Motion Capture and Contemporary Optimization Algorithms for Robust and Stable Motions on Simulated Biped Robots

Andreas Seekircher, Justin Stoecker, Saminda Abeyruwan, and Ubbo Visser

University of Miami, Department of Computer Science,
1365 Memorial Drive, Coral Gables, FL, 33146 USA
{aseek,justin,saminda,visser}@cs.miami.edu

Abstract. Biped soccer robots have shown drastic improvements in motion skills over the past few years. Still, a lot of work needs to be done with the RoboCup Federation's vision of 2050 in mind. One goal is creating a workflow for quickly generating reliable motions, preferably with inexpensive and accessible hardware. Our hypothesis is that using Microsoft's Kinect sensor in combination with a modern optimization algorithm can achieve this objective. We produced four complex and inherently unstable motions and then applied three contemporary optimization algorithms (CMA-ES, xNES, PSO) to make the motions robust; we performed 900 experiments with these motions on a 3D simulated Nao robot with full physics. In this paper we describe the motion mapping technique, compare the optimization algorithms, and discuss various basis functions and their impact on the learning performance. Our conclusion is that there is a straightforward process to achieve complex and stable motions in a short period of time.

1 Introduction and Related Work

Generating motions on a humanoid robot that operates under the constraints of physics is a time-consuming process; attempts to create even simple motions by manually adjusting parameters are tedious and often end in failure [16]. Our idea is to use motion capture to record and map human motions to a humanoid robot. The immediate problem with this approach is that humans and robots do not share motor capabilities, range of motion, dimensions, mass, and other physical attributes. For the purposes of our experiments, we assume the dimensions and body part masses of the human and robot are roughly equivalent; our focus is on the range of motion. The goal of the motion processing stage is to map from human motion space to a specific robot's motion space.

Several systems exist that enable effective human motion tracking. Perhaps the most familiar of these systems is marker-based optical motion capture: a user typically wears a suit with several reflective markers that are recorded by several overhead cameras, and the positions are triangulated. An example of motion mapping using an optical marker system with the Nao robot is demonstrated in [15]. One major downside to these systems is that they require large labs with expensive equipment and software. Our motion capture experiments were

X. Chen et al. (Eds.): RoboCup 2012, LNAI 7500, pp. 213–224, 2013.

performed with the Microsoft Kinect sensor[1]; while not as accurate as more expensive platforms, we believe the Kinect provides a sufficient level of detail and is easily accessible to researchers without the funds or space for a dedicated motion capture lab.

Even with a system that provides low-noise tracking, a significant challenge remains in stabilizing the robot when motors are adjusted; mapping of human to robot joints, particularly in the legs, will often result in the robot falling over. Kim et al. [9] produce stable whole-body motions from motion capture by imitating a zero moment point (ZMP) trajectory of a simplified human model and dynamically adjusting the pelvis for balance. Amor et al. [1] mention the use of evolutionary algorithms to adjust the features of a mapped motion until the result is stable. Grimes et al. [5] learn a nonparametric model of forward dynamics from constrained exploration to infer actions and full-body imitation. Many other authors do not attempt to solve the balancing problem and focus entirely on mapping the upper body. After acquiring motions either manually or with other methods, the mapped robot motions need further optimization in order to achieve maximum performance and/or robustness [10].

The Covariance Matrix Adaption Evolution Strategy (CMA-ES) algorithm [6,8] is one of the most widely used algorithms for parameter optimization. The family of Natural Evolution Strategies (NES) [17] algorithms are an alternative to CMA-ES in order to perform real-valued black box function optimization. For medium size dimensions, with highly correlated parameters, the exponential NES (xNES) [3] empirically shows significant performance compatible with CMA-ES. Particle swarm optimizations [7] are simple and yet effective algorithms for optimizing a wide range of functions. We use particle swarm optimization (PSO) for both our biped walking engine [11] and as an alternative method that can be applied to our motions.

2 Human Motion Capture

The Kinect itself does not generate motion capture (MoCap) information, but it provides color and depth images (RGB-D) that can be used to track a user's body. Microsoft's Kinect SDK[2] and an open source alternative, OpenNI [13], both implement skeletal tracking algorithms. OpenNI is a framework that provides an interface to a variety of *natural interaction* (NI) devices, such as vision or audio sensors, that record motion and sound for the purpose of human-computer interaction. Rather than directly providing implementations for all imaginable sensors, both low-level and high-level features of the OpenNI API are enabled by middleware packages. We chose to use OpenNI over the Kinect SDK as it can be used with non-Windows operating systems and provides access to existing and future NI devices, such as the Xtion Pro[3]. The PrimeSense NITE [14] middleware enables skeleton tracking for the Kinect sensor.

