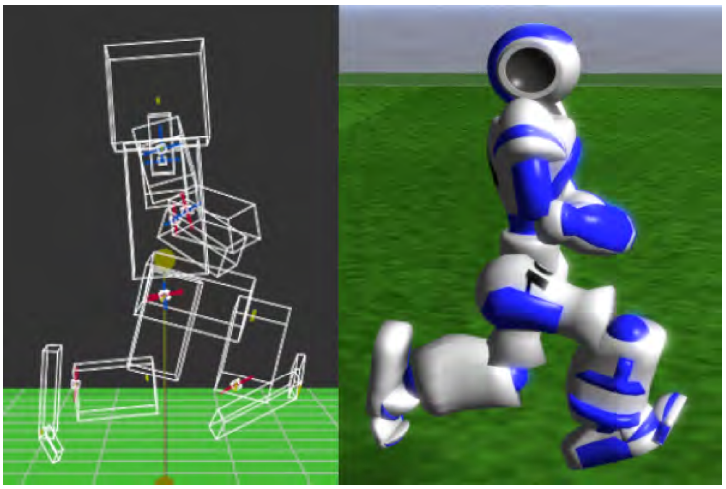
Learning to Walk With Toes

Jens Fischer and Klaus Dorer

Offenburg University of Applied Sciences
{jens.fischer, klaus.dorer}@hs-offenburg.de

Abstract. This paper explains how a model-free (with respect to the robot model and the behavior to learn) approach can facilitate learning to walk from scratch. It is applied to a simulated Nao robot with toes. Results show an improvement of 30% in speed compared to a model without toes and also compared to our model-based approach, but with less stability.

Keywords: machine learning, genetic algorithms, humanoid robot walking



1 Introduction

Utilizing toes of a humanoid robot is difficult for various reasons, one of which is that inverse kinematics is overdetermined with the introduction of toe joints. Nevertheless, a number of robots with either passive toe joints like the Monroe or HRP-2 robots [1,2] or active toe joints like Lola, the Toyota robot or Toni [3,4,5] have been developed. Recent work shows considerable progress on learning model-free behaviors using genetic learning [6] for kicking with toes and deep reinforcement learning [7,8,9] for walking without toe joints. In this work, we show that toe joints can significantly improve the walking behavior of a simulated Nao robot and can be learned model-free.

The remainder of this paper is organized as follows: Section 2 gives an overview of the domain in which learning took place. Section 3 explains the approach for model free learning with toes. Section 4 contains empirical results for various behaviors trained before we conclude in Section 5.

2 Domain

The robots used in this work are robots of the RoboCup 3D soccer simulation which is based on SimSpark[1] and initially initiated by [10]. It uses the ODE physics engine[2] and runs at an update speed of 50Hz. The simulator provides variations of Aldebaran Nao robots with 22 DoF for the robot types without toes and 24 DoF for the type with toes, NaoToe henceforth. More specifically, the robot has 6 (7) DoF in each leg, 4 in each arm and 2 in its neck. There are several simplifications in the simulation compared to the real Nao:

- all motors of the simulated Nao are of equal strength whereas the real Nao has weaker motors in the arms and different gears in the leg pitch motors.
- joints do not experience extensive backlash
- rotation axes of the hip yaw part of the hip are identical in both robots, but the simulated robot can move hip yaw for each leg independently, whereas for the real Nao, left and right hip yaw are coupled
- the simulated Naos do not have hands
- the touch model of the ground is softer and therefore more forgiving to stronger ground touches in the simulation
- energy consumption and heat is not simulated
- masses are assumed to be point masses in the center of each body part

