# Humanoid Robots Learning to Walk Faster: From the Real World to Simulation and Back

ALON FARCHY, SAMUEL BARRETT,
PATRICK MACALPINE, PETER STONE

The University of Texas at Austin

AAMAS Conference
May 8, 2013

# Motivation

Low-level robot skills are important

- Robust walking and turning
- Precise robotic arm movement
- Localization
- Stability

# Motivation

Low-level robot skills are important

- Robust walking and turning
- Precise robotic arm movement
- Localization
- Stability

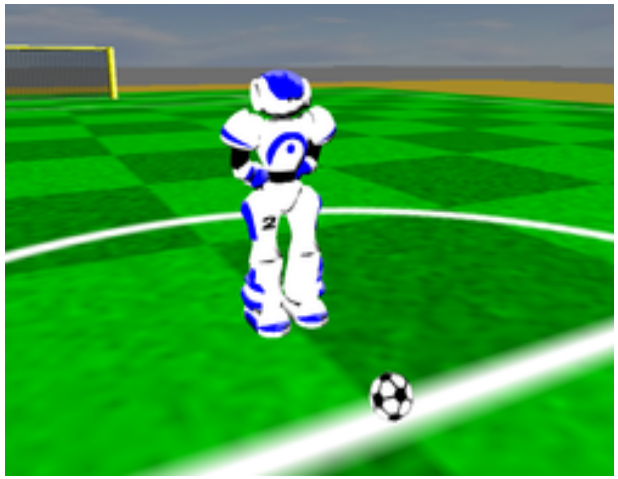
But learning on a robot is **challenging**:

- Many environmental factors
- Robot performance degrades with use
- Robots take time to operate
- Lack of ground truth
- Rarely successful

# Motivation

Learning in **simulation**:

- Can run many tests in **parallel**
- No human supervision
- No damage to robots



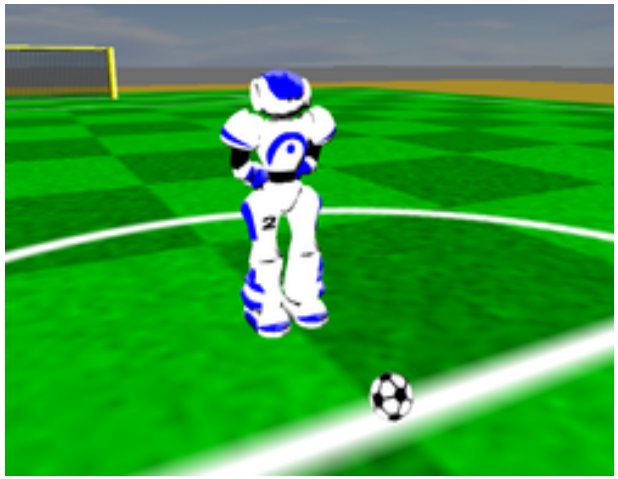
# Motivation

Learning in **simulation**:

- Can run many tests in **parallel**
- No human supervision
- No damage to robots

**But**:

- Behaviors learned in simulation **may not apply to robots**
- Imperfect physics



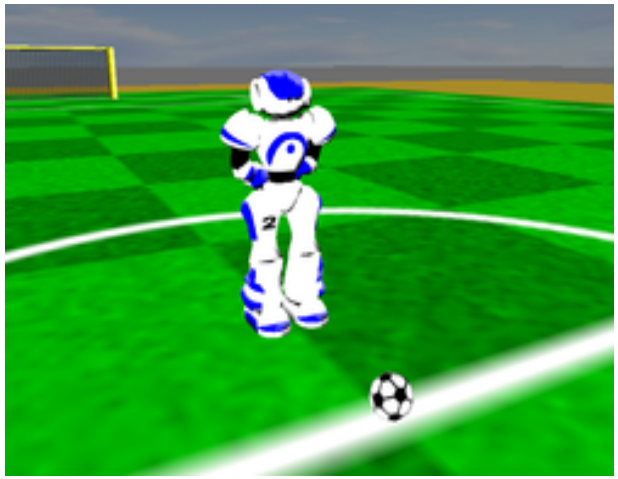
# Motivation

Learning in **simulation**:

- Can run many tests in **parallel**
- No human supervision
- No damage to robots

**But**:

- Behaviors learned in simulation **may not apply to robots**
- Imperfect physics

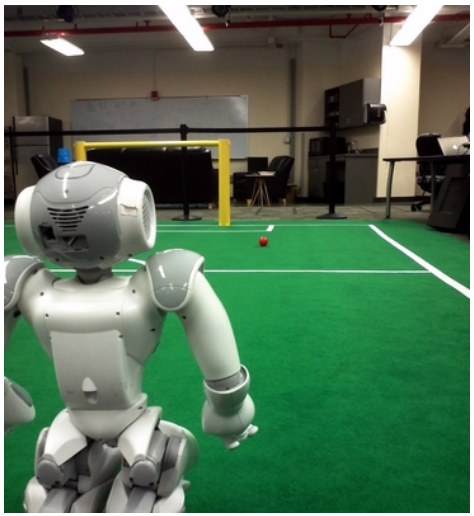


**Research Question**:

How can we **modify** a simulator to learn effective behaviors on **real robots**?