[1] `http://www.microsoft.com/en-us/kinectforwindows/`
[2] `http://www.microsoft.com/en-us/kinectforwindows/develop/beta.aspx`
[3] `http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO/`

When OpenNI is configured to provide user tracking, data is presented as a skeleton model that approximates the motions of a human user. This skeleton model is defined by fifteen joints, each containing a position in sensor space; these joints are shown in Fig. 1a. To map the OpenNI user skeleton to the simulated Nao model, we first calculate two local coordinate systems for the user skeleton in terms of the vectors $\mathbf{f}$ (forward), $\mathbf{r}$ (right), and $\mathbf{u}$ (up); one coordinate system has the upper torso as the origin, and the second has the lower torso as the origin. For the upper body, which is used to calculate the arm angles: $\mathbf{f} = (\mathsf{l\_shoulder} - \mathsf{torso}) \times (\mathsf{r\_shoulder} - \mathsf{torso})$, $\mathbf{r} = \mathsf{r\_shoulder} - \mathsf{neck}$, and $\mathbf{u} = \mathbf{r} \times \mathbf{f}$. For the lower body orientation, which is used to calculate the leg angles: $\mathbf{f} = (\mathsf{r\_hip} - \mathsf{torso}) \times (\mathsf{l\_hip} - \mathsf{torso})$, $\mathbf{r} = \mathsf{r\_hip} - \mathsf{l\_hip}$, and $\mathbf{u} = \mathbf{r} \times \mathbf{f}$. Finally, the $\mathbf{f}, \mathbf{r}, \mathbf{u}$ vectors for both the upper and lower body are normalized to unit length.

(a) (b)

Fig. 1. In (a), the OpenNI user skeleton model in the calibration pose. In (b), the physical Nao model with all joints at 0 degrees rotation. Hinge joints in the Nao are represented as cylinders through the respective rotation axis.

Once the skeleton coordinate systems are established, Euler angles are computed for the joints in the Nao model. Our mapping approach uses the vectors between skeleton joint positions to calculate the Nao joint angles; this approach can be extended to any robot model consisting of revolute joints. Inverse kinematics could be used as an alternative approach to determine joint angles, although it introduces a degree of unpredictability: the trajectory of intermediate joints in a kinematic chain are not guaranteed to follow the motion of the human. Furthermore, our goal is not to position the end effectors of the robot, but instead to ensure the relative angles of body parts are correct; a 90° bend in the human's elbow should result in a 90° bend in the robot's elbow. For these reasons, a direct calculation of the joint angles is the most appropriate. Unfortunately, the Nao's head and foot angles must be ignored, as the skeleton does not provide enough information to determine their orientations (see Fig. 1).

Each Nao arm has four joints that apply rotation in the following order: shoulder pitch (θ_s), shoulder yaw (ψ_s), arm roll (φ_a), arm yaw (ψ_a). Fig. 2 illustrates the calculation of these angles for the right arm. Using the joints of

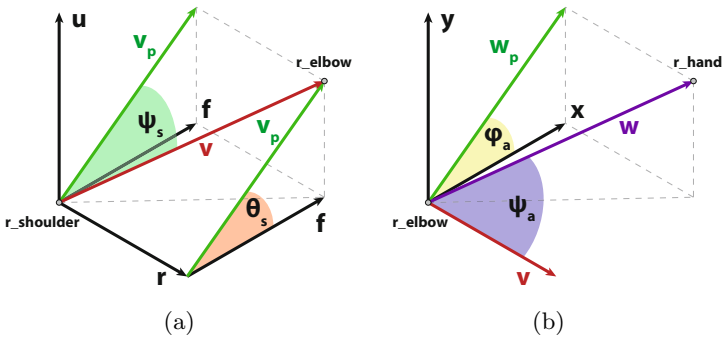


Fig. 2. Using OpenNI skeleton joint vectors calculate to Euler angles for the Nao joints θ_s = r_shoulder_pitch, ψ_s = r_shoulder_yaw, φ_a = r_arm_roll, and ψ_a = r_arm_yaw

