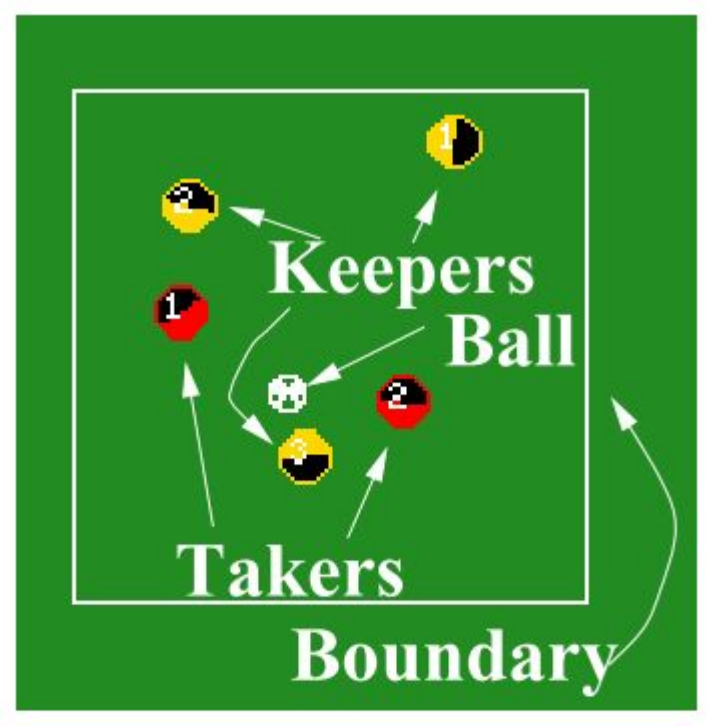
Learning Complex Behaviours and Keepaway in Robocup 3D

~ Nilesh Gupta

KeepAway

“**Keepers**”, tries to keep control of the ball for as long as possible despite the efforts of “**Takers**”.



Challenges in Keepaway

- Large and **continuous state space**, “curse of dimensionality”
- **Hidden state**, agent has only a partial world view
- **Noisy sensors and actuators**, do not perceive the world exactly as it is, nor can they affect the world exactly as intended
- **Asynchronous**, perception and action cycle different
- **Distributed** and **multi-agent** domain with teammates and adversaries

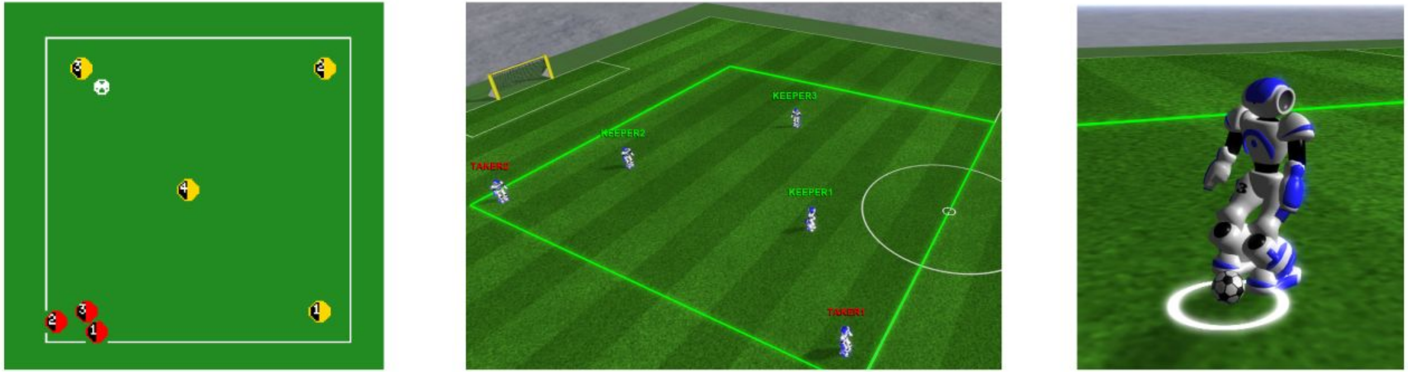
Keepaway in 2D

- Paper by P. Stone and R. Sutton describes the learning of higher-level decisions in 2D keepaway
- Modelled keepaway as an **SMDP**, keepers learn independently, takers strategy fixed
- Used **Sarsa(λ)** with **linear tile-coding** function approximation and variable λ