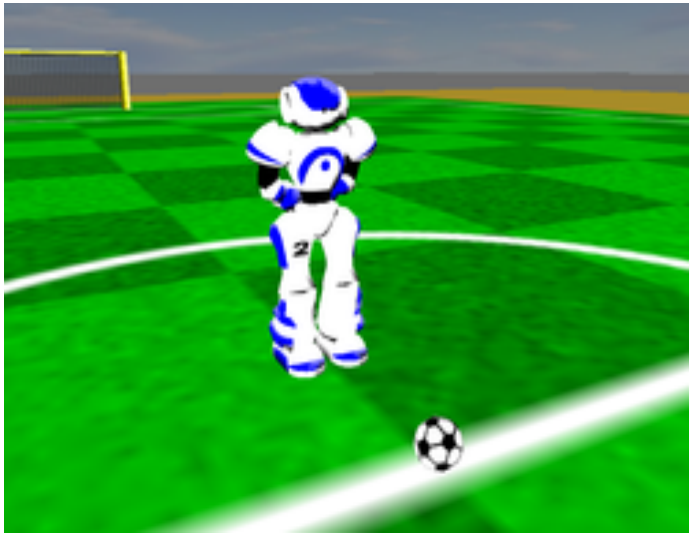
# A Little Background… **RoboCup**

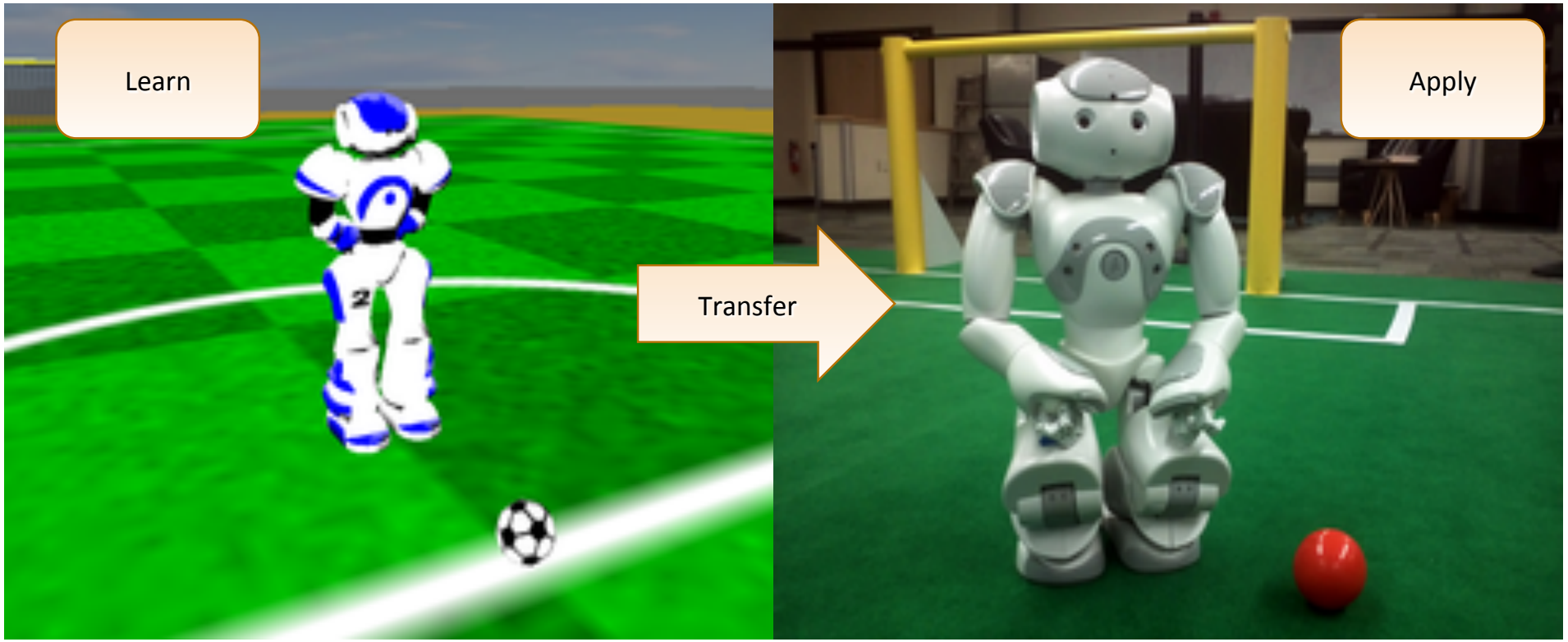
- The **RoboCup Standard Platform League**: Soccer for robots
- Requires **fast, stable** walk
- Robots **wear out**
- Running tests is **time consuming**
- Therefore, using machine learning is **hard**



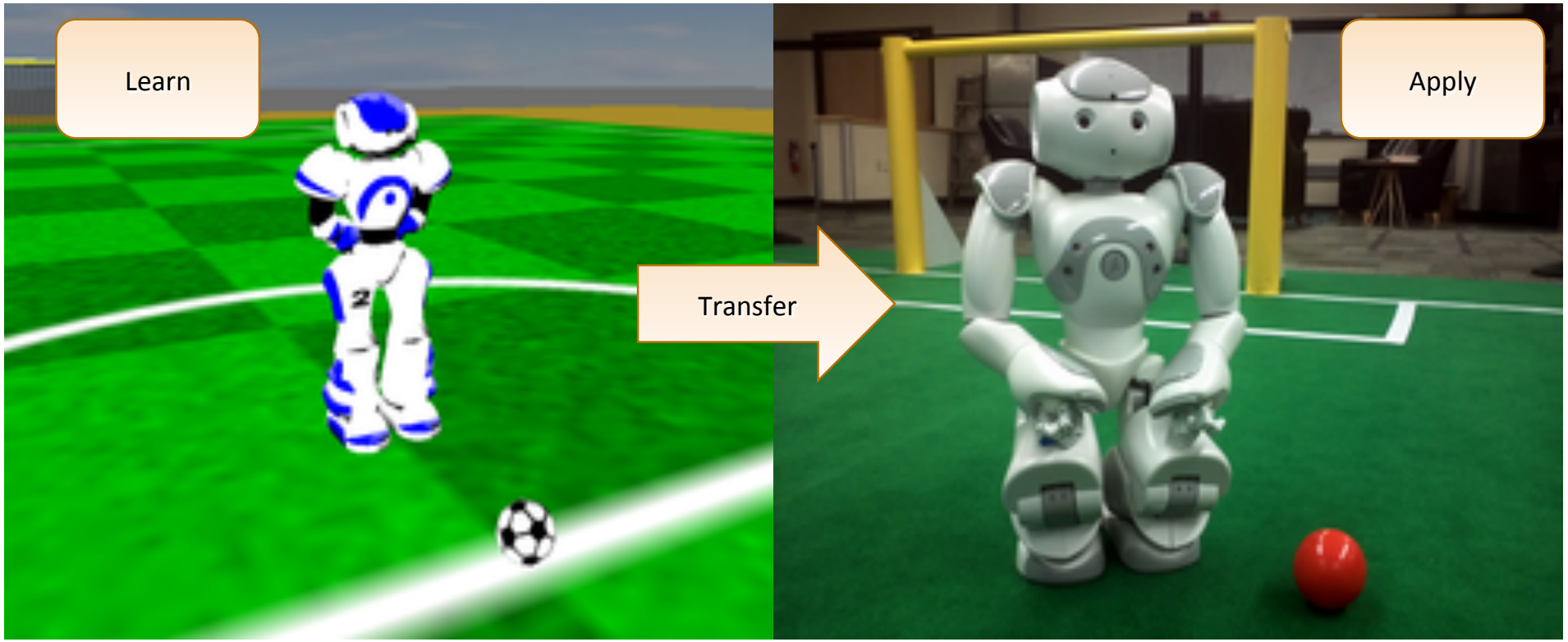
# A Little Background… RoboCup **Simulation**

