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

Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
{afarchy, sbarrett, patmac, pstone}@cs.utexas.edu

## ABSTRACT

Simulation is often used in research and industry as a low cost, high efficiency alternative to real model testing. Simulation has also been used to develop and test powerful learning algorithms. However, parameters learned in simulation often do not translate directly to the application, especially because heavy optimization in simulation has been observed to exploit the inevitable simulator simplifications, thus creating a gap between simulation and application that reduces the utility of learning in simulation.

This paper introduces Grounded Simulation Learning (GSL), an iterative optimization framework for speeding up robot learning using an imperfect simulator. In GSL, a behavior is developed on a robot and then repeatedly: 1) the behavior is optimized in simulation; 2) the resulting behavior is tested on the real robot and compared to the expected results from simulation, and 3) the simulator is modified, using a machine-learning approach to come closer in line with reality. This approach is fully implemented and validated on the task of learning to walk using an Aldebaran Nao humanoid robot. Starting from a set of stable, hand-coded walk parameters, four iterations of this three-step optimization loop led to more than a 25% increase in the robot's walking speed.

## Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

## General Terms

Performance, Design, Experimentation

## Keywords

Simulation Machine Learning, Bipedal Robotics

## 1. INTRODUCTION

The fields of robotics and autonomous agents are both progressing to the point where it is becoming plausible that before long, robot agents will be commonplace in our everyday environments. However, the large variety of real-world environments will place a large burden on the ability of people to generate appropriate skills and behaviors for these robots.

Machine learning is an enticing alternative to manual generation of these skills and behaviors, and indeed there have been some notable successes of such [4, 15, 20]. However, most learning algorithms require a large number of trials to be effective on most problems. This fact, coupled with the fragility of most robots and the time and effort required to gather data on robots, makes learning on physical robots infeasible except in the rare cases in which the behaviors can be learned in relatively few trials.

When faced with the challenges of learning in the real world, a natural reaction is to instead learn in simulation. Simulators facilitate running large numbers of trials without any manual intervention and do not risk breaking any robots. However, behaviors learned in simulation are notoriously difficult to transfer to the real world. Learning algorithms have a tendency to overfit to the inevitable inaccuracies and approximations that are inherent in even the best simulators.

This paper is motivated by a robot learning challenge, namely enabling a humanoid robot, the Aldebaran Nao, to walk quickly. This robot is used as the substrate for the RoboCup Standard Platform League, in which teams of four robots from different institutions compete against one another in a physical game of soccer. The same robot is the basis for the physics-based RoboCup 3D-simulator, the substrate for the 3D simulation league in which teams of eleven robots compete in a simulated game of soccer. Although there has been at least one example of a robot walk developed on the physical robot being useful as a starting point for learning in the simulator [17], we are not aware of any optimized walks from the simulator being successfully transferred back to the real world.

This lack of successful transfer from the simulator to the real robot is unsurprising, due to the simulated model of the robot being imperfect, and the simulation environment poorly representing a physical robot soccer field. More importantly, the physics engine in the simulator, while considered quite good, simply does not produce the same results as the real world.

We therefore introduce Grounded Simulation Learning (GSL), a novel, general, iterative optimization paradigm for learning skills in simulation for a physical robot. Our goal is to use parameters optimized in simulation to improve the robot's task performance in the real world. Each iteration in GSL begins by grounding the simulation's movements to

**Appears in:** *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

the real world via a machine learning approach. Then, the parameters are optimized on the grounded simulation, and the top parameter sets from this optimization are evaluated on the real robot. Finally, an expert guides the optimization by selecting parameter sets to investigate further, preferring ones that perform well on both the simulation and the real robot, and modifies the optimization algorithm to focus on promising parameters in the next iteration of GSL. In our testbed domain, starting from a set of stable, hand-coded walk parameters, four iterations of GSL led to more than a 25% increase in the robot's walking speed.

