To appear in *Proceedings of the 5th Workshop on Humanoid Soccer Robots at Humanoids 2010*, Nashville, TN, USA, December 2010.

Optimizing Interdependent Skills for Simulated 3D Humanoid Robot Soccer

Daniel Urieli, Patrick MacAlpine, Shivaram Kalyanakrishnan, Yinon Bentor and Peter Stone
Department of Computer Science, The University of Texas at Austin
{urieli,patmac,shivaram,yinon,pstone}@cs.utexas.edu

***Abstract*— In several realistic domains an agent's behavior is composed of multiple interdependent skills. For example, consider a humanoid robot that must play soccer, as is the focus of this paper. In order to succeed, it is clear that the robot needs to walk quickly, turn sharply, and kick the ball far. However, these individual skills are ineffective if the robot falls down when switching from walking to turning, or if it cannot position itself behind the ball for a kick.**

This paper presents a learning architecture for a humanoid robot soccer agent that has been fully deployed and tested within the RoboCup 3D simulation environment. First, we demonstrate that individual skills such as walking and turning can be parameterized and optimized to match the best performance statistics reported in the literature. These results are achieved through effective use of the CMA-ES optimization algorithm. Next, we describe a framework for optimizing skills *in conjunction* with one another, a little-understood problem with substantial practical significance. Over several phases of learning, a total of roughly 100–150 parameters are optimized. Detailed experiments show that an agent thus optimized performs comparably with the top teams from the RoboCup 2010 competitions, while taking relatively few man-hours for development.

I. Introduction

As agents gain complexity and autonomy, automatic learning and optimization methods become attractive, as (a) they can improve and refine human intuition, especially in complex, dynamic environments, and (b) they demand significantly less labor to adapt to changes in the agent and environment. As most complex systems naturally decompose into smaller sub-units, for learning within such systems, it becomes convenient, even beneficial, to explicitly recognize such decomposition. In this paper we investigate the learning of agent behavior that can be decomposed into a *sequence* of atomic skills. Specifically we focus on optimizing multiple skills *within* each agent, and present a learning architecture for a humanoid robot soccer agent, which is fully deployed and tested within the RoboCup [3] 3D simulation environment.

In general, factors such as nonstationarity make it hard to provide strong theoretical guarantees when learning multiple behaviors. Therefore it becomes relevant to investigate such learning through empirical means. Our case study is performed on a complex domain, with realistic physics, state noise, multi-dimensional actions, and real-time control. In our test domain, teams of six autonomous humanoid robots play soccer in a physically realistic environment. Although each robot is ultimately controlled through low-level commands to its joint motors, we devise primitives for skills such as walking, turning, and kicking. In turn, such skills are strung together for implementing higher-level behaviors such as GoToTarget() and DriveBallToGoal(). It is quite clear that a behavior such as DriveBallToGoal() will be more successful if the robot can walk fast, turn quickly and sharply, and kick the ball with speed and accuracy. On the other hand, a very fast walk might tend to lead to a fall when transitioning into a turn; kicks lose their potency if the robot cannot accurately position behind the ball through precise side-walking and turning. The key idea in this paper is that skills can be optimized while respecting the tight coupling induced over them by high-level behaviors.

Robot soccer has served as an excellent platform for testing learning scenarios in which multiple skills, decisions, and controls have to be learned by a single agent, and agents themselves have to cooperate or compete. Although there is a rich literature based on this domain, most reported work primarily addresses (a) low-level concerns such as perception and motor control [5], [16], or (b) high-level decision-making problems [10], [18]. Thus the first contribution of our paper is a general methodology for optimizing the intermediate stratum of skills in an agent's control architecture. The volume of the space thus optimized (hundreds of parameters) marks a qualitative shift from a predominantly hand-coded approach to one significantly based on learning.

A second contribution of our paper is the light it sheds on designing objective functions ("fitness" functions) for optimization. On the one hand, "raw" statistics such as the precision and speed of soccer skills do not yield skills that operate well in unison. On the other hand, true objectives such as goal difference and win-loss record are too noisy to use effectively as a signal for learning. We demonstrate that carefully designed sequences of skills — such as those used by DriveBallToGoal() — can promote learning to achieve high-quality performance.

