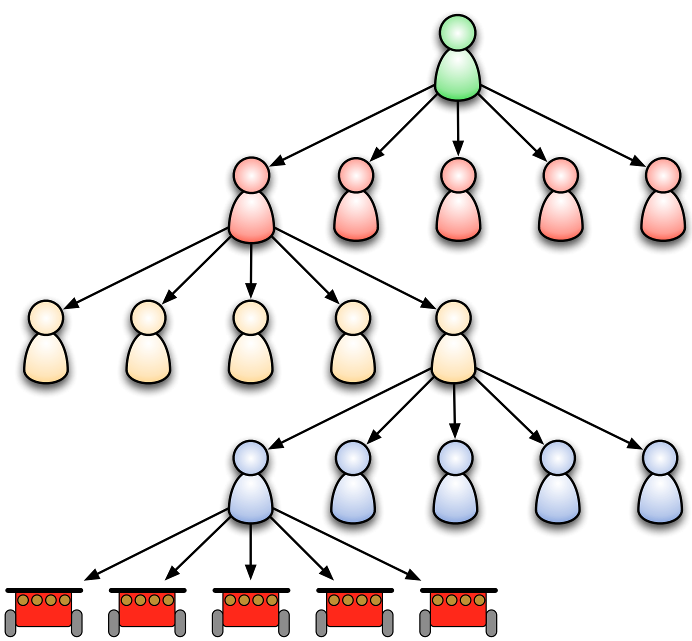
Simple Multi-Agent Example

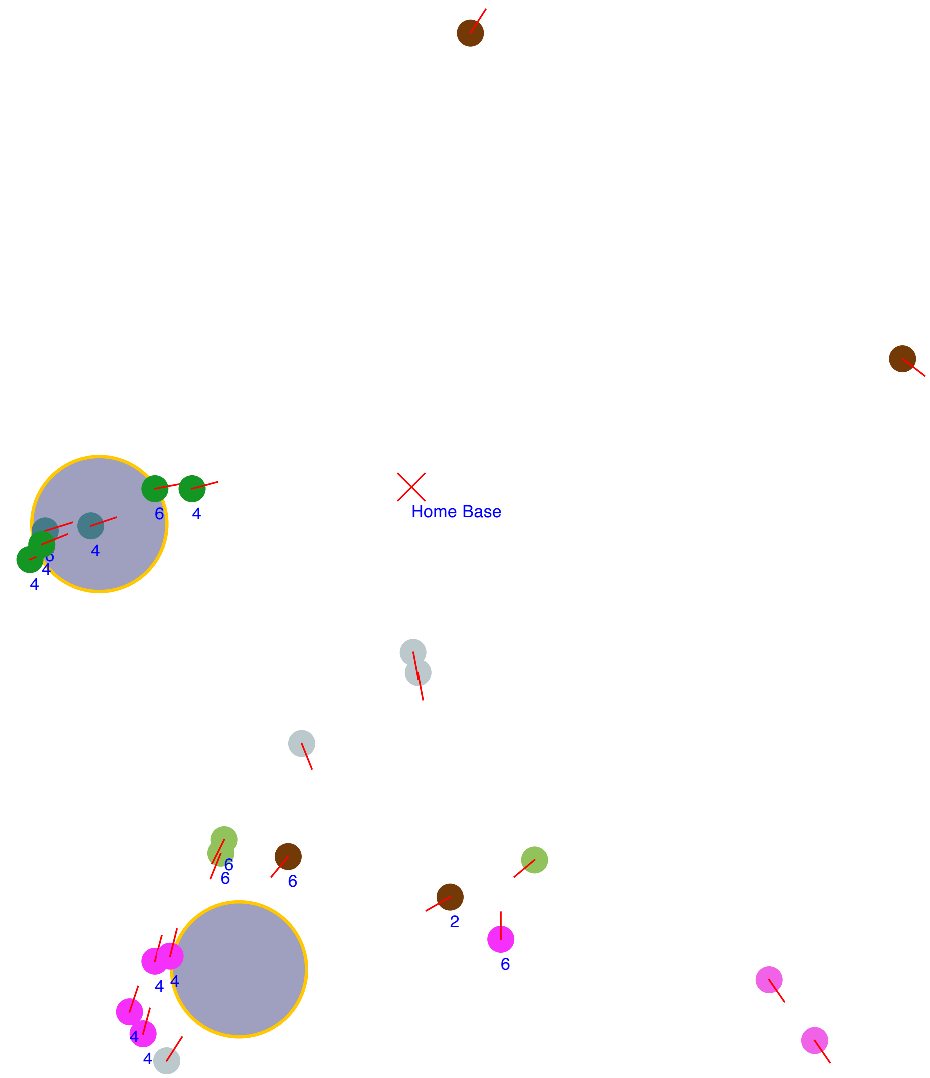


Larger Multi-Agent Model

- **Box Collecting**
  Boxes require 5, 25, or 125 agents to retrieve

- **We've trained up to 625 agents**





Collaborators

HiTab
Daniele Nardi
Vittorio Ziparo
University of Rome, La Sapienza

Students

Ant Pheromones
Brian Hrolenok
Liviu Panait
Gabriel Balan
Katherine Russell

Single-Agent HiTab
Katherine Russell
Khaled Talukder
Ahmed ElMolla
Kevin Andrea

Multi-Agent HiTaB, Unlearning, Behavioral Bootstrapping
Keith Sullivan
Bill Squires

NSF
ONR
Office of Naval Research