The remainder of this paper is organized as follows. Section 2 describes our general methodology. Section 3 provides background information about our test domain, and implementation details of GSL are discussed and evaluated in Section 4. Section 5 provides related works, and Section 6 describes unsuccessful strategies we attempted which are worth mention. Finally, Section 7 concludes.

# 2. METHODOLOGY

This section describes the problem setting and introduces the novel Grounded Simulation Learning algorithm.

## 2.1 Objective and Assumptions

This paper focuses on optimizing a physical robot's behavior that is influenced by a set of parameters. For example, this paper considers a parameterized humanoid walk engine described in Section 3.1.

We assume the following:

1. There is an imperfect simulation of the robot that permits evaluation of the parameterized behavior via **$\mathsf{Fitness}_{\mathsf{sim}}$**. In addition, this function must be modifiable in order to make the simulation better match the real robot's behavior. If the simulation itself is not easily modifiable, this modification can be in the form of changes to the chosen actions of the simulated robot.

2. A small number of evaluations can be run on the real robot via **$\mathsf{Fitness}_{\mathsf{robot}}$**, which evaluates the fitness of the parameterized behavior.

3. A small number of explorations can be run on the real robot, via an **$\mathsf{Explore}_{\mathsf{robot}}$** routine, to collect states and actions relevant to the current parameterization of the behavior.

4. There is a supervised learning algorithm, **Learn**, that can be used to learn a model of the effects of actions on state of the real robot. This model will be used to modify $\mathsf{Fitness}_{\mathsf{sim}}$ to make it better reflect the behavior on the real robot.

5. There is an optimization algorithm, **Optimize**, that can be used to find better parameters in simulation.

To clarify, the number of evaluations and explorations required on the real robot may be orders of magnitude fewer than the number of evaluations in simulation. In our implemenation, for each evaluation on the robot, 150 evaluations were performed in simulation. We believe that these assumptions are broadly realizable across a large variety of real robot tasks.

## 2.2 Grounded Simulation Learning

The Grounded Simulation Learning algorithm (GSL) is designed to tackle the problem described in the preceding section. It follows two main principles: *grounding* and *guidance*. *Grounding* refers to trying to make the simulation of the behavior match the real robot's behavior in relation to the parameters of interest. To this end, we try to limit any differences between the simulator and the real robot which result in good performance in simulation, but poor performance on the robot. With adequate grounding, we may expect the simulation to perform similarly enough to the robot for the optimization to work. However, in the large parameter spaces involved in learning robot tasks, this may be insufficient. Therefore, it is helpful to have an expert *guide* the optimization algorithm to focus on specific parameters.

The full GSL algorithm is described in Algorithm 1. It takes an initial parameter set $P_0$, as well as the five functions described in Section 2.1, and produces an approximately optimal set of parameters for performing the behavior on the robot.

**Algorithm 1:** Grounded Simulation Learning

**Input**:
$P_0$ – Initial parameter set
$\mathsf{Fitness}_{\mathsf{sim}}$ – a simulation fitness function that uses a model that maps joint commands to outputs
$\mathsf{Fitness}_{\mathsf{robot}}$ – a robot fitness function
$\mathsf{Explore}_{\mathsf{robot}}$ – a robot exploration routine
Learn – a supervised learning algorithm
Optimize – an optimization algorithm to run in simulation
**Output**: $P_{\text{opt}}$– Optimized parameter set

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

On lines 5–6, GSL learns a new model of the effects of actions on the robot which is explained in more detail in Section 4.3. Then, on line 7, it runs the optimization algorithm in simulation (discussed in Section 4.2) to find new parameters to investigate. Next, lines 8–14 show how GSL updates the OpenParams and tracks the best parameters. The idea of the OpenParams is to keep track of distinct parameters that perform well on both the robot and simulation. It can be thought of as tracking tree nodes whose children should be visited. Finally, in line 15, a human is put into the loop as a guide in order to speed up the optimization by focusing the optimization on specific parameters to investigate in greater depth in the next iteration. More information about guidance is given in Section 4.4. This loop repeats until the resulting parameters do not improve the robot's fitness.