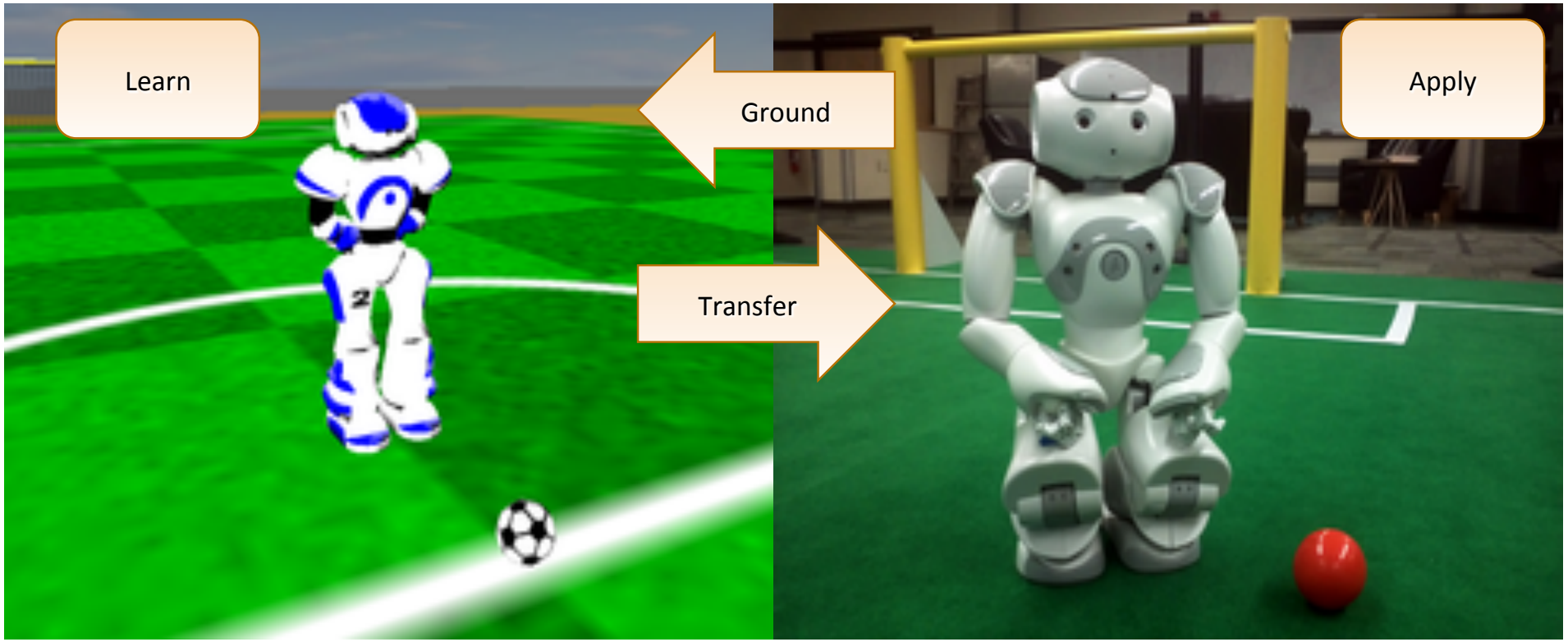
- The **3D Simulation League**: Soccer for virtual robots
- Requires **fast, stable,** intelligent robots
- Robots are **unlimited**
- **Great environment for machine learning**
- In 2011-2012, UT Austin Villa Simulation League outpaced the competition by using **machine learning**.




Learn
Transfer
Apply

# Walking in Simulation


Learn
Transfer
Apply


Learn
Ground
Transfer
Apply

# Challenges

Many **differences** between simulation and the real world:

- Open Dynamics Engine (ODE) is far from perfect
  - Nearly **infinite torque**
  - No heat simulation
- Virtual Nao is greatly approximated
  - Equal joint strength, perfect balance, simple foot shape
- Soccer environment is greatly approximated
  - Perfectly **flat** surface

# Outline

- Introduction
- Grounded Simulation Learning (GSL)
- Implementation
- Results
- Conclusion

# Outline

- Introduction
- **Grounded Simulation Learning (GSL)**
- Implementation
- Results
- Conclusion

# Grounded Simulation Learning (GSL)

**Concept**:

- Optimize parameters in simulation
- Ground simulator to match real robots
- Repeat