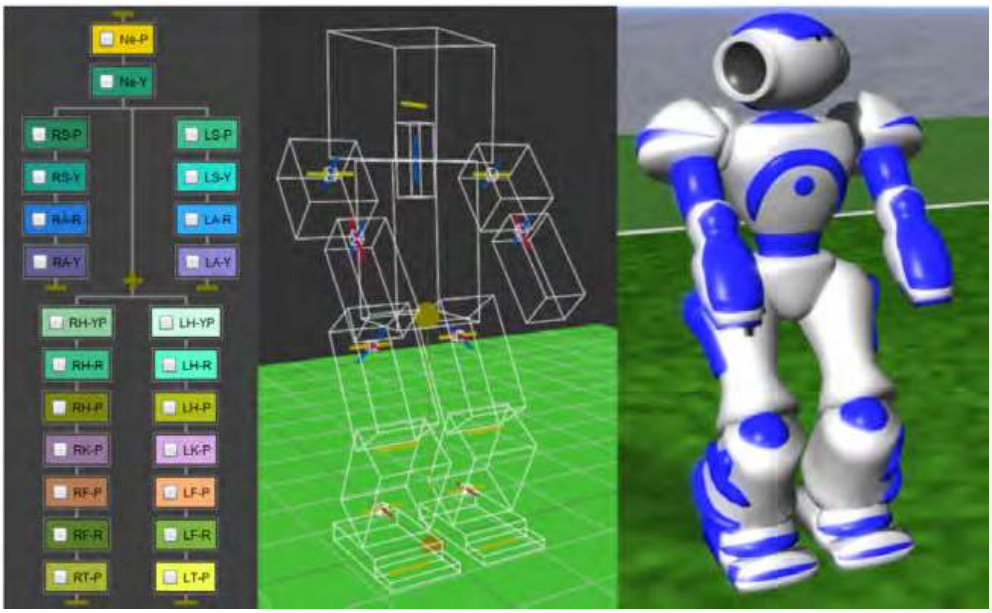


Fig. 1: Joint model (left), wire model (center) and rendered model (right) of the NaoToe robot.

The feet of NaoToe are modeled as rectangular body parts of size 8cm x 12cm x 2cm for the foot and 8cm x 4cm x 1cm for the toes (see Figure 1). The two body parts are connected with a hinge joint that can move from -1 degrees (downward) to 70 degrees.

[1] http://simspark.sourceforge.net/
[2] http://www.ode.org/

All joints can move at an angular speed of at most 7.02 degrees per 20ms. The simulation server expects to get the desired speed at 50 Hz for each joint. If no speeds are sent to the server it will continue movement of the joint with the last speed received. Joint angles are noiselessly perceived at 50Hz, but with a delay of 40ms compared to sent actions. So only after two cycles, the robot knows the result of a triggered action. A controller provided for each joint inside the server tries to achieve the requested speed, but is subject to maximum torque, maximum angular speed and maximum joint angles.

The simulator is able to run 22 simulated Naos in real-time on reasonable CPUs. It is used as competition platform for the RoboCup 3D soccer simulation league[3]. In this context, only a single agent was running in the simulator.

3 Approach

The following subsections describe how we approached the learning problem. This includes a description of the design of the behavior parameters used, what the fitness functions for the genetic algorithm look like, which hyperparameters were used and how the fitness calculation in the SimSpark simulation environment works exactly.

3.1 Behavior Parameters

The guiding goal behind our approach is to learn a model-free walk behavior. With model-free we depict an approach that does not make any assumptions about a robot's architecture nor the task to be performed. Thus, from the viewpoint of learning, our model consists of a set of flat parameters. These parameters are later grounded inside the domain.

The server requires 50 values per second for each joint. To reduce the search space, we make use of the fact that output values of a joint over time are not independent. Therefore, we learn *keyframes*, i.e. all joint angles for discrete phases of movement together with the duration of the phase from keyframe to keyframe. The experiments described in this paper used four to eight of such phases. The number of phases is variable between learning runs, but not subject to learning for now, except for skipping phases by learning a zero duration for it.

The RoboCup server requires robots to send the actual angular speed of each joint as a command. When only leg joints are included, this would require to learn 15 parameters per phase (14 joints + 1 for the duration of the phase), resulting in 60, 90 and 120 parameters for the 4, 6, 8 phases worked with. The disadvantage of this approach is that the speed during a particular phase is constant, thus making it unable to adapt to discrepancies between the desired and the actual motor movement.

Therefore, a combination of angular value and the maximum amount of angular speed each joint should have is used. The direction and final value of movement is entirely encoded in the angular values, but the speed can be controlled separately. It follows that:

- If the amount of angular speed does not allow reaching the angular value, the joint behaves like in version 1.
- If the amount of angular speed is bigger, the joint stops to move even if the phase is not over.