# 3. TESTBED DOMAIN

In order to validate GSL, we implemented and tested it on a concrete, challenging robot task, namely fast biped locomotion on an Aldebaran Nao robot, shown in Figure 1. The Nao is a humanoid robot that stands a little more than half a meter tall. It has twenty-five degrees of freedom, eleven of which are in the pelvis and legs. In addition, the Nao has proprioception of all joints, pressure sensors on its feet, two gyrometers, and an accelerometer.

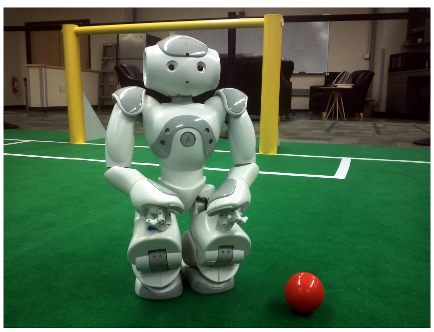

Figure 1: Aldebaran Nao.

Section 3.1 describes the parameterized walk engine that this work optimizes, and Section 3.2 specifies the simulator used for this work.

## 3.1 Walk Engine

A walk engine converts a requested walking velocity to a set of desired joint angles that are sent to the joint motors at each time step. While the Nao robots come equipped with Aldebaran's closed-source walk engine, research from several RoboCup teams have shown that significantly faster walks are possible on the Naos [7, 27, 30]. Therefore, we start from one of the existing walk engines designed by a RoboCup team. Specifically, the walk engine used in this work is based on the walk by Nao-Team HTWK from Leipzig University of Applied Sciences [30], which was in turn inspired by Behnke [5]. Due to space constraints, we limit our attention in this paper to the optimized parameters. Further details on the walk engine are available in [5].

Specifically, 17 parameters were optimized, a list of which can be found in Table 1. A phase of the walk is the time it takes the robot to take two steps (one with the left foot and one with the right), and it is denoted as the *stepPeriod* parameter. A step consists of three components. The first component is shifting the center of mass onto the stance leg. Then, the back leg is lifted by bending according to the *knee*, $v_{\text{short}}$, $\phi_{\text{short}}$, and $a_{\text{short}}$ parameters. Finally, the lifted leg is swung forward according to the $amp_{\text{swing}}$, $v_{\text{swing}}$, and $\phi_{\text{swing}}$ parameters. The *fwdOffset* parameter is applied to prevent the robot from drifting forwards when walking in place. The remaining parameters scale and offset the sensor values of the gyrometers, which are used as the closed loop component during the calculations of the three movement components.

| Parameter | Description |
|---|---|
| *stepPeriod* | Number of frames to take two steps. |
| $amp_{\text{swing}}$ | Amplitude of the swing calculation. |
| *knee* | Base of the leg lifting calculation. |
| *startLength* | Used in calculating initial ramp up. |
| $v_{\text{short}}$ | Factor for the leg lifting calculation. |
| $a_{\text{short}}$ | Amplitude of the leg lifting calculation. |
| $\phi_{\text{short}}$ | Offset of the leg lifting calculation. |
| $v_{\text{swing}}$ | Factor for the swing calculation. |
| $\phi_{\text{swing}}$ | Offset for the swing calculation. |
| $gyro_{\text{hipPitch}}$ | Body pitch factor for calculating hip pitch. |
| $gyro_{\text{kneePitch}}$ | Body pitch factor for calculating knee pitch. |
| $gyro_{\text{hipRoll}}$ | Body roll factor for calculating hip roll. |
| $gyro_{\text{ankleRoll}}$ | Body roll factor for calculating ankle roll. |
| $scale_{\text{roll}}$ | Scale for sensor value of body roll. |
| $offset_{\text{pitch}}$ | Offset for sensor value of body pitch. |
| $scale_{\text{pitch}}$ | Scale for sensor value of body pitch. |
| *fwdOffset* | Offset to have the robot walk in place. |

Table 1: The walk engine parameters examined in this project.