3D Robocup Domain Description

- Each robot has **22 degrees of freedom** : six in each leg, four in each arm, and two in the neck
- Agent is equipped with **joint perceptors and effectors**
- **Noisy visual information** about the environment is given to an agent every third simulation cycle (60 ms)
- Agents can communicate with each other every other simulation cycle (40 ms)

Additional Challenges in 3D Robocup



(a) 2D Keepaway (b) 3D Keepaway (c) Nao Agent

Fig. 1: Comparison between 2D keepaway and 3D keepaway

Additional Challenges in 3D Robocup

- 2D robocup provides convenient primitive action such as **turn(angle)**, **dash(power)**, or **kick(power, angle)**
- Primitives in 3D robocup include **apply specified amount of torque on specified hinge**.
- To achieve a dash or kick, the agent has to figure out **correct sequence of torque** values to apply across all it's **22 hinges** over different timesteps

UT Austin’s 2011 Base Code

- Luckily we didn’t had to start from scratch!
- **Omnidirectional walk engine** based on a double inverted pendulum model
- A couple **basic skills for kicking**, one of which uses inverse kinematics
- **Particle filter** for localization and **Kalman filter** for tracking objects
- All **necessary parsing code** for sending/receiving messages from/to the server

How to get keepaway working in 3D domain

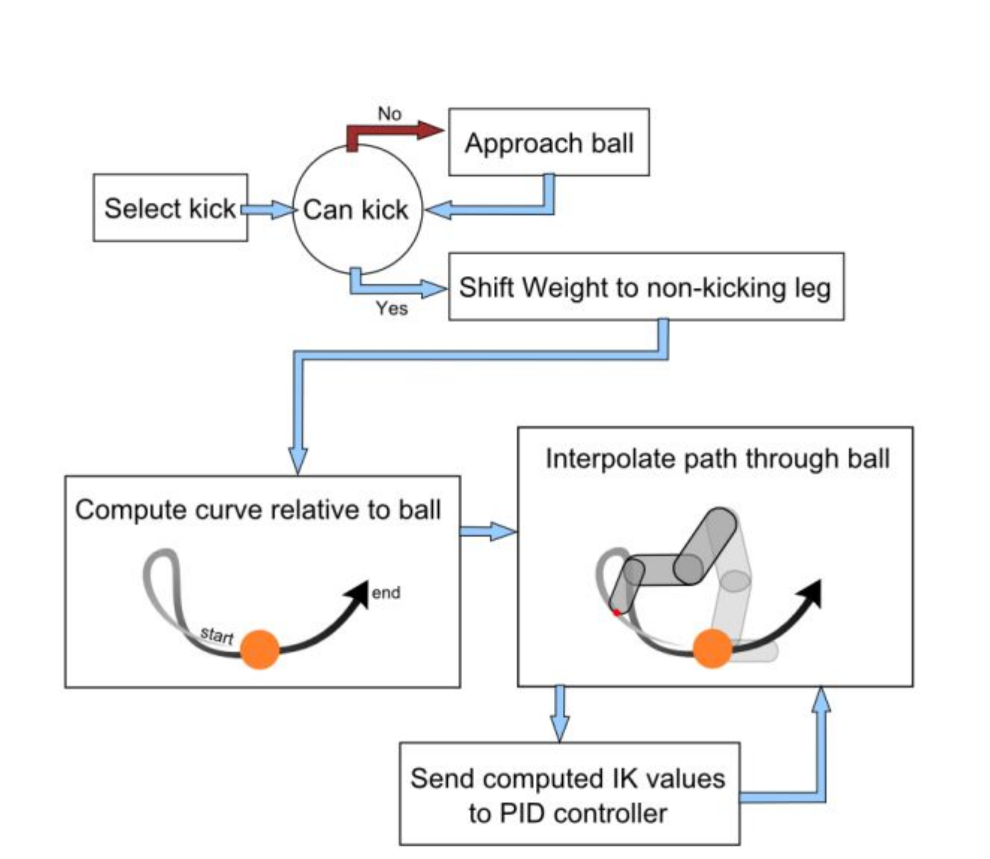
- **Optimise** existing **basic skills** as per keepaway requirements
- **Learn complex behaviours** such as getting possession of the ball
- **Learn high level decision making policy** for keepers

Kick Optimization

Kick Engine

For playing keepaway, a **robust, precise** and **mid-range** (since keepaway is played within a confined boundary) kick is required.