the OpenNI skeleton, the vector from r_shoulder to r_elbow, $\mathbf{v}$, is projected onto the plane spanned by $\mathbf{f}$ and $\mathbf{u}$ (upper body) to get $\mathbf{v_p} = \mathbf{f}(\mathbf{f} \cdot \mathbf{v}) + \mathbf{u}(\mathbf{u} \cdot \mathbf{v})$. The shoulder pitch $\theta_s = \angle(\mathbf{f}, \mathbf{v_p})$, where the notation $\angle(\mathbf{a}, \mathbf{b})$ means the angle between vectors $\mathbf{a}$ and $\mathbf{b}$. The shoulder yaw $\psi_s = \angle(\mathbf{v}, \mathbf{v_p})$. After the shoulder joint angles are calculated, the arm joints are found using the same process. If the arm has no roll, it rotates (yaw) in the plane with $\mathbf{y} = \mathbf{v} \times \mathbf{v_p}$ as the normal, and $\mathbf{v}$ and $\mathbf{x} = \mathbf{y} \times \mathbf{v}$ as basis vectors; when roll is introduced, this plane is rotated around the $\mathbf{v}$ vector. The amount of roll can be found by projecting $\mathbf{w}$, the vector from r_elbow to the r_hand, onto the plane spanned by $\mathbf{x}$ and $\mathbf{y}$ to get $\mathbf{w_p} = \mathbf{x}(\mathbf{x} \cdot \mathbf{w}) + \mathbf{y}(\mathbf{y} \cdot \mathbf{w})$. The roll $\varphi_a = \angle(\mathbf{x}, \mathbf{w_p})$, and the yaw $\psi_a = \angle(\mathbf{v}, \mathbf{w})$.

For the legs, we observe that the hip, knee, and foot joints form a plane in space. The hip_yawpitch and hip_roll angles establish the orientation of this plane, and the hip_pitch and knee_pitch angles rotate the leg within this plane. The current thigh vector (knee - hip) and tibia vector (foot - knee) can be used to determine all angles for the leg. We initialize a lookup table to store the hip_yawpitch and hip_roll angles as well as the thigh vector: forward kinematics is used to iterate over possible combinations of these angles, and the normal vector of the leg plane is used as the key. To retrieve the hip_yawpitch and hip_roll angles during mapping, the current leg normal (the cross product of the thigh and tibia vectors from the skeleton) is compared with normals in the lookup table. The knee_pitch is simply the angle between the thigh and tibia vectors. Finally, the hip_pitch angle is calculated as the angle between the thigh vector stored in the lookup table and the current thigh vector.

3 Motion Optimization

The MoCap framework provides a set of traces for each motion. These motions have variable durations, and are inherently noisy. The MoCap framework captures traces only for most of the angles, but any unknown angles (head and foot) default to zero. The direct replay of the captured motions (synchronized to

50 Hz) causes the agent to fall, since the mapping of the human motion to the robot does not consider physics or the masses and capabilities of the robot. The sequences of joint angles provided by the direct mapping have to be adjusted to obtain a stable motion. This is the main problem that we are addressing in this section. Given some motions as input, we extended our framework to (1) construct models of the motions; (2) initialize model parameters using maximum likelihood and least squares; (3) optimize prior parameters to follow the original motions; and (4) find joint angles that can be replayed by the robot without falling.

3.1 Models and Initialization

A motion consists of a sequence of target angles for each joint. These sequences consist of 50 angles per second for the joint control. Instead of adjusting these angles directly, we create models for the movements of the joints. By approximating the given joint angles with functions, only a relatively small set of parameters has to be optimized to achieve the correct motion on the robot.