## 3.2 SimSpark

The RoboCup 3D simulation league uses the SimSpark [2] multi-agent simulator, which was designed by the RoboCup initiative. SimSpark uses the Open Dynamics Engine (ODE) to simulate rigid body dynamics, including collision detection. Although ODE provides a realistic simulation of physics, it does make several approximations. For example, there is no friction model on hinges in ODE, and so no friction acts on the simulated robot's joints [1].

Figure 2: Simulated Nao agent.

The physics simulator's update cycle occurs every 20 milliseconds, at which time it calculates pending events and sends sensor information to the simulated agent. At that point, the agent's behavior code can use the sensor information to determine the robot's next action. To walk, a request for a walk velocity is sent to the walk engine, which uses this request and the sensor information to determine the next desired joint commands. To achieve these joint angles, PID controllers compute torque values that are to be applied to each joint, and then these torque values are sent back to the simulator to process. As we have control over this process, this simulator meets assumption 1 described in Section 2.1.

The simulated agent, shown in Figure 2, was originally based on an older model of the Nao, and so some of the dimensions and masses are different from the physical Nao robot. Also, some details in the simulated model of the Nao are greatly approximated. For example, the simulated robot's foot is box-shaped, in comparison to the curved shape of the physical robot's foot. Additionally, the surface the robot walks on in simulation is completely flat and smooth, which is a poor representation of the carpeted surface encountered on a real robot soccer field.

# 4. GSL INSTANTIATION

This section describes how GSL was applied to optimize the walk speed of a Nao robot. To this end, this section details how the functions listed in Section 2.1 apply to the domain described in Section 3.

## 4.1 Fitness Evaluation

In our $\mathsf{Fitness}_{\mathsf{robot}}$ routine, the robot walks straight forwards towards a ball placed on the field as shown in Figure 3. The robot continually walks forward and turns towards the ball, in case the walk drifts. The robot walks until its feet touch the center line of the field, approximately 238 cm away from the starting location. The velocity of the robot over this distance serves as the fitness of the parameters, averaged over five trials. If the robot falls or veers too heavily off course, the trial is run again. We refer to parameters as stable if they did not result in the robot falling in any of the five trials. The original walk parameters ($P_0$) were stable and were measured at a speed of 11.9 cm/s. This function meets assumption 2 of Section 2.1, allowing for a small number of evaluations to be run on the real robot.

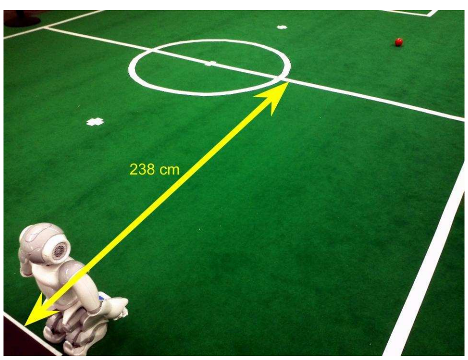


Figure 3: To evaluate the robot, it was placed in the position shown in this image and was ordered to walk towards the orange ball. Once it reached the center line of the field, it was ordered to stop, at which point it recorded its own walking time.

We used two $\mathsf{Fitness}_{\mathsf{sim}}$ routines, both of which were evaluated on the averaged results over 7 runs. The first, denoted OmniWalk, involves the robot walking through an obstacle course, described in [17] as the goToTarget task. In this obstacle course, the robot is given a number of destinations to approach, requiring the robot to walk in all directions and deal with fast changes of direction. The resulting performance is calculated as the sum of the robot's distances traveled toward the various destinations with a penalty applied if the simulated robot falls. In the second fitness function, called WalkFront, the simulated robot walks forward towards a target for 15 seconds. The forwards velocity during this time is used as the fitness of the parameters.