UT's kick need to be optimized

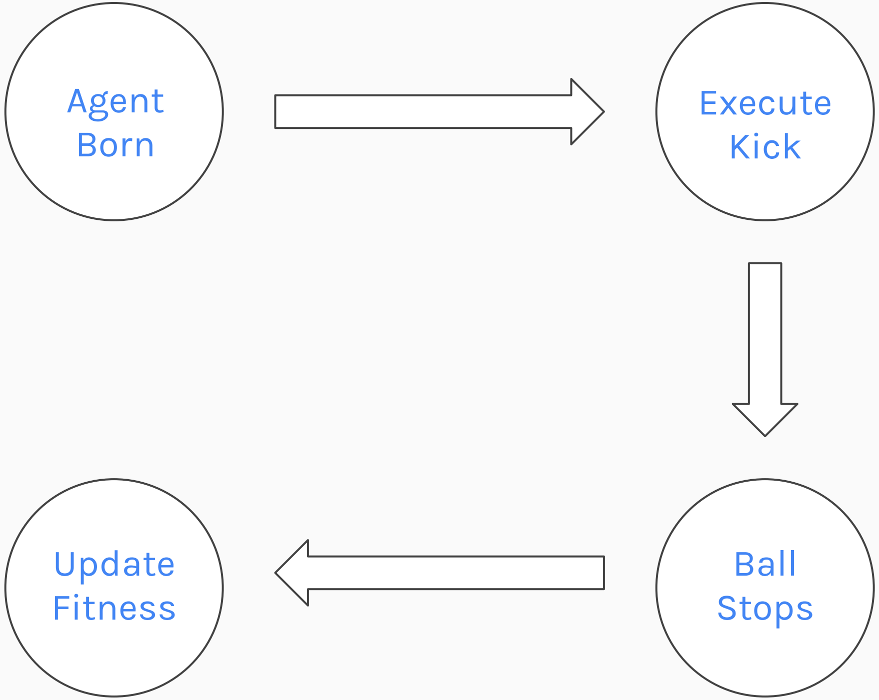


Kick Optimization

We optimize IK kick with respect to the **control points** of kick trajectory

Used **CMA-ES** for optimization

candidate parameter is evaluated over 12 episodes

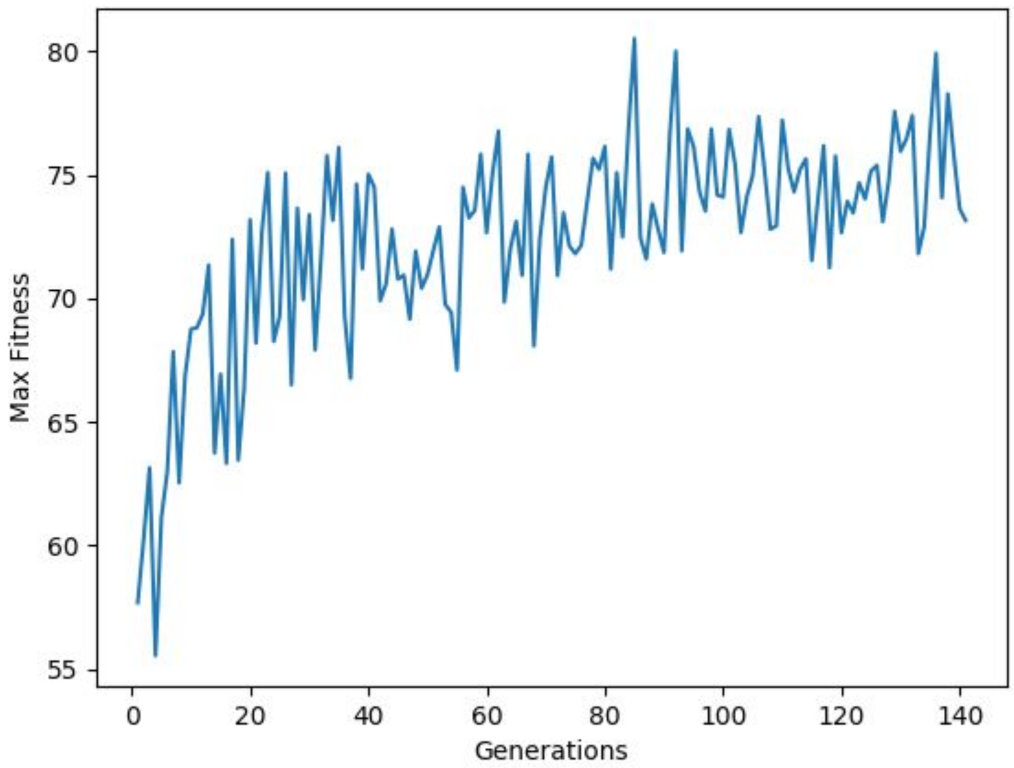


Fitness Function

- $time\_factor = episode\_end\_time - episode\_start\_time$
- $angle\_factor = 2^{-(angle(ball\_finish, ball\_start, target)^2/180.0)}$
- $distance\_factor = max(distance(ball\_start, ball\_finish), 6.0)$

$$episode\_fitness = \begin{cases} -1 & \text{Failure} \\ distance\_factor * angle\_factor / time\_factor & \text{Otherwise} \end{cases}$$