[3] http://www.robocup.org/robocup-soccer/simulation/

This almost doubles the amount of parameters to learn, but the co-domain of values for the speed values is half the size, since here we only require an absolute amount of angular speed. With these parameters, the robot learns a single step and mirrors the movement to get a double step.

3.2 Fitness Function

Feedback from the domain is provided by a fitness function that defines the utility of a robot. The fitness function subtracts a penalty for falling from the walked distance in X-direction in meters. There is also a penalty for the maximum deviation in Y-direction reached during an episode, weighted by a constant factor. In practice, the values chosen for $fallenPenalty$ and a factor f were usually 3 and 2 respectively.

$$fitness_{walk} = distanceX - fallenPenalty - (f * maxY) \tag{1}$$

This same fitness function can be used without modification for forward, backward and sideward walk learning, simply by adjusting the initial orientation of the agent. The also trained turn behavior requires a different fitness function.

$$fitness_{turn} = (g * totalTurn) - distance \tag{2}$$

Where $totalTurn$ refers to the cumulative rotation performed in degrees, weighted by a constant factor g (typically 1/100). We penalize any deviation from the initial starting X / Y position (*distance*) as incentive to turn in-place. It is noteworthy that other than swapping out the fitness function and a few more minor adjustments mentioned in 3.3, everything else about the learning setup remained the same thanks to the model-free approach.

3.3 Fitness Calculation

Naturally, the fitness calculation for an individual requires connecting an agent to the SimSpark simulation server and having it execute the behavior defined by the learned parameters. In detail, this works as follows:

At the start of each "episode", the agent starts walking with the old model-based walk engine at full speed. Once 80 simulation cycles (roughly 1.5 seconds) have elapsed, the robot starts checking the foot force sensors. As soon as the left foot touches the ground, it switches to the learned behavior. This ensures that the learned walk has comparable starting conditions each time. If this does not occur within 70 cycles (which sometimes happens due to non-determinism in the domain and noise in the foot force perception), the robot switches anyway.

From that point on, the robot keeps performing the learned behavior that represents a single step, alternating between the original learned parameters and a mirrored version (right step and left step). An episode ends once the agents has fallen or 8 seconds have elapsed.

To train different walk directions (forward, backward, sideward), the initial orientation of the player is simply changed accordingly. In addition, the robot uses a different walk direction of the model-based walk engine for the initial steps that are not subject to learning.

In case of training a morphing behavior (see 4.5), the episode duration is extended to 12 seconds. When a morphing behavior should be trained, the step behavior from another learning run is used. This also means that a morphing behavior is always trained for a

specific set of walk parameters. After 6 seconds, the morphing behavior is triggered once the foot force sensors detect that the left foot has just touched the ground. Unlike the step / walk behavior, this behavior is just executed once and not mirrored or repeated. Then the robot switches back to walking at full speed with the model-based walk engine. To maximize the reward, the agent has to learn a morphing behavior that enables the transition between learned model-free and old model-based walk to work as reliably as possible.

Finally, for the turn behavior, the robot keeps repeating the learned behavior without alternating with a mirrored version. In any case, if the robot falls, a training run is over.

Experiments were run over 200 generations with 10 oversampling runs per robot to average out non-determinism. Combined with the population size of 200, this means that 400000 fitness calculations are done per learning run:

$$200\ \text{(Population Size)} * 200\ \text{(Generations)} * 10\ \text{(Oversampling)} = 400000 \tag{3}$$

The overall runtime of each such learning run is 2.5 days on our hardware.

3.4 Hyperparameters

Learning is done using plain genetic algorithms. The following hyperparameters were used:

- Population Size: **200 Individuals**
- Genders: **2**
- Parents per Individual: **2**
- Selection Strategy: **MonteCarloSelection**
- Reproduction Strategy: **MultiCrossoverRecombination**
- Mutation Strategy: **RandomMutation**
- Mutation Probability per Individual: **10%**
- Mutation Probability per Gene: **20%**

More details on the approach can be found in [11].