Finally, as an empirical contribution, we conduct detailed and extensive experiments related to our investigation. In particular, we compare several existing optimization methods, and find CMA-ES [8], a relatively recent addition to the literature, to be the most robust and effective. We also show evidence that conjunctive skill optimization can yield a very competitive soccer agent, which ranks among the top 8 teams from the RoboCup 2010 competitions.

The remainder of this paper is organized as follows. In Section II we describe our domain and agent's skills. Section III describes how individual skills are parameterized and set up for optimization, and reports the skill performance achieved through optimization. Section IV then presents our methodology for optimizing these skills in sequence. Comprehensive experimental results are presented both in Section III and in Section V. We conclude the paper with a summary and discussion in Section VI.

II. Domain and Agent's Skills

The RoboCup 3D simulation environment is based on SimSpark[4], a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine[2] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints used in the humanoid agents.

The robot agents in the simulation are homogeneous and are modeled after the Aldebaran Nao robot [1], which has a height of about 57 cm, and a mass of 4.5 kg. The agents interact with the simulator by sending actuation commands and receiving perceptual information. Each robot has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. In order to monitor and control its hinge joints, an agent is equipped with joint perceptors and effectors. Joint perceptors provide the agent with noise-free angular measurements every simulation cycle (20 ms), while joint effectors allow the agent to specify the direction and speed (torque) in which to move a joint. Although there is no intentional noise in actuation, there is slight actuation noise that results from approximations in the physics engine and the need to constrain computations to be performed in real-time. Visual information about the environment is given to an agent every third simulation cycle (60 ms) through noisy measurements of the distance and angle to objects within a restricted vision cone (120°). Agents are also outfitted with noisy accelerometer and gyroscope perceptors as well as force resistance perceptors on both feet. Additionally agents can communicate with each other every other simulation cycle (40 ms) by sending messages limited to 20 bytes.

At the lowest level of control, each robot is operated by specifying torques to its joints. As a more convenient abstraction, we implement PID controllers for each joint, which take as input a desired *target angle* and compute the appropriate torque for achieving it. In turn, skills use the PID controllers as primitives. The set of skills needed to develop a successful agent, and the focus of this paper, include walking (forwards, backwards, and sideways), turning, kicking, standing, goalie-diving and getting up after falling. Further, it is useful to explicitly breakdown skills such as walking forwards into several different speeds. Whereas we are able to manually program fairly successful goalie-diving and getting up skills, effective locomotion and kicking skills are harder to develop manually: in contrast to getting up and goalie-diving, successful locomotion and kicking require a combination of dynamic balancing, precision and high speed. Locomotion skills further need to be able to transition well to and from other skills. Thus, for these skills we devise templates with parameters, which are subsequently learned.

Bipedal locomotion has long been an active area of research. Pratt's thesis [15] provides an excellent overview of the field; Katić and Vukobratović [11] specifically survey intelligent control techniques used therein. A majority of the literature on bipedal locomotion focuses on model-based approaches. For instance, a humanoid robot is commonly modeled as an inverted pendulum [9], whose dynamics can be analyzed and used to plan trajectories. Recent approaches have also considered learning more complicated models, such as Poincaré maps [14]. Analytical modeling has indeed resulted in classical techniques — such as monitoring the "Zero Moment Point" of the robot [19] — which can resist noise in sensing, planning, and actuation, and small irregularities on the walking surface [13]. Even without explicit modeling of the dynamics, deviations from the intended trajectory can be constantly corrected through "closed-loop" control [7].