## 4.2 Optimization

In order to optimize parameters in simulation, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm was employed as our Optimize function. CMA-ES was chosen after having found it to perform well when optimizing parameters for similar walking tasks in [31]. This satisfies assumption 5 in Section 2.1. The CMA-ES algorithm is described in detail by Nikolaus Hansen in [10]. CMA-ES uses a population based approach with generations similar to a genetic algorithm. For each generation, CMA-ES produces a population of parameter samples to evaluate, and then uses the fitness of the samples to determine which parameters to try in the next generation. To generate the next generation, rather than using crossover and mutation as a genetic algorithm would, CMA-ES tracks a mean and covariance matrix of the well-performing parameters to bias its sampling. At each iteration, the mean is shifted from its previous position towards the weighted average of some number of the top valued samples (usually half). The new covariance matrix is also derived from these top samples, but it is also adapted based on the trajectory of the mean over the past few generations. The new distribution is then re-sampled for the next generation.

The Condor workload management system [28] was used to distribute the task of evaluating the population across different machines on the department clusters. Up to 150 jobs can be run simultaneously on our system, and so the population size was naturally fixed to 150 parameter sets. The framework reruns samples that fail, regardless of the reason for failure, until 80% of the trials have completed successfully, at which point the algorithm can move to the next generation.

This learning framework is very good at finding parameter values that exploit the fact that the agent is being simulated. For example, to move its joints, the agent calculates a torque value for each joint that will achieve the desired joint command. However, there is very little limit on this torque value, and there is no friction model in the joints. Therefore, the learning framework will take advantage of the agent's ability to move its joints much faster than would be possible on the real robot. Worse, although the agent is penalized for falling during the optimization, the real robot falls more often and is generally less stable than the simulated one. This problem is likely due to the amount of noise encountered in the actuation of the joints on the real robot and the elastisticity of their motion under force. Consequently, it is difficult to force the learning framework to find a walk optimized for speed that will not cause the real robot to be unstable and possibly break.

## 4.3 Grounding

In order to reduce the differences between the robot and the simulator, we modified the behavior of the robot in simulation. To this end, this section specifies the Learn and $\mathsf{Explore}_{\mathsf{robot}}$ routines used in this optimization.

Our $\mathsf{Explore}_{\mathsf{robot}}$ routine collects the state of the robot and the effects of its actions in a variety of situations. Specifically, the $\mathsf{Explore}_{\mathsf{robot}}$ routine uses the OmniWalk routine described in Section 4.1. However, instead of allowing full omnidirectional walk motions, the $\mathsf{Explore}_{\mathsf{robot}}$ routine limits the walk commands to only forwards walking and turning, and the step size is limited to 67% of the maximum. These limitations were added to reduce instability on the robot and reduce wear on the joints. In addition, as the robot does not have ground-truth knowledge of its position, the OmniWalk routine is run in simulation, and the requested walk speeds are recorded. Then, the requested walk speeds are replayed on the robot, and the states and actions are recorded. This routine addresses assumption 3 of Section 2.1 and was chosen because it captures a variety of walking situations, collecting a wide variety of data to be used by the Learn function.

Our Learn function is a supervised learning algorithm which outputs a function to predict the effects of actions taken on the robot (see Section 2.1, assumption 4). The state in our implementation is the robot's current joint angles and center of mass. The goal is to predict the state of the robot in the next frame given the current state and the joint commands issued by the walk engine. New joint commands are generated every 10 milliseconds on the robot, as compared to every 20 milliseconds in the simulator. As the real robot operates at twice the frame rate of the simulator, it collects joint angles on every other frame for each prediction step. The inputs to Learn are the 11 joints of the legs (5 in each leg and a shared HipYawPitch joint), a three-dimensional measurement of the center of mass, and the 11 joint commands, resulting in a 25 dimensional feature vector. The output is an 11 dimensional vector representing the predicted joint angles that will be reached in the next frame. We ran two full iterations of OmniWalk , which took a total of ten minutes and collected approximately 12,000 data points.