# Grounded Simulation Learning (GSL)

**Assumptions:**

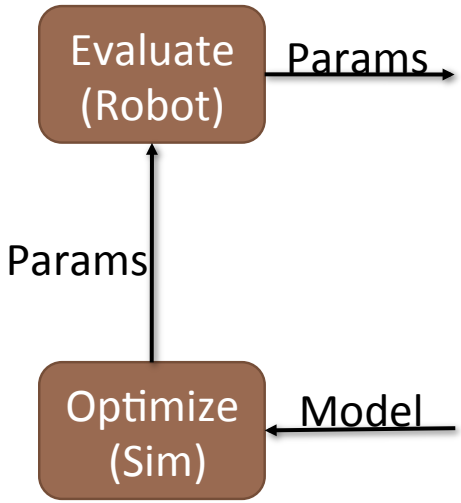
1. Can modify simulator evaluation
2. Can run **a few** robot tests
3. Can run **many** simulation tests

# Grounded Simulation Learning (GSL)

**Assumptions:**

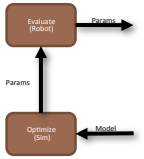
1. Can modify simulator evaluation
2. Can run **a few** robot tests
3. Can run **many** simulation tests
4. Can learn model of joint actions
5. Can bias optimization's exploration of parameters

# GSL: Optimize

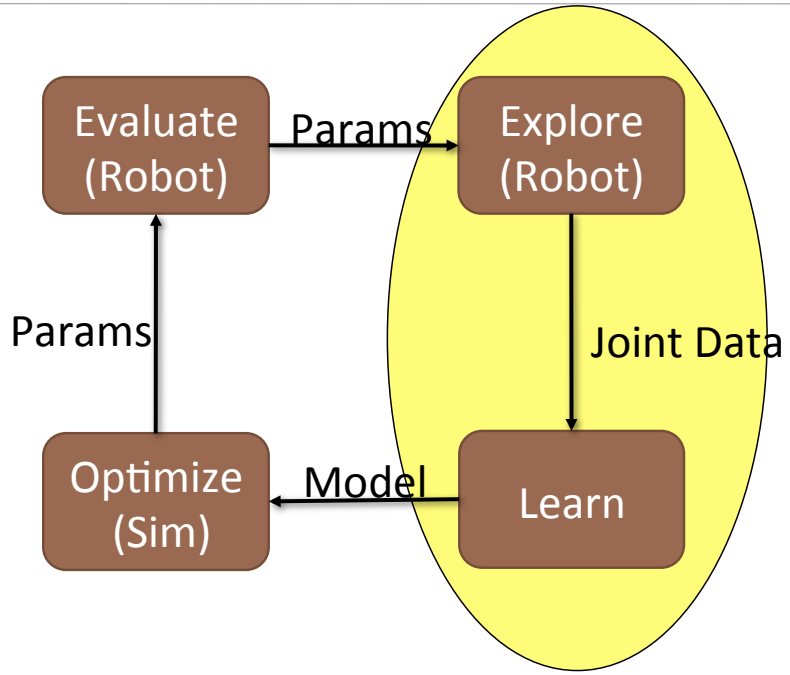


# GSL: Optimize

- Optimize parameters in **simulation**
  - Can run many evaluations in parallel
  - Constrained by model of real world
- Only perform **local** search
  - Searching far from the base parameters very likely to exploit idiosyncrasies in the simulation



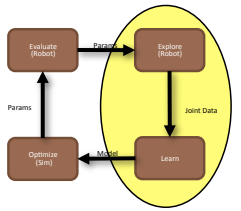
# GSL: Ground


Evaluate
(Robot)
Params
Explore
(Robot)
Params
Joint Data
Optimize
(Sim)
Model
Learn

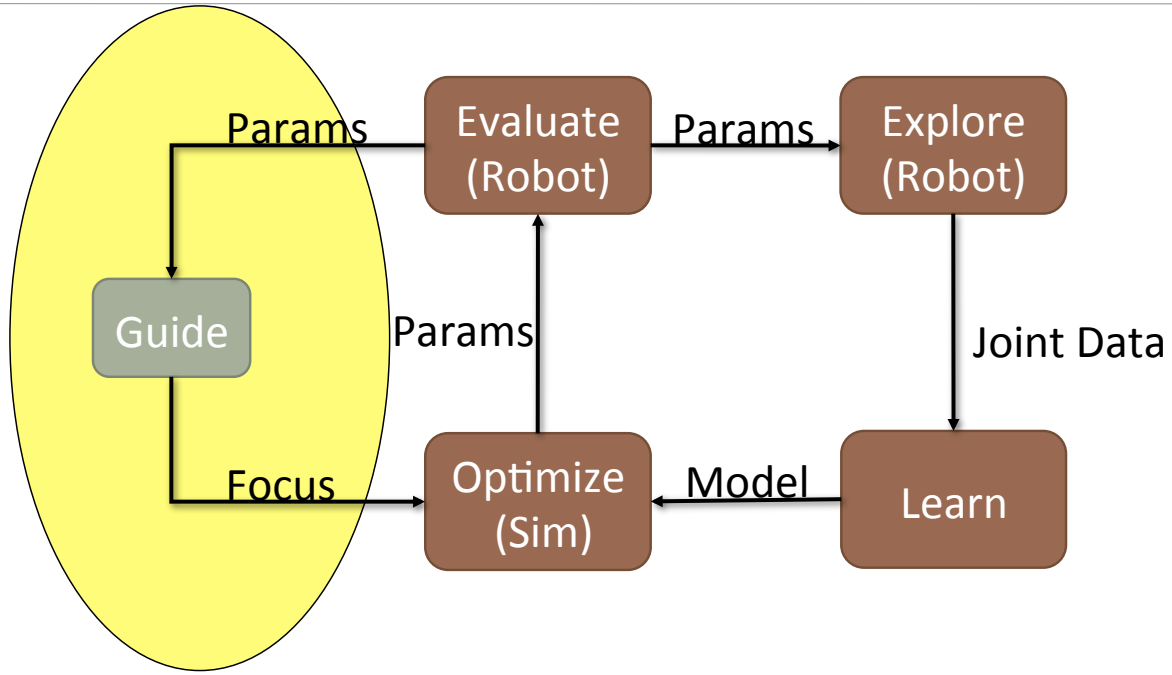
# GSL: Ground

**Force simulation to act like the robot**

1. **Collect** joint data and actions
2. **Learn** model of actions
3. **Modify** simulator using model