4 Results

This section presents the results for each kind of behavior trained. This includes three different walk directions, a turn behavior and a behavior for morphing.

4.1 Forward Walk

The main focus of this work has been on training a forward walk movement. Figure 2 shows a sequence of images for a learned step. The best result reaches a speed of 1.3 m/s compared to the 1.0 m/s of our model-based walk and 0.96 m/s for a walk behavior learned on the Nao robot without toes. The learned walk with toes is less stable, however, and shows a fall rate of 30% compared to 2% of the model-based walk.

Regarding the characteristics of this walk, it utilizes remarkably long steps[4]. Table 1 shows an in-depth comparison of various properties, including step duration, length and height, which are all considerably bigger compared to our previous model-based walk. The forward leaning of the agent has increased by 80.4%, while 28.1% more time is spent with both legs off the ground. However, the maximum deviation from the intended path ($maxY$) has also increased by 137.8%.

[4] `https://youtu.be/ytM61yTcJ-Q`

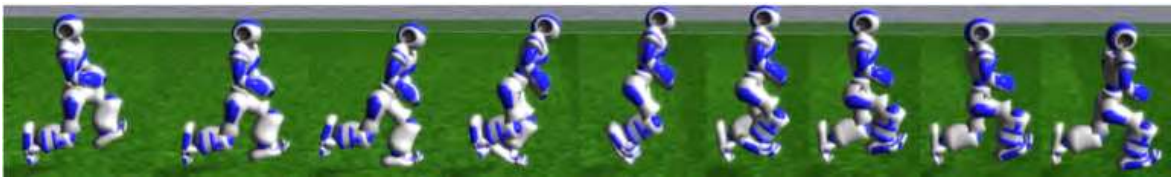


Fig. 2: Movement sequence of a learned walk behavior with toes.

Value	Old	New	Diff	Diff %
utility	5.102	5.118	+0.016	+0.3%
distanceX	6.095	7.482	+1.387	+22.8%
speedX	0.993	1.291	+0.298	+30%
maxY	0.497	1.182	+0.685	+137.8%
bothLegsOffGround	0.495	0.634	+0.139	+28.1%
oneLegOffGround	0.507	0.366	-0.141	-27.8%
noLegsOffGround	0	0.003	+0.003	
stepDuration	0.08	0.301	+0.221	+276.3%
stepLength	0.13	0.429	+0.299	+230%
stepHeight	0.02	0.123	+0.103	+515%
leaningX	0	0.001	+0.001	
leaningY	-0.107	-0.193	-0.086	-80.4%

Table 1: Comparison of the previously fastest and the fastest learned forward walk

4.2 Backward Walk

Once a working forward walk was achieved, it was natural to try to train a backward walk behavior as well, since this only requires a minor modification in the learning environment (changing the initial rotation of the agent and model-based walk direction to start with). The best backward walk learned reaches a speed of 1.03 m/s, which is significantly faster than the 0.67 m/s of its model-based counterpart. Unfortunately, the agent also falls 15% more frequently.

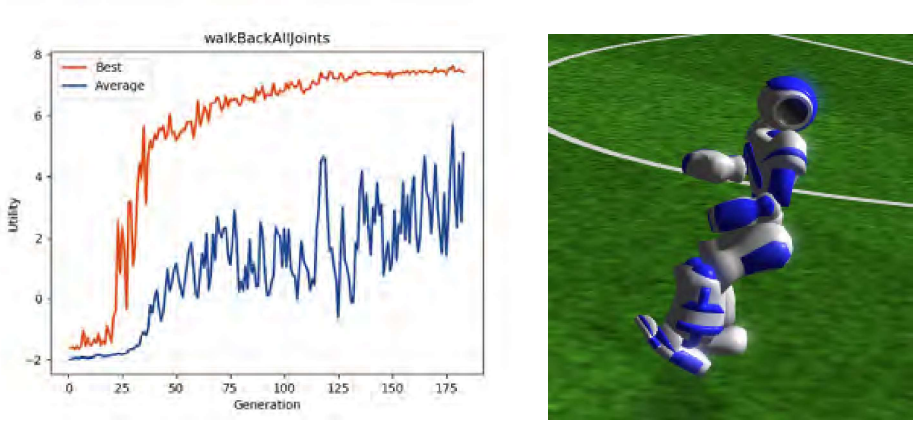
It is interesting just how backward-leaning the agent is during this walk behavior. It could almost be described as "controlled falling"[5] (see Figure 3).