"Open-loop" approaches that do not rely on corrective feedback are typically simpler to implement and tend to yield faster walks, even if they are less robust to disturbances. However, in our simulation there is only minor noise in sensing or actuating joint angles (note that vision percepts are still noisy), and the soccer field is perfectly flat. Consequently we find it effective to develop open-loop skills for our agent. It must be noted that although the absence of significant actuation noise simplifies skill-development in our 3D simulation environment, in compensation the domain necessitates the development of an entire *suite* of soccer-related skills: multi-directional walks, turns, and kicks. Thus simulation enables us to investigate a concept that is relatively unexplored in the mainstream bipedal control literature. Even the few learning approaches within the 3D simulation environment have mainly been in the context of straight walking [17].

Each of our open-loop skills is implemented as a periodic state machine with multiple *key frames*, where a key frame is a static pose of fixed joint positions. To provide us flexibility in designing and parameterizing skills, we design an intuitive skill description language that facilitates the specification of keyframes and the waiting times between them. Below is an illustrative example describing the WalkFront skill (further explained in Section III).

```
SKILL WALK_FRONT

KEYFRAME 1
reset ARM_LEFT ARM_RIGHT LEG_LEFT LEG_RIGHT end
setTarget JOINT1 $jointvalue1 JOINT2 $jointvalue2 ...
setTarget JOINT3 4.3 JOINT4 52.5
wait 0.08

KEYFRAME 2
increaseTarget JOINT1 -2 JOINT2 7 ...
setTarget JOINT3 $jointvalue3 JOINT4 (2 * $jointvalue3)
wait 0.08
.
.
.
```

As seen above, joint angle values can either be numbers or be parameterized as `$<varname>`, where `<varname>` is a variable value that can be loaded after being learned. Note that due to left-right symmetry, some of these parameters influence multiple keyframes.

Before proceeding to details about our skill optimization, it is relevant to observe that alternative parameterizations of skills could also be conceived. For example, rather than direct control of joints, foot trajectories could be parameterized and tracked using inverse kinematics [12]. We plan to explore such variations in future work.

III. Optimizing Individual Skills

In this section we describe how we optimize a forward walk, which essentially illustrates the basic procedure adopted for optimizing any of our skills. As a starting point for subsequent optimization, we achieve a relatively stable front walk by programming the robot to raise its left and right feet alternately to a certain height above the ground, swinging them slightly forward, and then retracting them to their initial configurations. This hand-coding exercise results in slow but stable skills, which are not very competitive themselves, but which serve as useful seeds for further optimization. Our walk consists of four key frames through which the agent periodically loops. General intuition for a straight and stable walk suggests that the legs should move in a symmetric and periodic manner. For this reason the joint positions of our first two frames are the same as our next two, except that the positions of the left and right legs are mirrored. Based on informal experimentation we decide to optimize three joint positions in each leg for each key frame as they appear to be the most meaningful for a forward walk. These joints are the hip moving the leg forward and backwards, knee, and ankle moving the foot up and down. This provides a 12-dimensional parameter space to optimize, as we have 6 joint positions for each frame (3 for each leg), across two frames (as frames 3 and 4 are just mirrored values of frames 1 and 2). See Figure 1 for screenshots with the joints we are optimizing circled. We set the time to transition between key frames to be 80ms. This time was also determined by informal experimentation and gives the agent a walk cycle duration of 320ms (4x80ms).

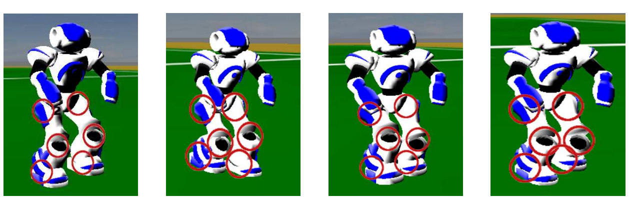


Fig. 1. Nao robot walk frames with joints we are optimizing circled.