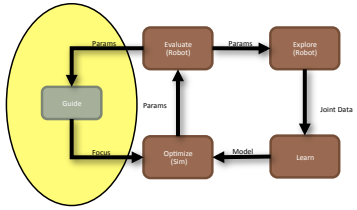


# GSL: Guide


Params
Evaluate (Robot)
Params
Explore (Robot)
Guide
Params
Joint Data
Focus
Optimize (Sim)
Model
Learn

# GSL: Guide

1. Try learned parameters on robot
2. **Select** which parameters to optimize
   - In our case, this selection was performed manually.
   - Control initial variance of parameters
3. Repeat



# Implementation

Fitness Evaluation

Predicting Joint Angles

Optimizing (CMA-ES)

Guidance

Results

# Fitness Evaluation

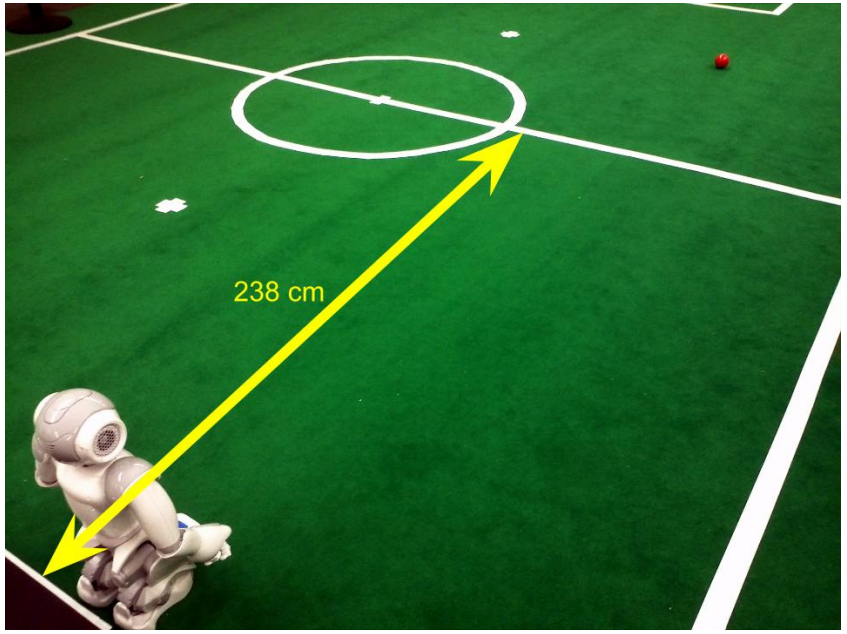
**Forwards walk speed**

Real world:

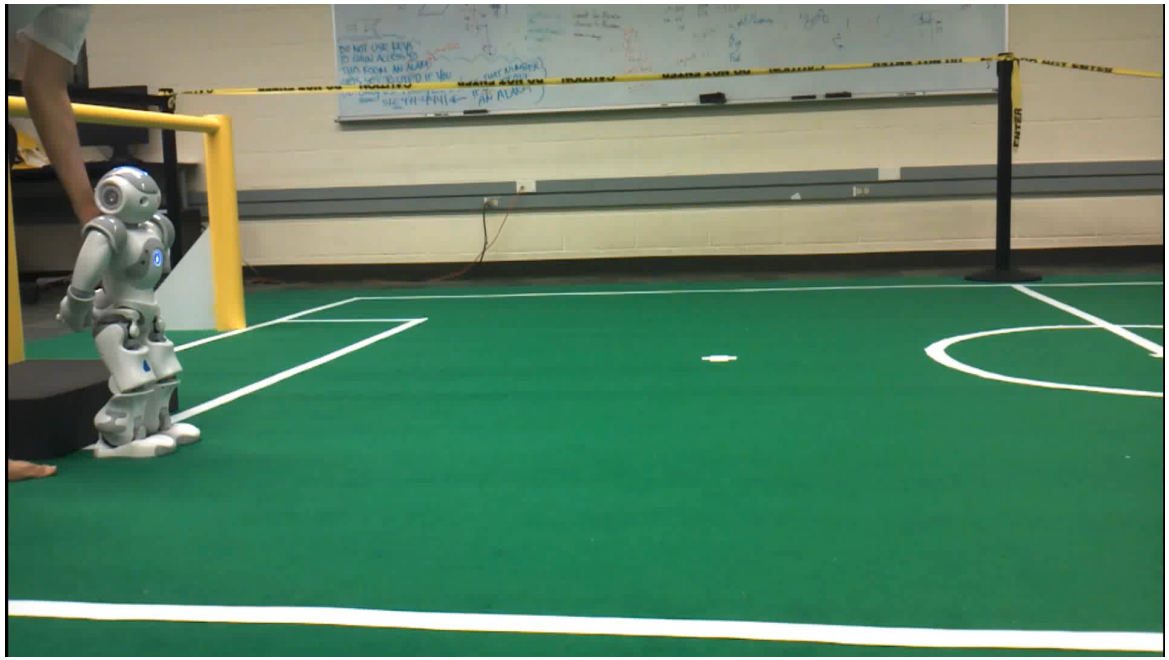
- Walk 238cm forward (towards ball)
- Stop when foot reaches white line
- Goal is to minimize travel time

Simulation:

- Walk forward for 15 seconds
- Maximize forward movement



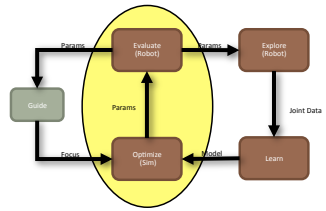
# Fitness Evaluation



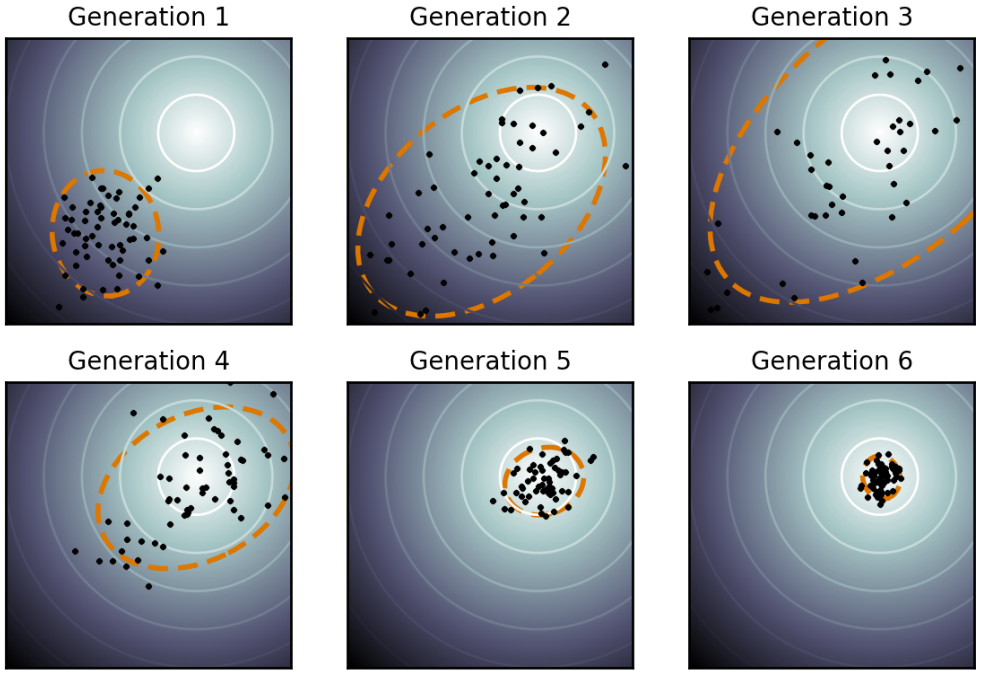
# Optimizing in Simulation: **CMA-ES**

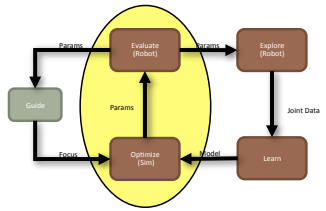
**Covariance Matrix Adaptation Evolution Strategy**

- Candidates sampled from multi-dimensional Gaussian distribution
- Weighted average of members with highest fitness used to update mean of distribution
- Covariance updated using search history



# Optimizing in Simulation: **CMA-ES**




Source: wikipedia.org

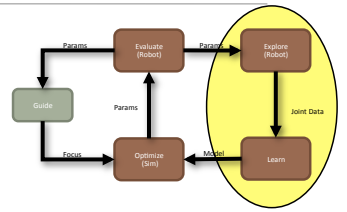
# Grounding: Predicting Joint Angles

- Walk forward and turn
- Record joint angles and commands
- **M5P** regression tree
- Learn mapping from:

  Angles + Actions

  to

  Next Angles



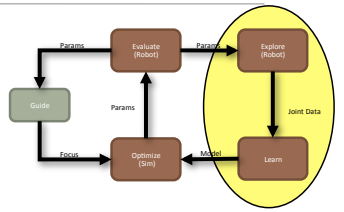

| Class | Time(s) | RAE(%) | RRSE(%) |
|---|---|---|---|
| SimpleLinearRegression | 0.35 | 21.7 | 25.5 |
| LinearRegression | 1.18 | 16.0 | 17.6 |
| LeastMedSq | 169.0 | 16.9 | 19.2 |
| PaceRegression | 2.21 | 15.9 | 17.5 |
| MultilayerPerceptron | 376 | 12.9 | 14.2 |
| DecisionStump | 1.82 | 56.8 | 57.5 |
| M5P | 37.3 | 11.6 | 16.3 |
| REPTree | 3.68 | 12.4 | 14.5 |
| IBk | 128.9 | 12.3 | 16.6 |
| RegressionByDiscretization | 20.2 | 16.7 | 18.1 |

RAE – Relative Absolute Error

RRSE – Relative Root Squared Error

# Grounding: Predicting Joint Angles

How to apply model to simulation?




**Linear combination** of requested joint angles and predicted joint angles

From initial tests: 70% requested + 30% predicted.

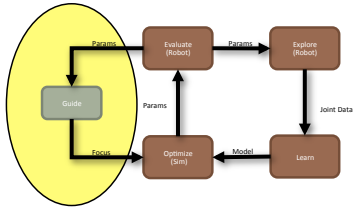
Now we can use this grounded simulation in **Optimize**.