4.3 Sideward Walk

Sideward walk learning was the least successful out of the three walk directions. Like with all directions, the agent starts out using the old walk engine and then switches to the learned behavior after a short time. In this case however, instead of continuing to walk sideward, the agent has learned to turn around and walk forward instead, see Figure 4.

The resulting forward walk is not very fast and usually causes the agent to fall within a few meters[6], but it is still remarkable that the learned behavior manages to both turn the agent around *and* make it walk forward with the same repeating step movement. It is also remarkable that the robot learned that it is quicker with the given legs at least for long distances to turn and run forward than to keep making sidesteps.

[5] `https://youtu.be/4mq06V_Sk9Y`
[6] `https://youtu.be/4WnPzM1pPfU`



(a) Learning curve (b) Heavy backward lean

Fig. 3: Learning curve and backward lean of the backward walk

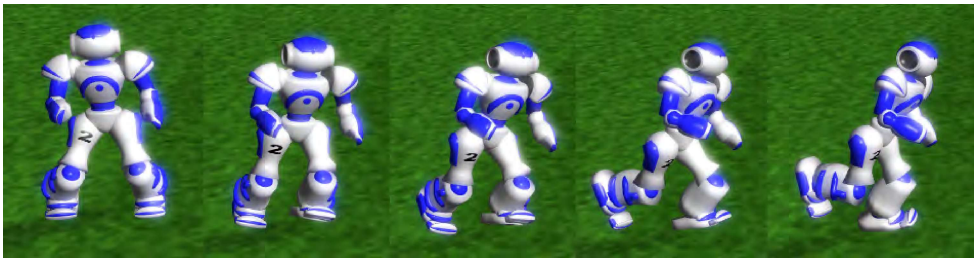


Fig. 4: Change of the walk direction during sideward walk learning

4.4 Turn

With the alternate fitness function presented in section 3, the agent managed to learn a turn behavior that is comparable in speed to that of the existing walk engine. Despite this, the approach is actually different: while the old walk engine uses small, angled steps[7], the learned behavior uses the left leg as a "pivot", creating angular momentum with the right leg[8]. Figure 5 shows the movement sequence in detail.

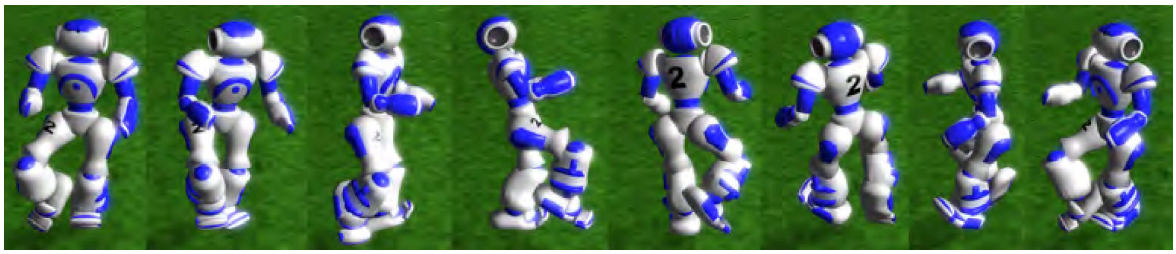


Fig. 5: Movement sequence of a learned turn behavior.

Unfortunately, despite the comparable speed, the learned turn behavior suffers from much worse stability. With the old turn behavior, the agent only falls in roughly 3% of cases, with the learned behavior it falls in roughly 55% of the attempts.

[7] `https://youtu.be/PryjLMXIta0`
[8] `https://youtu.be/xgyZtCS68Wo`

4.5 Morphing

One of the major hurdles for using the learned walk behaviors in a RoboCup competition is the smooth transition between them and other existing behaviors such as kicks. The initial transition to the learned walk is already built into the learning setup described in 3 by switching mid-walk, so it does not have to be given special consideration. More problematic is switching to another behavior afterwards without falling.