In order to evaluate the performance of a forward walk, we measure the distance in the forward direction the agent can travel in 15 seconds. Our performance metric of displacement in the forward direction not only rewards speed, but it also encourages straight walks (as the shortest distance to walk is a straight line) and penalizes for lack of robustness (if the agent falls over it takes several seconds for it to stand up again). These measurements are taken in an automated fashion using a distributed computing cluster. Our setup on the distributed computing cluster allows us to run massive amounts of simulations in parallel, which is necessary in order for our learning algorithms to complete in a reasonable amount of time. We conducted a comprehensive comparison between four learning algorithms: hill climbing (HC), cross-entropy method (CEM) [6], genetic algorithm (GA), and covariance matrix adaptation evolution strategy (CMA-ES) [8]. While the details of the comparison are left out of this paper due to space constrains, the main result was that CMA-ES performed significantly better then all the other algorithms. Table I shows the best results we achieve when optimizing each of our main skills using the method described above and using the CMA-ES algorithm. To the best of our knowledge, these results are among the fastest that have been achieved in our domain [17].

IV. Optimizing Sequences of Skills

Whereas the results from Table I signify that our parameterized skills can effectively be optimized using CMA-ES, the job of deploying these skills to play soccer remains unfinished. Fast locomotion skills, however stable they are when executed individually, result in frequent falls of the robot if integrated directly. To see why, consider a typical log of the skills invoked (every 320 ms, as described in the previous section) by the agent during soccer play: ... WalkFront, WalkFront, Turn(R), Turn(R), Turn(R), WalkFront, WalkFront, WalkFront, Turn(L), Turn(L), WalkBack, WalkBack, ... The trace shows that skills are highly interleaved, with frequent transitions between them. In game scenarios, the same skill is seldom executed for more than a few consecutive cycles. Therefore, optimizing skills in isolation does not necessarily benefit their *combined operation*.

In order to optimize sequences of skills to work together, carefully designed constraints are necessary. We begin by revising the evaluation criterion used by the learning process. Ideally, when learning a skill, it would be best to evaluate it with respect to our ultimate goal: the team's win-loss record or mean goal difference against a set of opponents. However, as this is an extremely noisy measure, the number of runs needed in order to obtain reliable performance estimates becomes impractical. A much less noisy measure, which still aligns well with the team's objective, is the time taken by a single agent to score a goal on an empty field. We denote this behavior DriveBallToGoal(), and the associated evaluation metric time-to-score. Pseudo-code for DriveBallToGoal() is as follows:

```
function DriveBallToGoal()
  if robotDistanceFromBall > threshold_0
    getRoughlyBehindBall()
```

TABLE I

Performance statistics for various skills optimized using CMA-ES. In this table and all subsequent ones, entries within parentheses correspond to one standard error.

Skill	Statistic	Performance
WalkFront	Speed	1.07(.00) m/s
WalkBack	Speed	1.03(.00) m/s
WalkSide	Speed	.62(.01) m/s
Turn	Angular speed	112.03(.24) °/s
Kick	Ball displacement	5.09(.07) m

```
else
  chooseKickDirectionAndType()
  computeThresholdsForPositioning()
  # Position to kick / dribble:
  if distanceToPosition > threshold_1
    walkFront()
  elseif robotOffsetFromKickDirection > threshold_2
    turn()
  elseif lateralLegAlignementWithBall > threshold_3
    sideWalk()
  else
    kickOrDribble()
```

We use this behavior for our evaluations, as it achieves a good balance between eliminating noisy effects such as the actions of other players, while still requiring the agent to combine its basic skills in a complex, realistic manner. Later, we show empirical results validating the choice of time-to-score as an evaluation metric while learning skills.

Several skills are used during a learning evaluation through DriveBallToGoal(). However, it would be inefficient to try and learn all of them at once, due to the high dimensionality of the search space (roughly 100 – 150 parameters). Instead we use a more efficient approach, which learns one skill (roughly 12 parameters) at a time, while keeping others fixed. This process results in a sequence of incremental improvements in the agent, with the crucial invariant property that at any time all the skills work well together. In particular the optimization process improves the agent's speed while keeping it stable, as falls typically result in poor time-to-score values. In turn, the amount each individual skill can be optimized is limited by the need to cooperate with other skills.