Learning Curve



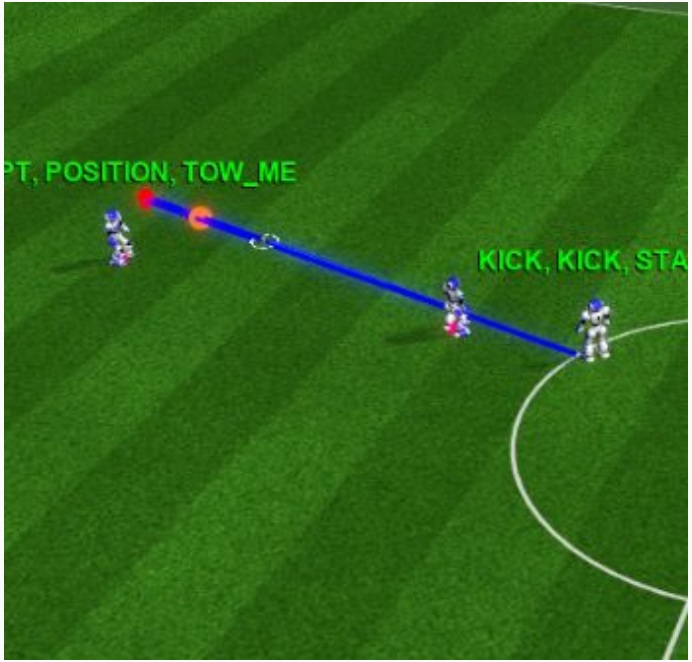
Learning to Get Possession of Ball

Getting Possession of Ball

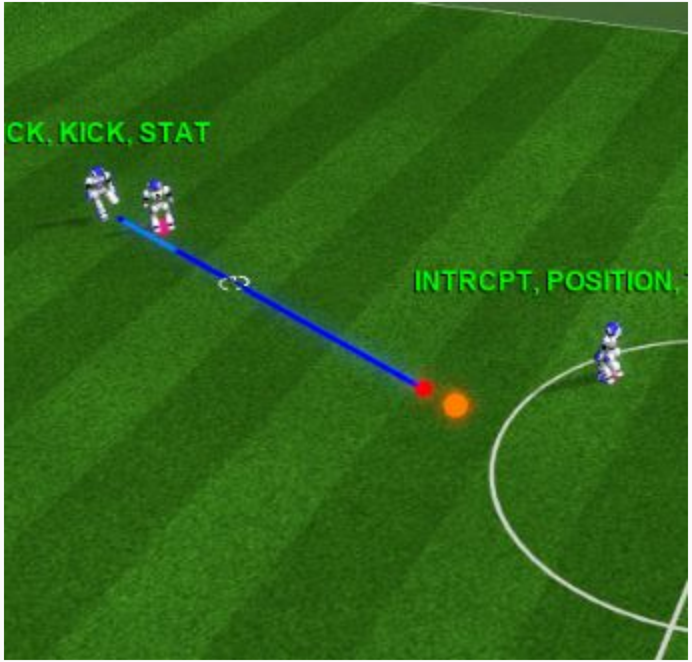
- Not easy! Different actions suitable for different scenarios
- We define following 4 primitive actions to choose from :
  - **INTERCEPT** : move(intercept point), where intercept point is defined as the point perpendicular to ball's trajectory from agent's position
  - **GO TO FINISH** : move(ballfinish), where ballfinish is ball's finish position
  - **POSITION** : try to position around the ball to prepare for the kick
  - **HOLD** : remain still at current location
- Need to learn which action to choose based on current world state

Getting Possession of Ball

Better To Intercept



Better To Go to Finish



Getting Possession of Ball

Complex Behaviour

- Any behaviour which requires **combination of primitive actions based on world state** to achieve the goal
- We represent a complex behaviour as a **state machine** whose **transitions are governed by the world state**
- ComplexBehaviour : (**invoke**, **abort**, **getSkill**, **action_map**, **state**, **nextstate**)

Control by ANNs

state transitions define the control of a complex behaviour

state transitions can be defined using an ANN