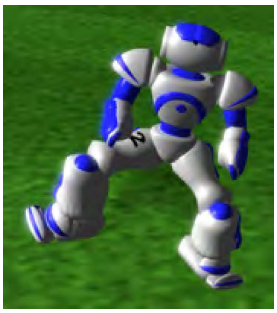


Fig. 6: Lunge performed by the trained morphing behavior.

To handle this, the robot simply attempted to train a "morphing" behavior using the same model-free learning setup. The result is something that could be described as a "lunge" (see Figure 6) that reduces the forward momentum sufficiently to allow it to transition to the slower model-based walk when successful.[9] However, the morphing is not successful in about 50% of cases, resulting in a fall.

5 Conclusion and Future Work

We were able to successfully train forward and backward walk behaviors, as well as a morphing and turn behavior using plain genetic algorithms and a very flexible model-free approach. The usage of the toe joint in particular makes the walks look more natural and human-like than that of the model-based walk engine.

However, while the learned behaviors outperform or at least match our old model-based walk engine in terms of speed, they are not stable enough to be used during actual RoboCup 3D Simulation League competitions. We think this is an inherent limitation of the approach: We train a static behavior that is unable to adapt to changing circumstances in the environment, which is common in SimSpark's non-deterministic simulation with perception noise.

Deep Reinforcement Learning seems more promising in this regard, as the neural network can dynamically react to the environment since sensor data serves as input. It is also arguably even less restrictive than the keyframe-based behavior parameterization we presented in this paper, as a neural network can output raw joint actions each simulation cycle. At least two other RoboCup 3D Simulation League teams, FC Portugal [8] and ITAndroids [9], have had great success with this approach, Everything points towards this becoming the *state-of-the-art* approach in RoboCup 3D Soccer Simulation in the near future, so we want to concentrate our future efforts here as well.

[9] `https://youtu.be/NB_4i2_b-Pg`

References

1. Ogura, Y., Shimomura, K., Kondo, H., Morishima, A., Okubo, T., Momoki, S., Lim, H., Takanishi, A.: Human-like Walking with Knee Stretched, Heel-contact and Toe- off Motion by a Humanoid Robot. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots Systems (IROS) (2006) 3976–81
2. Sellaouti, R., Stasse, O., Kajita, S., Yokoi, K., Kheddar, A.: Faster and Smoother Walking of Humanoid HRP-2 with Passive Toe Joints. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots Systems (IROS) (2006)
3. Buschmann, T., Lohmeier, S., Ulbrich, H.: Humanoid robot lola: Design and walking control. Journal of physiology Vol 103, Paris (2009) 141–148
4. Tajima, R., Honda, D., Suga, K.: Fast running experiments involving a humanoid robot. (05 2009) 1571–1576
5. Behnke, S.: Human-like walking using toes joint and straight stance leg. Proceedings of 3rd International Symposium on Adaptive Motion in Animals and Machines (AMAM) (11 2005)
6. Dorer, K.: Learning to use toes in a humanoid robot. In Akiyama, H., Obst, O., Sammut, C., Tonidandel, F., eds.: RoboCup 2017: Robot World Cup XXI, Springer (1 2018) 168–179
7. Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S.M.A., Riedmiller, M.A., Silver, D.: Emergence of locomotion behaviours in rich environments. arXiv:1707.02286 (2017)
8. Abreu, M., Lau, N., Sousa, A., Reis, L.P.: Learning to run faster in a humanoid robot soccer environment through reinforcement learning. RoboCup 2019: Robot World Cup XXIII (2019)
9. Melo, L.C., Maximo, M.R.: Learning humanoid robot running skills through proximal policy optimization. arXiv preprint arXiv:1910.10620 (2019)
10. Obst, O., Rollmann, M.: Spark - a generic simulator for physical multi-agent simulations. Volume 20. (07 2004)
11. Fischer, J.: Laufenlernen eines humanoiden Roboters mit Zehen mit Hilfe von Machine Learning und Behavior Morphing. Master thesis, Offenburg (2019)