Apart from goalie dives and getting up skills, all the skills used by our final agent are optimized. Yet, for the purposes of this paper, we present an isolated study of our optimization procedure involving only forward and backward walks, namely WalkFront and WalkBack, respectively. We start with a base agent that uses basic, hand-coded versions of these skills. Let us call this agent A0. Under A0 these skills are not very fast, but they ensure relative stability during locomotion and skill transitions. The idea is to use A0 as a seed for successive optimizations. Figure 2(a) presents a skill transition diagram, which shows the main skills of agent A0 along with the legal transitions between them (marked by arrows). Notice that the agent can only invoke Kick if it is already standing; nor can it transition into a skill other than Stand after executing Kick. In Figure 2(a) the walking skills of A0 are suffixed "_S" to denote that they are "slow".

We improve upon A0 in five incremental steps, each step creating a new agent based on the agent that resulted from the previous step. We denote the resulting agents A1, A2, A3, A4, and A5. The first improvement, A1, is created from A0 by optimizing "WalkFront_S" using CMA-ES, under the time-to-score measure. Consider that while WalkFront_S is being optimized under this measure, we are searching for a set of parameters that both improve speed and maintain stability. The need to maintain stability while cooperating with all other skills puts multiple constraints on WalkFront_S and therefore limits how fast WalkFront_S can get. We address this problem in A2, by "decoupling" from WalkFront_S an additional skill called WalkFront_F ("F" denoting "fast"). As seen in Figure 2(c), we constrain the behavior of agent A2 such that WalkFront_F can only be invoked following WalkFront_S, and to transition to any other skills, it must first transition into WalkFront_S. The skills WalkFront_S and WalkFront_F have exactly the same template, and initially the same parameter values. However, optimizing the parameters of WalkFront_F after first optimizing WalkFront_S (under A1) allows the agent to achieve greater speed while retaining its stability. These properties result from the fact that WalkFront_F is unconstrained by most of the skills that constrain WalkFront_S.

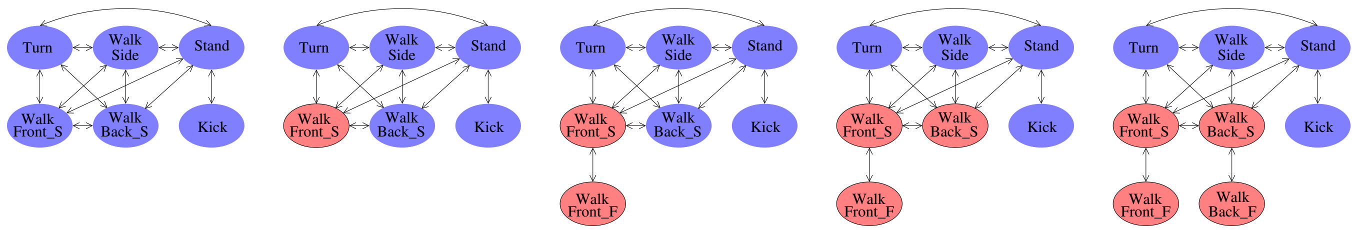
Results in Section V demonstrate tangible gains consistent with our progressive refinements from A0 to A1 to A2. Indeed the trend is carried forward to agents A3 (Figure 2(d)) and A4 (Figure 2(e)), which are obtained based on a similar decoupling procedure applied to the WalkBack skill. Recall that agents A1 through A4 are all obtained solely by skill optimizations of one skill at a time, starting from the seed agent A0. To obtain our final agent, A5, we take A4 and *manually* retune thresholds and the logic for selecting and invoking our new learned skills in order to best utilize them to their full potential. For example, a change in skill speeds can change the robot's stopping distance, which in turn affects the threshold for the decision of whether to continue to WalkFront, as can be seen in the DriveBallToGoal() pseudocode. While the tuning is done here manually, it could potentially be automated and learned. However, in this paper we focus on skill learning, and leave the learned tuning as possible future work.[1]

Note that agents A0 through A5 all use the same skills, apart from WalkFront and WalkBack. The turns and side walks used were also optimized in the manner described above and were already integrated into our agent A0. It is worth mentioning, however, that time-to-score does *not* serve as an ideal fitness measure while optimizing kicks, as the kick skill is used only a small fraction of time, and most of the time is spent on locomotion and positioning behind the ball. Since Kick is only executed after an intermediate Stand skill, we optimize kicks by starting the robot behind the ball, using the distance covered by the ball in the kick direction as an informative evaluation measure.