# Guidance

Evaluate optimized parameters on robot

Select parameters for **OpenParams** (easy)

- Robot Falls? ✖
- Robot Faster? ✔



# Guidance

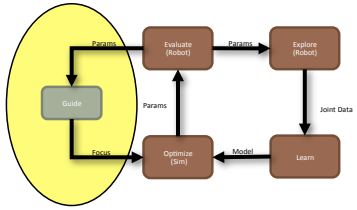
Evaluate optimized parameters on robot

Select parameters for **OpenParams** (easy)

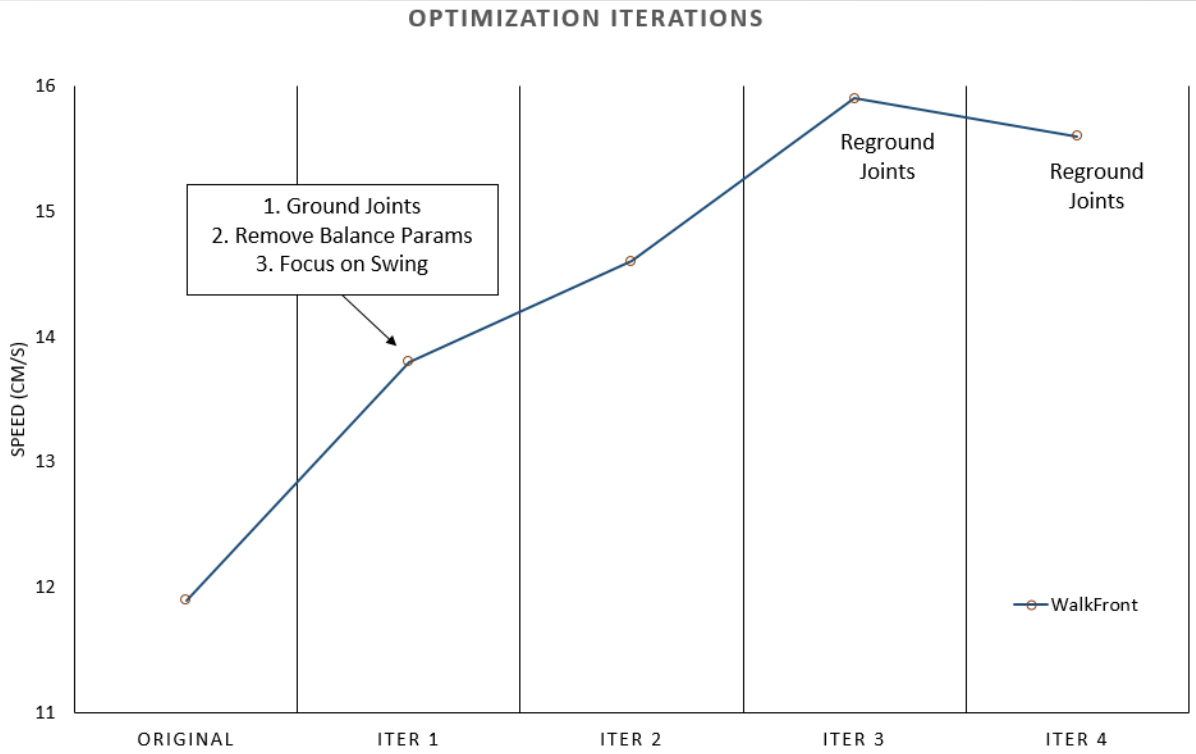
- Robot Falls? ✖
- Robot Faster? ✓

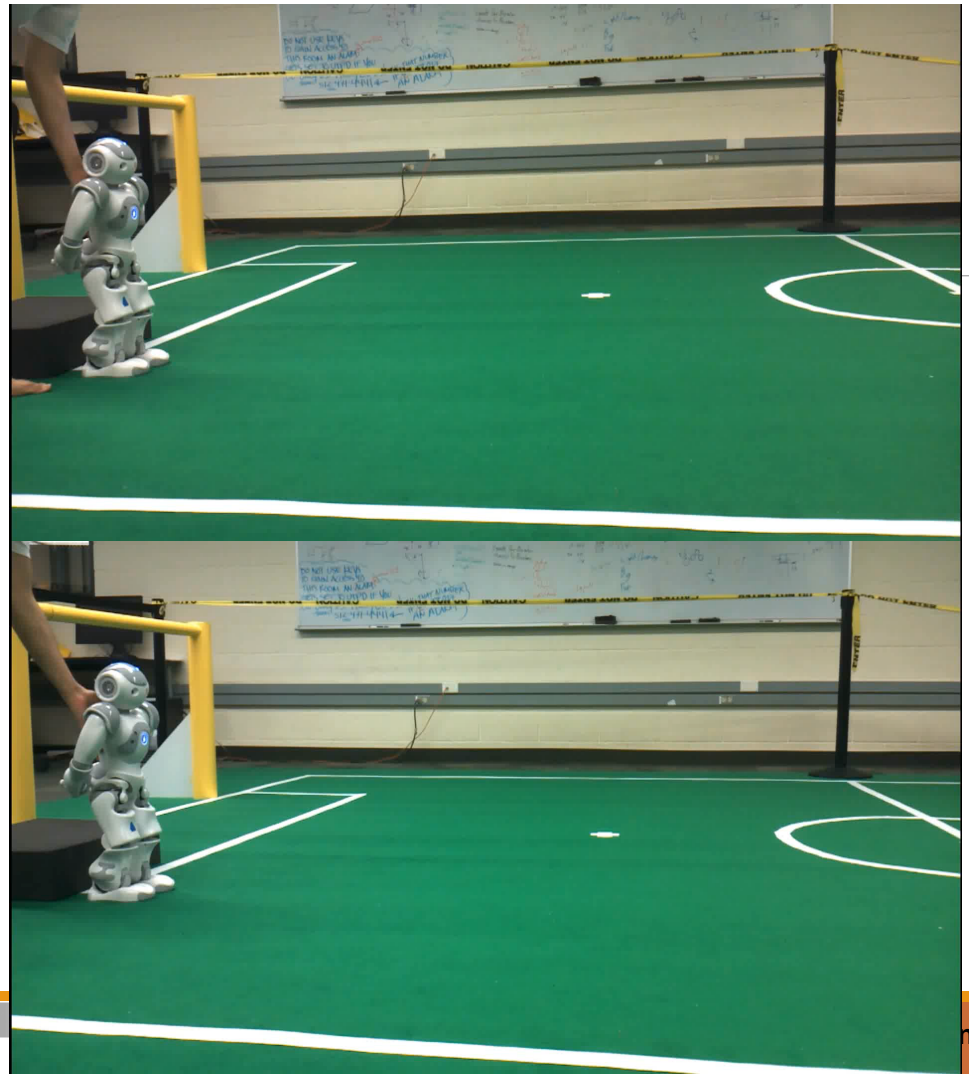
Bias **Optimize** to better parameters (harder)

- Manually tweaked variance of parameters in the CMA-ES.
- Could be automated.