We use linear combination of fixed nonlinear functions of the input variable to build the motion model. The input variable, $\mathbf{x} = \{x_1, \ldots, x_M\}$, is the the number of frames in a motion. We use a model of the form $y(\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=0}^{N-1} \theta_j \phi_j(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\Phi}(\mathbf{x})$, where $\phi_j(\mathbf{x})$ are the basis functions with $\phi_0(\mathbf{x}) = 1$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{N-1})^{\mathrm{T}}$, and $\boldsymbol{\Phi} = (\phi_0, \ldots, \phi_{N-1})^{\mathrm{T}}$. There are N total number of parameters in the model. With the choice of suitable basis functions, we model arbitrary nonlinearities in the input traces. Basis functions take many forms, and we use polynomial basis functions of the form $\phi_j(x) = x^j$, and sigmoidal basis functions of the form $\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$, where $\sigma(a)$ is the logistic sigmoidal function defined by $\sigma(a) = \frac{1}{1+\exp(-a)}$, μ_j fixes the location of the basis functions in the input space, and s represents the spacial scale. Polynomial basis functions are global functions of the input, which cause changes in one region to affect all the other regions. On the other hand, sigmoidal basis functions are local, and a small change to input only affect some of the nearby basis functions. The application of global and local basis function can have a significant influence on the optimization. The target variable, $\mathbf{t} = (t_1, \ldots, t_M)^{\mathrm{T}}$, of the motion is the desired angle of a given trace. Each motion contains K traces ($K = 22$ in our experiments), and each trace of the motion is fitted using the linear basis function model. We minimize the objective function $\sum_{i=1}^{M} (t_i - \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{\Phi}(\mathbf{x}))^2$ using ES/PSO algorithms to find the maximum likelihood parameters.

3.2 Model Optimization

The evaluation of the models with the initial parameters provides approximately close enough traces to the original motions. A replay of a motion with the respective model initially fails to capture the desired outcome of the original motion. The joints are moved according to the input motion, however depending on specific robot model (e.g., the masses of body parts) it is necessary to change the motion slightly.

The initial optimization of the model parameters is used as a seed for the optimization of the motion for stability on the robot. This task is an optimization problem with two conflicting objectives. Following exactly the joint angles provided by the MoCap does not guarantee that the outcome is the correct motion. For instance, for a kick motion the robot could fall back and kick into the air, which might follow exactly the given joint angles. This can be avoided by including the captured torso orientation of the human for every time step in the motion capture data. The torso orientation of the simulated robot is provided by the simulator as ground truth. The difference between the captured angles and the robots torso orientation is one component of the fitness function. At the same time the changes in the angles have to be small to make sure that the final result is close to the captured motion, e.g., a kick motion should not be stabilized by removing the actual kick from the motion. The differences between the angles provided by the MoCap and the joint angles of the robot are the second component of the fitness function.

We use ES/PSO algorithms again to optimize the model parameters until the desired motion is learned. In this phase, we optimize $N \times L$, where $L < K$, parameters directly. We use the sum of the torso errors and the joint errors over all frames of the motion as the fitness to perform real valued black box function optimization. We have decided to optimize only the traces of the agent's legs. There are twelve such traces for each motion, and we directly optimize $N \times 12$ parameters. e.g., if we commit to a polynomial model with eight parameters, we optimize 96 parameters.

4 Experimental Setup

The experiments in section 5 have been conducted using SimSpark (based on Spark[12]), the simulator of the RoboCup 3D Soccer Simulation League. The simulated robot is a humanoid robot that is similar to the Nao [4]. The robot is equipped with 21 degrees of freedom, and it receives sensor information every cycle (50 Hz) from the server.

We use four different motions in the experiments in which the robot (1) lifts the right leg for a few seconds (motion `leg`); (2) performs a simple kick motion (motion `kick`); (3) leans forward and balances on one leg while stretching the other leg back (motion `balance`); and (4) leans the torso to the side (motion `side`). The joint motions are modeled using two different functions: polynomials and linear weighted sigmoidal basis functions. These models are initialized by minimizing the least squared error to the input angles. Using the initial parameters as a seed, the twelve leg joints are optimized by CMA-ES, xNES, and PSO using the fitness function based on the joint and the torso error explained in 3.2.

In [16], twelve parameters were learned with a population size of 30; since we optimize up to 96 parameters, we use a higher population size of 50 for all optimization methods. CMA-ES and xNES start with the parameters from the initialization of the models as the mean and a standard deviation of 0.06, which produces a suitable amount of exploration in the beginning of the optimization.