V. Experimental Evaluation

In the previous section, we described how we used two main ideas for learning and optimizing skills: the idea of optimizing a skill under the constraints of cooperating with other skills, and the idea of skill decoupling. This section shows that our skill optimization process achieved tangible gains, that were reflected directly in the agent's performance with respect to its ultimate objective: its win-loss record or goal difference against a set of opponents.

[1]Videos showing optimized skills and behavior are provided at the following URL: `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2010/html/skilloptimization2010.html`.



(a) A0 (b) A1 (c) A2 (d) A3 (e) A4

Fig. 2. Constraints on transitions between skills represented as state diagrams. For Agent A0 neither the WalkFront_S nor the WalkBack_S skills is optimized; the former is optimized (shown with thick border) under A1. Further skills are added and optimized subsequently under agents A2, A3, and A4. Agent A5 is identical to A4, except for retuning thresholds and the logic for selecting and invoking our new learned skills.

We ran three sets of experiments, in which we measured our agent both with respect to the time-to-score measure and with respect to its actual game performance, and compared the results with released binaries from RoboCup 2010. In the first set of experiments we measured the progress achieved by each step of our optimization process, which started from the seed agent A0, continued by creating the agents A1-A4 by optimizing one skill at a time, and finally tuned A4 to be the final agent A5. Table II shows the results of playing agents A0-A5 against each other. Each cell in the table shows the mean goal difference along with the standard error, averaged over 100 full games. It can be seen that every agent plays better than its predecessors. This demonstrates how our skills optimization process directly resulted in better game performance.

In the second set of experiments we compared the time-to-score performance of our initial agent A0, our final agent A5, and a set of released agent binaries from RoboCup 2010. In each experiment, we placed the ball in the middle of the field, which is 9 m from the goal, and then placed the agent 1 meter behind the ball. We then measured the time it takes the agent to score a goal. Table III shows the mean time it takes the agents to score from this position, averaged over 500 runs, along with the standard error. Our agent A5 is ranked second with a mean time of 34.49 seconds, whereas the top agent's mean time to score is 31.08 seconds. Note that A0 is ranked in the middle of the table with a time of 63.52. Agents A1–A4, which are not shown in the table achieved times that are between A0 and A5.

In the third set of experiments, we tested our agents A0 and A5 in playing full games against the released RoboCup 2010 agent binaries. The results are shown in Table IV. The leftmost column shows the row agent's rank in RoboCup 2010. The rightmost columns show the results achieved by agents A0 and A5, when playing against RoboCup binaries. Each cell shows the mean goal difference between a row agent and a column agent, averaged over 100 full games, along with the standard error. Note that negative values (in bold) mean a positive goal difference for our agent, therefore the bolded part of the table is where our agent performed better than the row agent.

Two interesting facts can be observed in Table IV. The first one is the correlation between the actual game performance and the time-to-score measure (Table III). An agent, whether our agent or another team's agent, with good game performance usually had good time-to-score performance. Recall that while optimizing our agent's skills, we used the time-to-score measure along with the DriveBallToGoal() behavior as a less-noisy alternative for measuring real game performance. Here we confirmed that while doing so, much of the complexities of real game scenarios that are relevant to skills execution were still retained. Therefore the time-to-score measure is both effective, as it correlates with game performance, and efficient, due to the reduced noise. Note that the correlation is not expected to be perfect: in real games there are factors like decision-making strategies, formations, defensive tactics and more, that affect the game performance but do not reflect in the DriveBallToGoal() behavior. The second interesting fact is that our final agent, A5, was ranked in the table among the top 8 teams of RoboCup 2010. As this ranking was achieved mainly using our skill optimization process, with some additional tuning, this demonstrates the effectiveness of our suggested method of optimizing skills under constraints.