Algorithm 1: getSkill

Input: world state

Output: primtive action

$\boldsymbol{state} \leftarrow \boldsymbol{nextstate}$

$ann.\text{load}(world\_state)$

$output[1, .., n] \leftarrow ann.\text{activate}()$

$\boldsymbol{nextstate} \leftarrow arg\,max_{i \in \{1,..,n\}}\ output[i]$

if $\boldsymbol{state} \neq \boldsymbol{nextstate}$ **then**

  $action\_map[\boldsymbol{state}].\text{abort}()$

  $action\_map[\boldsymbol{nextstate}].\text{invoke}()$

return $action\_map[\boldsymbol{state}].\text{getSkill}()$

Optimising Control

- We can optimise ANN to suit our goal but **no well defined loss function**, too much variance
- Can **define a reward function** as how well a particular candidate did on the task, evaluate over few episodes (~20) to reduce variance
- We try to optimise the ANN using **Neuro Evolution of Augmenting Topologies** (NEAT)
- NEAT is a genetic algorithm for the generation of **evolving artificial neural networks**

Initialising NEAT with good candidates

- 3D simulations are **computationally expensive**, need to be sample efficient
- Start NEAT with good seed to **minimize the training time** and motivate the optimisation towards the **candidates that we expect to be good**
- **Imitate a human behaviour** of state transitions to generate good seed