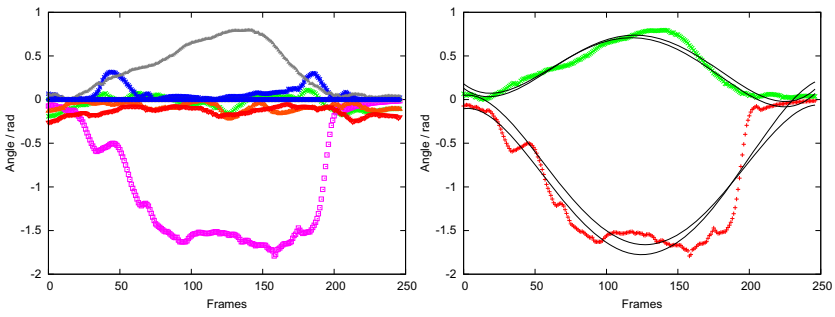
The samples for PSO are initialized using the same mean and standard deviation. Our PSO implementation uses the parameters proposed in [2].

For each evaluation of parameters, the robot is initialized to the pose in the first frame of the motion. While the motion is executed using the current parameters, all torso and joint errors are added; these errors are used as the fitness of the tested parameters. For each parameter set, the motion is executed only once since the environment is simulated and we can expect less noise than with a physical robot.

Learning the transition between different motions is beyond the scope of this paper. The evaluation of a parameter set starts with a short phase during which the robot moves all joints to the angles at frame 0 of the motion. The robot is also moved to an initial torso position using a trainer interface of the simulator. This way, the motion is always started from the same initial situation. At the end of the motion, the robot keeps the joints at the angles from the last frame for half a second. During this time, the torso and joint errors for the evaluation are still accumulated; this prevents learning of motions that are unstable in the end and would make the robot fall immediately after the motion is done. Learning transitions between motions is planned as future work.

5 Experiments and Results

For the model of the first experiment, we used polynomial basis functions with five parameters. Fig. 3a shows the joint angles of the balance motion, and Fig. 3b shows the traces of two of the joints. The optimized motion slightly adjusts the initial values of the parameters to obtain stable motion, and the combination of all joint traces together needs to be stable. However, the experiments show that often only very small changes in the joint motion provide a stable and complete motion. In Fig. 3b the polynomial seems to be able to sufficiently approximate the motion of the joint.



(a) (b)

Fig. 3. (a) The leg joint angles of the balance motion. (b) The motions of two leg joints (l_knee_pitch and r_hip_pitch) during the balance motion with the corresponding models using the initial parameters and the adjusted models of the stabilized motion.