# Results


OPTIMIZATION ITERATIONS
SPEED (CM/S)
16
15
14
13
12
11
1. Ground Joints
2. Remove Balance Params
3. Focus on Swing
Reground Joints
Reground Joints
WalkFront
ORIGINAL
ITER 1
ITER 2
ITER 3
ITER 4



Original: 13.5 cm/s

Optimized: **17.1 cm/s**

**26.7% Improvement!**

# Related Work

P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In International Conference on Machine Learning (ICML) Pittsburgh, pages 1-8. ACM Press, 2006.

J. C. Zagal, J. Delpiano, and J. Ruiz-del Solar. Self-modeling in humanoid soccer robots. Robot. Auton. Syst., 57(8): 819{827, July 2009.

L. Iocchi, F. D. Libera, and E. Menegatti. Learning humanoid soccer actions interleaving simulated and real data. In Second Workshop on Humanoid Soccer Robots, November 2007.

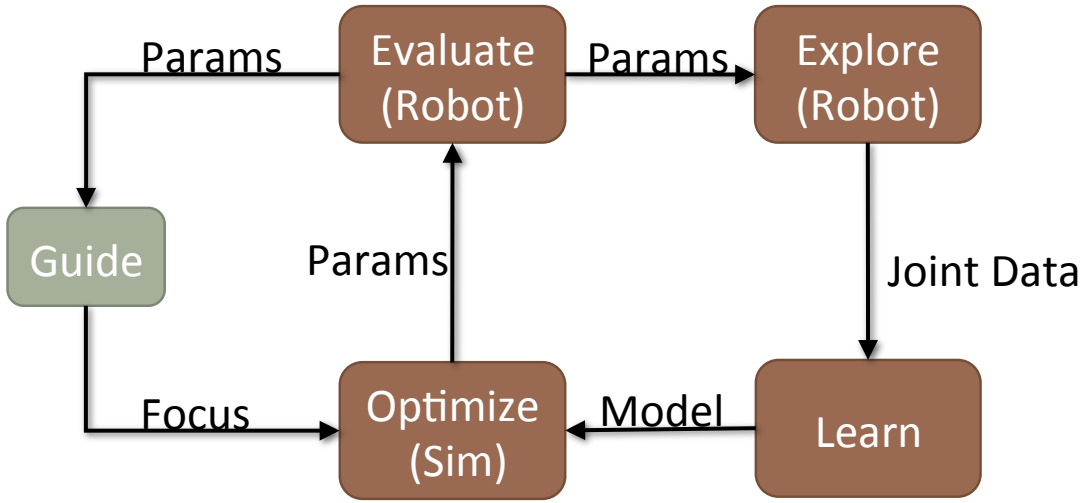
S. Koos, J. B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In Proceedings of the 12th annual conference on Genetic and Evolutionary computation, GECCO '10, pages 119{126, New York, NY, USA, 2010. ACM.

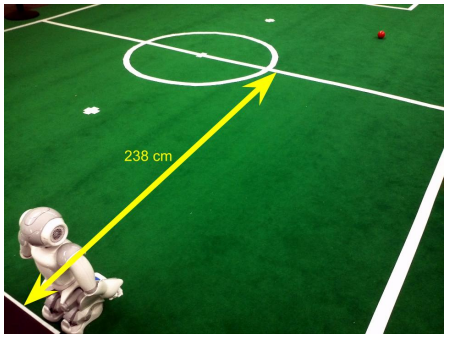
# Conclusions

**Can learn good low-level skills using simulation**

- Optimize in simulation
- Evaluate on real robot
- Iteratively refine simulator to match real world
- Guide optimization

# Questions?




Poster: Tomorrow 10-11am P3-36

# GSL: Parameters

Input:

- **$P_0$** – Initial parameter set
- **$Fitness_{sim}$** – A simulation fitness function that uses a model that maps joint commands to outputs
- **$Fitness_{robot}$** – a robot fitness function
- **$Explore_{robot}$** – a robot exploration routine
- **Learn** – A supervised learning algorithm
- **Optimize** – An optimization algorithm to run in simulation

Output:

- **$P_{opt}$** – Optimized parameter set

Variables:

- **BestFitness** – Current best fitness evaluation on the robot
- **OpenParams** – Bag of pairs (Parameter set, Fitness) to try

# GSL: Putting it together

- **$P_0$** – Initial parameter set
- **$\text{Fitness}_{sim}$** – Simulation fitness function
- **$\text{Fitness}_{robot}$** – Robot fitness function
- **$\text{Explore}_{robot}$** – Robot exploration routine
- **Learn** – Supervised learning algorithm
- **Optimize** – Simulation optimization algorithm
- **BestFitness** – Current best fitness
- **OpenParams** – Bag (Parameter set, Fitness)

```
   // Initialize
 1 BestFitness ← Fitness_robot(P_0)
 2 OpenParams ← stack((P_0,BestFitness))
 3 while OpenParams is not empty do
 4 |   p, f ← pop(OpenParams)
   |   // Ground
 5 |   Collect actions,state from running Explore_robot(p)
 6 |   Model ← Learn(actions,state)
   |   // Optimize
 7 |   ParamSets ← Optimize(Fitness_sim(p,Model))
   |   // Guide
   |   // Update OpenParams and Best
 8 |   foreach p' in ParamSets do
 9 |   |   f' ← Fitness_robot(p')
10 |   |   if f' > f then
11 |   |   |   push(OpenParams,(p', f'))
12 |   |   if f' > BestFitness then
13 |   |   |   P_opt ← p'
14 |   |   |   BestFitness ← f'
15 |   Choose parameters to focus on for subsequent
   |   optimization (line 7).
```