TABLE II

GAME RESULTS BETWEEN AGENTS A0 THROUGH A5. ENTRIES SHOW THE GOAL DIFFERENCE (ROW − COLUMN) FROM 10 MINUTE GAMES.

	A0	A1	A2	A3	A4
A5	2.11(.10)	.77(.10)	.70(.10)	.58(.09)	.48(.08)
A4	1.66(.10)	.46(.08)	.15(.07)	.03(.07)	
A3	1.67(.10)	.28(.08)	.01(.08)		
A2	1.33(.10)	.20(.07)			
A1	1.23(.10)				

VI. SUMMARY AND DISCUSSION

In several practical tasks an agent's behavior is composed of qualitatively distinct components. Can this natural decomposition be used as a means to scale learning to complex tasks? In this paper we presented a successful case study of doing so in the context of humanoid robot soccer. In particular we focused on the intermediate "skills" layer of a soccer agent's architecture. Together, the skills of a soccer agent constitute a rich and complex aspect of behavior, which it would be impractical to optimize as a single monolithic block. We carefully engineered skills and rules for transitions, and showed that optimizing components in an incremental manner could significantly improve performance. Each skill has 10–20 parameters; overall the number of parameters optimized is around 100–150.

We believe our case study is a compelling example for the methodology of decomposing a large learning problem into

TABLE III

TIME TO SCORE ON AN EMPTY FIELD, STARTING THE CENTER OF THE FIELD. EACH ROW CORRESPONDS TO A0, A5 OR AN AGENT FROM THE ROBOCUP 2010 COMPETITION. AVERAGES ARE OVER 500 RUNS.

Agent	Time-To-Score/s
Apollo3d	31.08 (1.46)
A5	**34.49 (0.89)**
RoboCanes	36.18 (1.40)
NaoTH	36.75 (1.63)
UTAustinVilla	37.20 (0.89)
FCPortugal	47.54 (1.94)
SEURedSun	52.11 (2.49)
A0	**63.52 (1.05)**
Little Green Bats	71.02 (1.96)
FutK	77.89 (4.19)
BeeStanbul	98.56 (3.63)
Nexus3D	152.76 (5.15)
RoboPub	291.86 (1.17)
NomoFC	295.48 (1.32)
Bahia3D	300.01 (0.00)
Alzahra	300.01 (0.00)

TABLE IV

FULL GAME RESULTS, AVERAGED OVER 100 GAMES. EACH ROW CORRESPONDS TO AN AGENT FROM THE ROBOCUP 2010 COMPETITIONS, WITH ITS RANK THEREIN ACHIEVED. THE TWO RIGHTMOST COLUMNS CORRESPONDS TO OUR BASE AGENT A0 AND FINAL AGENT A5. ENTRIES SHOW THE GOAL DIFFERENCE (ROW − COLUMN) FROM 10 MINUTE GAMES. GOAL DIFFERENCES IN FAVOR OF A0 AND A5 ARE SHOWN IN BOLD.

Rank	Team	A0	A5
1	Apollo3d	4.29 (.17)	1.88 (.13)
2	NaoTH	3.79 (0.14)	1.85 (0.10)
4	BoldHearts	3.15 (0.13)	0.08 (0.11)
5-8	SEURedSun	1.93 (0.13)	1.16 (0.1)
5-8	RoboCanes	1.81 (0.12)	0.38 (0.09)
5-8	FCPortugal	1.57 (0.11)	**-0.43** (0.09)
9-16	UTAustinVilla	1.54 (0.09)	**-0.9** (0.09)
9-16	FutK	0.23 (0.06)	**-2.14** (0.1)
9-16	BeeStanbul	**-0.76** (0.07)	**-4.08** (0.11)
9-16	Nexus3D	**-1.67** (0.06)	**-4.08** (0.09)
9-16	Little Green Bats	**-1.84** (0.08)	**-5.0** (0.11)
9-16	NomoFC	**-3.62** (0.09)	**-7.07** (0.09)
17-20	Bahia3D	**-3.59** (0.08)	**-7.49** (0.1)
17-20	RoboPub	**-5.25** (0.08)	**-7.92** (0.1)
17-20	Alzahra	**-6.39** (0.08)	**-10.59** (0.09)