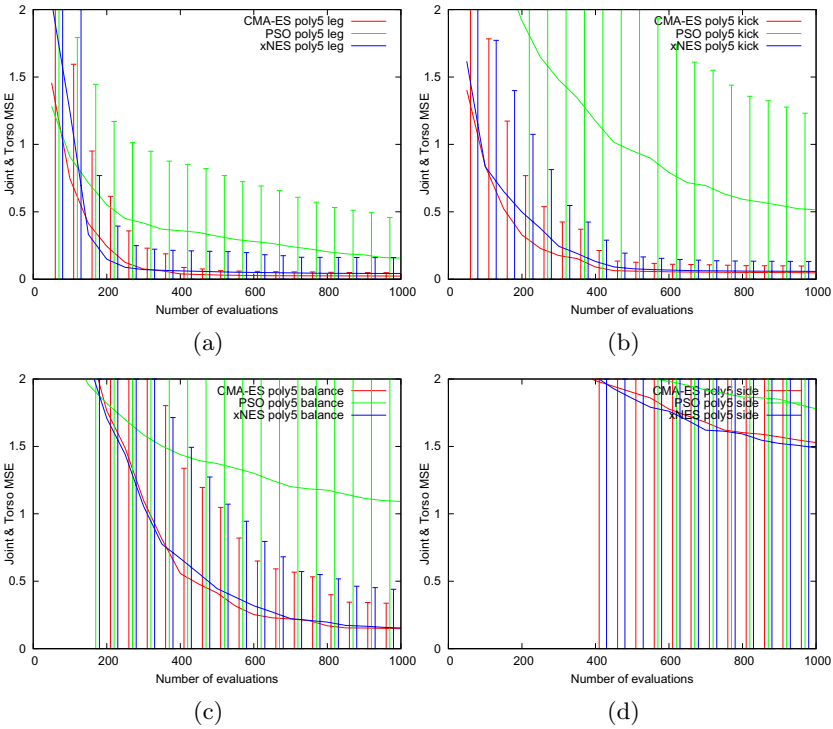


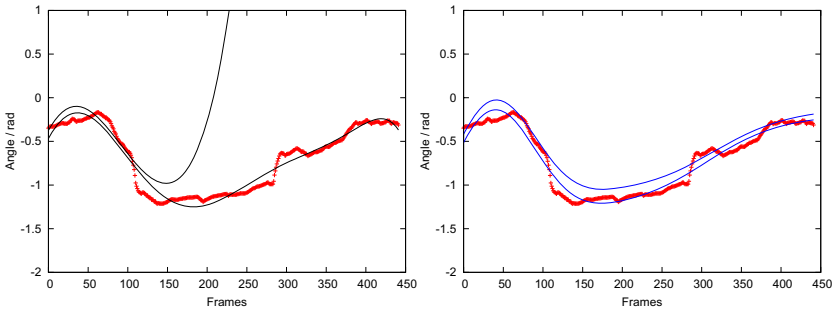
Fig. 4. The learning curves for different motions using polynomials of degree 4. The motions for the twelve leg joints were optimized using CMA-ES, PSO and xNES. The error is the minimum total error (joints MSE + torso angle MSE) averaged over 30 runs. The error bars represent the standard deviation over these 30 runs.

We used the same polynomial approximation to learn four different motions, and Fig. 4 shows the average learning curves. In these experiments, both CMA-ES and xNES quickly learn solutions, with CMA-ES finding a solution with a slightly smaller variance. Both algorithms perform better than PSO. It is possible that the results of PSO could be improved by tuning some internal parameters; CMA-ES and xNES do not require this.

There is a swift learning curve for leg motion. For the kick motion, PSO yields a very high variance, which indicates that the found motion is often unstable. For the balance motion in Fig. 4c, CMA-ES and xNES find good solutions, but the learning time increases. While the polynomials work for these three motions, the algorithms could not find a stable motion for the side balance motion in Fig. 4d. In fact, the results for the other motions are also often unsatisfactory. Polynomials cause these motions to be very smooth. Although the errors are often small and the motion is stable, there is a noticeable lack of detail, and the high-degree polynomials introduce numerical instabilities. In most motions, polynomials cause some joints to move unexpectedly towards the end.

Table 1. The errors after learning for 5 hours simulated time using polynomials with 5 parameters as models (average over 30 runs)

Motion	Optimization	evals	min.err.	avg.err.	stddev	torso err.	joint err.	success
leg	CMA-ES	2989	0.009	0.016	0.004	0.006	0.009	83%
leg	PSO	2987	0.006	0.058	0.154	0.049	0.009	70%
leg	xNES	2990	0.013	0.017	0.003	0.007	0.010	86%
kick	CMA-ES	4926	0.018	0.027	0.005	0.011	0.016	60%
kick	PSO	4924	0.015	0.135	0.210	0.106	0.029	46%
kick	xNES	4931	0.023	0.037	0.047	0.020	0.017	60%
balance	CMA-ES	2970	0.051	0.096	0.076	0.046	0.050	60%
balance	PSO	2971	0.072	0.807	0.403	0.720	0.088	13%
balance	xNES	2971	0.047	0.089	0.153	0.050	0.039	60%
side	CMA-ES	1816	0.432	1.351	0.466	1.203	0.149	0%
side	PSO	1816	0.474	1.634	0.370	1.505	0.129	0%
side	xNES	1816	0.401	1.229	0.490	1.092	0.137	0%



(a) (b)

Fig. 5. An example joint motion (l_knee_pitch) of the side balance and the initial and learned model using CMA-ES and (a) a polynomial with 8 parameters or (b) sigmoidal basis functions with a sum of 8 parameters

Table 1 shows the error values of the experiments after five hours of simulated time. The success rates are created by manually evaluating how many result motions are stable and close enough to the original motion. A longer learning time improves the success rates. However, another reason for lower success rates, despite small average errors, is the noise in the fitness values. There is a chance that an unstable motion gets a small error once and never works again. Averaging the fitness over several runs could lower this noise and improve the learning, but each evaluation would need much more time.

As an attempt to improve the learning, we ran the optimization of the side balance motion again using polynomials, but increased the number of parameters to eight. It still did not find a solution. Increasing the number of degrees only creates more instabilities. Fig. 5a shows that a reason for the unsatisfactory performance is the global influence of parameters on the function. Changing the first part of the motion can create completely wrong angles for the remaining motion.