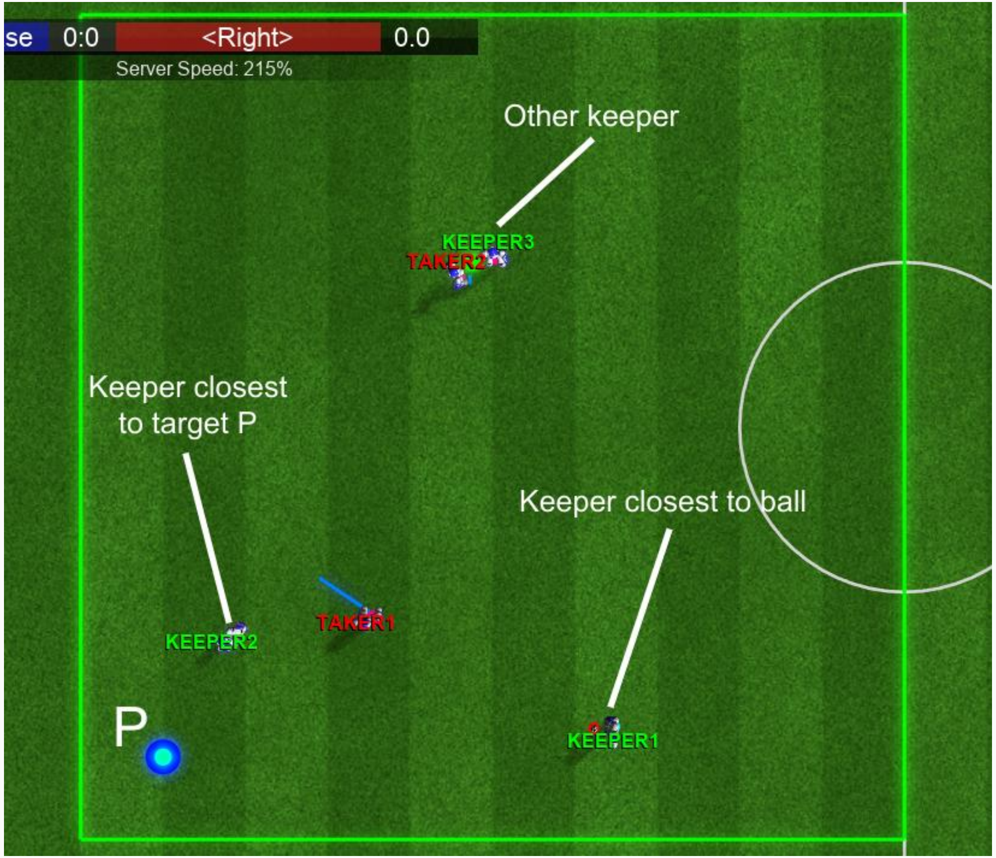
Results on Keepaway

3 vs 2 Keepaway : High-level decision policy

Keepers

- At any instant each keeper takes a role, named **K1**, **K2** and **K3**.
- **K1** : keeper closest to *ballfinish*, **K1** decides role of other players, let **K1** selects **T** as its kick target
- **K2** : keeper closest to **T**, tries to move to position **T**
- **K3** : remaining keeper, tries to go to its home position

Keepers



se 0:0 <Right> 0.0
Server Speed: 215%
Other keeper
KEEPER3
TAKER2
Keeper closest
to target P
Keeper closest to ball
TAKER1
KEEPER2
P
KEEPER1

K1 Keeper

- **K1** evaluates the best target position **T** and tries to get possession of the ball
- If **K1** has ball's possession and takers are far away, holds the ball
- Else attempts to kick at target **T**
- So, essentially the choices **K1** can learn are the choices of target **T** based on the state of keepaway
- We hope to learn better choices of target **T** to yield long episodes of keepaway

Takers

- Takers strategy is **fixed** and they don’t try to learn or improve their strategy
- **GREEDY+RANDOM** : the taker closest to the ball’s position greedily moves towards the ball
- The other taker randomly selects a keeper other than **K1** and follows this randomly selected keeper.
- To avoid thrashing, the random selection of the keeper to follow is only done when a pass is made

Takers

se 0:0 <Right> 0.0
Server Speed: 215%
KEEPER3
TAKER2
Random Taker
Closest to ball,
Greedy Taker
TAKER1
KEEPER2
P
KEEPER1

Mapping Keepaway onto NEAT optimisation

- Select **targets from continuous domain**, not feasible or desirable
- Select target from a **finite set S** of points spread out across the field
- **Action space very large for RL** methods to gather enough sample to learn
- learn a function **cost(F)** which **scores each point** in **S** based on features from **F**

Snapshot of Keepaway

Learning Evaluation Function

- Represent the evaluation function as a neural network that computes a real value for a target location **p** $\in$ **S** given input features
- Use NEAT to learn the neural network behind the evaluation function
- A particular candidate f is evaluated on **20 episodes of keepaway** and the reward for each episode is **number of passes** made in that episode

Results

Table 1: Comparison of hand coded vs learned evaluation function averaged over 100 episodes

Evaluation Function	Number of Passes	Hold Time
Hand-Coded	3.1 ± 0.062	$31.764 \pm 0.482s$
Learned	4.55 ± 0.129	$43.238 \pm 1.034s$

Demo

Thank You!