components and devising informative objective functions. Several practical systems resemble a soccer agent's control hierarchy, and often are indeed evaluated ultimately through success (win) and failure (loss). This paper also leads to recommendations for an optimization framework and experimental support for the CMA-ES algorithm, which can serve as a useful starting point for related undertakings.

The RoboCup 3D simulation environment engenders the novel research question of developing a suite of interacting humanoid robotic skills, a relatively unexplored question in the literature, which this paper addresses. Our demonstration specifically finds appeal for developing humanoid robot soccer teams by investing significantly in learning and optimization. Our detailed experimental results provide conclusive evidence for the improvements achieved with each incremental optimization, and the final agent we develop ranks among the top eight teams from the RoboCup 2010 competitions. The human labor involved in developing our agent is relatively low compared to the CPU time spent optimizing skills, which is on the order of 100,000 hours.

In future work we intend to further extend the scope of learning within our agent by replacing currently hand-coded components (such as fine positioning and getting up). For our basic locomotion skills, it is also relevant to consider alternative parameterizations that involve closed-loop control and inverse kinematics. Such approaches are likely to eventually extend the reach of our learning paradigm to hardware platforms when using simulators that more precisely model physical robots.

ACKNOWLEDGEMENTS

We thank Suyog Dutt Jain for his contributions to early versions of this work. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-0917122), ONR (N00014-09-1-0658), DARPA (FA8650-08-C-7812), and the FHWA (DTFH61-07-H-00030). This research was also supported in part by the NSF under CISE Research Infrastructure Grant EIA-0303609.

REFERENCES

[1] Aldebaran Humanoid Robot Nao. `http://www.aldebaran-robotics.com/eng/`.
[2] Open Dynamics Engine. `http://www.ode.org/`.
[3] RoboCup. `http://www.robocup.org/`.
[4] SimSpark. `http://simspark.sourceforge.net/`.
[5] S. Behnke, M. Schreiber, J. Stückler, R. Renner, and H. Strasdat. See, walk, and kick: Humanoid robots start to play soccer, 2006.
[6] P. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, February 2005.
[7] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. *Int. Journal of Robotics Research*, 27(2):213–228, 2008.
[8] N. Hansen. *The CMA Evolution Strategy: A Tutorial*, January 2009.
[9] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by a simple three-dimensional inverted pendulum model. *Advanced Robotics*, 17(2):131–147, 2003.
[10] Shivaram Kalyanakrishnan and Peter Stone. Learning complementary multiagent behaviors: A case study, 2009. To appear.
[11] D. Katić and M. Vukobratović. Survey of intelligent control techniques for humanoid robots. *Journal of Intelligent Robotic Systems*, 37(2):117–141, 2003.
[12] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion, 2004.
[13] Ç. Meriçli and M. Veloso. Biped walk learning through playback and corrective demonstration, 2010. To Appear.
[14] J. Morimoto and C. G. Atkeson. Nonparametric representation of an approximated poincaré map for learning biped locomotion. *Autonomous Robots*, 27(2):131–144, 2009.
[15] J. E. Pratt. *Exploiting Inherent Robustness and Natural Dynamics in the Control of Bipedal Walking Robots*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2000.
[16] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.
[17] N. Shafii, L. P. Reis, and N. Lao. Biped walking using coronal and sagittal movements based on truncated fourier series, January 2010.
[18] P. Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon Univerity, Pittsburgh, PA, USA, December 1998.
[19] M. Vukobratović and B. Borovac. Zero-moment point - thirty five years of its life. *Int. Journal of Humanoid Robotics*, 1(1):157–173, 2005.