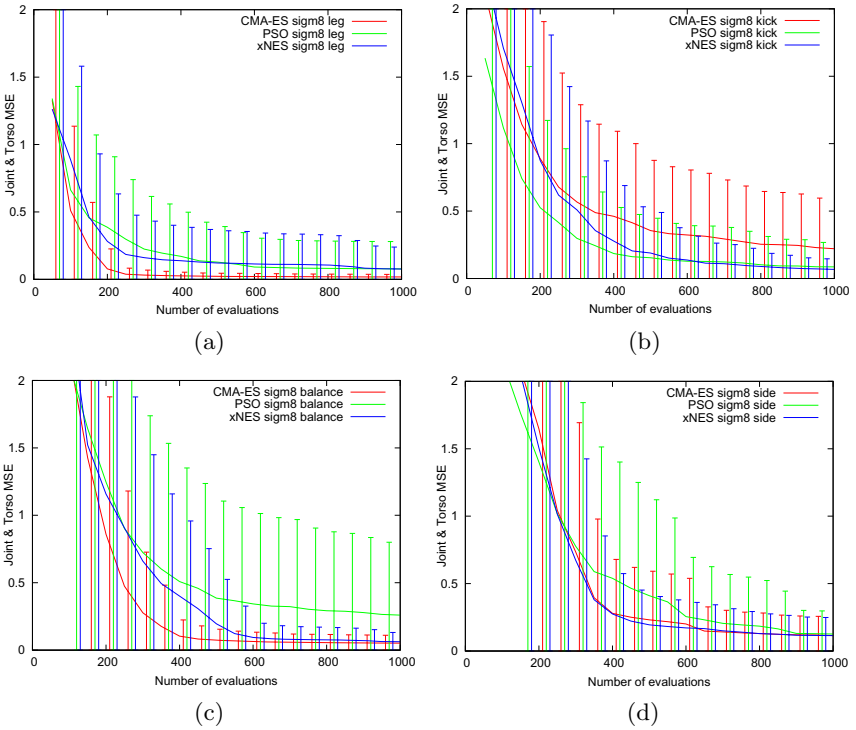


Fig. 6. Using sigmoidal basis functions instead of polynomials improves the results of the learning. The learning curves of the same experiments as in Fig. 4 are shown (averages over 30 runs).

Table 2. Results of the optimization using sigmoidal basis functions

Motion	Optimization	evals	min.err.	avg.err.	stddev	torso err.	joint err.	success
leg	CMA-ES	1781	0.011	0.016	0.004	0.007	0.008	93%
leg	PSO	1782	0.005	0.051	0.149	0.044	0.007	76%
leg	xNES	1782	0.012	0.042	0.086	0.027	0.014	70%
kick	CMA-ES	2935	0.029	0.150	0.206	0.124	0.027	53%
kick	PSO	2935	0.012	0.038	0.068	0.029	0.009	43%
kick	xNES	2940	0.037	0.047	0.007	0.025	0.023	40%
balance	CMA-ES	2970	0.029	0.042	0.016	0.018	0.024	93%
balance	PSO	2971	0.027	0.119	0.256	0.092	0.027	73%
balance	xNES	2971	0.030	0.041	0.009	0.020	0.020	76%
side	CMA-ES	1816	0.041	0.099	0.060	0.073	0.026	70%
side	PSO	1816	0.024	0.093	0.085	0.064	0.029	60%
side	xNES	1817	0.038	0.102	0.062	0.078	0.024	40%

Since the side motion could not be learned at all and the other motions suffered from the high generalization and numerical instabilities of the polynomials, we ran the same experiments again with the linear weighted

sigmoidal basis functions as the model. Fig. 5b shows that the local basis functions improve the optimization. All four motions can be adjusted to be stable on the robot using this model (Fig. 6). Although the number of parameters that are optimized has been increased from 60 to 96 (twelve joints, five or eight parameters per model), the optimization needs less time to find solutions for the balance motion and also works for the side balance. The average joint and torso errors are listed in Table 2. For the balance motion the improved models yield significantly smaller joint angle errors compared to the polynomials.

6 Conclusions and Future Work

Our hypothesis that robust and human-like motions for a biped soccer robot can be quickly generated using inexpensive motion capture techniques with optimization is verified. The Kinect sensor provides enough information to map complex and initially unstable poses, such as the balancing motions. Applying modern optimization algorithms to the mapped motions will lead to robust and stable motions on the robot; that said, we do not make the claim that our approach will hold for *all* possible motions. Nevertheless, our results are very promising, and it shows that we have found a process that can be used to produce a number of motions needed for humanoid soccer robots. We ran our parameter optimization on a desktop CPU with four cores and 2.27 GHz. After running experiments with four motions, three algorithms, two model functions with different number of parameters and 30 runs each, we can say that the motions were mostly stable after 30 min (in average). Making use of parallel processing on a cluster of computers can bring the entire processing time down to 10-30 min per motion, including the motion capture.

We have observed that CMA-ES and xNES are similar in performance. Also, the PSO algorithm shows partly smaller errors with a larger variance and it takes longer to learn. However, a smaller error does not necessarily mean a better motion. Sometimes parameters that created a small error once produce motions that result in a falling robot. The influence of running a motion multiple times for each evaluation and averaging the error needs to be evaluated in the future. Nevertheless, the optimization using models with sigmoidal basis functions yields good results for all three algorithms. Optimizing only the leg motions is sufficient for the used robot model. Further experiments have shown that including the arms yields a higher error and variance without finding better solutions.

On the motion capture end, we would like to compare our motions mapped using the Kinect with motions captured from a Vicon optical marker system. Some motions are particularly tricky to perform on the Kinect, such as standing after a fall or kicking, as the skeletal tracking algorithms can be confused when the user rotates or has an body part that is at least partially occluded. Our motion capture pipeline is written in a modular fashion to accommodate changes to the input sensor, the mapping mechanism, and the robot model; we believe this framework can be used to experiment with at least a few other robot models, both simulated and physical, and provide a convenient platform for quickly creating motions on heterogeneous humanoid robots.

References

1. Amor, H., Berger, E., Vogt, D., Jung, B.: Towards Responsive Humanoids: Learning Interaction Models for Humanoid Robots. In: International Conference on Machine Learning, pp. 1–4 (2011)
2. Carlisle, A., Dozier, G.: An off-the-shelf PSO. In: Proceedings of the Workshop on Particle Swarm Optimization, pp. 1–6. Purdue School of Engineering and Technology, IUPUI, Indianapolis (2001)
3. Glasmachers, T., Schaul, T., Schmidhuber, J.: A natural evolution strategy for multi-objective optimization. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 627–636. Springer, Heidelberg (2010)
4. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: The nao humanoid: a combination of performance and affordability. CoRR abs/0807.3223 (2008)
5. Grimes, D.B., Rashid, D.R., Rao, R.P.N.: Learning nonparametric models for probabilistic imitation. In: Advances in Neural Information Processing Systems (NIPS). MIT Press (2006)
6. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
7. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks 1995, vol. 4, pp. 1942–1948. IEEE (1995)
8. Kern, S., Müller, S., Hansen, N., Büche, D., Ocenasek, J., Koumoutsakos, P.: Learning probability distributions in continuous evolutionary algorithms–a comparative review. Natural Computing 3(1), 77–112 (2004)
9. Kim, S., Kim, C., You, B., Oh, S.: Stable whole-body motion generation for humanoid robots to imitate human motions. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, pp. 2518–2524. IEEE (2009)
10. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012) (June 2012)
11. Niehaus, C., Röfer, T., Laue, T.: Gait Optimization on a Humanoid Robot using Particle Swarm Optimization (2007)
12. Obst, O., Rollmann, M., Rollmann, M.: Spark - a generic simulator for physical multi-agent simulations. In: Computer Systems Science and Engineering (2004)
13. OpenNI organization: OpenNI User Guide (November 2010), `http://www.openni.org/documentation` (last viewed February 26, 2012)
14. PrimeSense Inc.: Prime Sensor NITE 1.3 Algorithms notes (2010), `http://www.primesense.com` (last viewed February 26, 2012)
15. Setapen, A., Quinlan, M., Stone, P.: Marionet: Motion acquisition for robots through iterative online evaluative training (extended abstract). In: The Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS). International Foundation for Autonomous Agents and Multiagent Systems (May 2010)
16. Urieli, D., MacAlpine, P., Kalyanakrishnan, S., Bentor, Y., Stone, P.: On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In: Tumer, K., Yolum, P., Sonenberg, L., Stone, P. (eds.) Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011), vol. 2, pp. 769–776. IFAAMAS (May 2011)
17. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: Proceedings of the Congress on Evolutionary Computation (CEC 2008), Hongkong. IEEE Press (2008)