We tried a number of supervised learning algorithms for the Learn function using the open source machine learning framework Weka [9]. We evaluated these methods for their runtime and prediction error as shown in Table 2. Following this evaluation, we chose the M5P tree regression algorithm due to its good performance and reasonable runtime. A detailed description of the M5 algorithm can be found in [22]. The resulting model is a decision tree in which each leaf uses linear regression to predict the output joint angle.

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

Table 2: These regression classes were tested with the joint command to joint value data using ten fold cross validation. Here only one joint, HipYawPitch, is presented since this joint was consistently the most inaccurately predicted. RAE stands for relative absolute error, and RRSE is root relative squared error.

The function generated by Learn is used to transform the walk engine's generated joint commands into new joint commands which are converted to torques by PID controllers and sent to the SimSpark simulator. This method is used over modifying the simulator itself for the sake of simplicity. However, directly using the predicted joint angles proved to be impractical due to the noisy predictions of the model, compared to the smooth motions requested by the walk engine. This type of jitter makes the walk unstable, often resulting in the simulated robot falling.

Therefore, we smooth the commands using a linear combination of the requested joint angles and the predicted joint angles. To balance this tradeoff, we performed a series of tests to measure the error difference between the predicted joint angles and the joint angles achieved in each frame. The results of these tests indicated that the best combination used a weight of 0.7 for the requested angles and 0.3 for the predicted angles.

## 4.4 Guidance

The work described in the previous section successfully constrained CMA-ES to select parameter sets which could be run on the real robot. Many of these parameters were generated within the first ten generations of optimization. However, even within these ten generations, many parameter sets had changed enough from the initial parameter set to cause the robot to become unbalanced, and few parameter sets improved the walk speed on the real robot. Even though the search space was constrained, it was still too large. Therefore, we tried to *guide* the optimization to explore promising parameters and avoid damaging ones.

In addition to selecting which parameters to add to OpenParams (lines 8–14 of GSL), we guide the optimization by selecting which parameters to optimize (line 15). Manual analysis of the performance of the parameter sets reveals that modifying seven of the parameters left the robot unbalanced. These parameters are the four *gyro* parameters and the $scale_{\text{roll}}$, $offset_{\text{pitch}}$, and $scale_{\text{pitch}}$ parameters. This problem is likely due to the inaccuracies in the simulation's model of the inertial sensors, which differ greatly from those used on the robot. Therefore, the expert *guided* the optimization by removing these parameters for all iterations of the optimization following the first.

## 4.5 Results

We ran the GSL algorithm twice, first with OmniWalk for $\text{Fitness}_{\text{sim}}$, and again with WalkFront. During optimization, our walk step size was capped at 67% of its original value for stability. At this step size, the orignal walk speed was measured at 11.9 cm/s. A summary of the improvement is shown in Figure 4, and the steps are discussed in greater depth in the remainder of this section. For visual comparison, videos of the robot walking using both original and optimized walk parameters can be found online.[1]

The first run of the GSL algorithm used OmniWalk (described in Section 4.1) as the $\text{Fitness}_{\text{sim}}$. For the first iteration of the algorithm, we tried the optimization with and without the grounding described in Section 4.3. The top one or two best parameter sets from each of the first ten generations of CMA-ES optimization in simulation were tested using $\text{Fitness}_{\text{robot}}$. Those which showed improvement and had distinct parameter values were added to OpenParams. We found several parameter sets which performed better when optimized with grounding, but no parameter sets optimized without grounding were consistently stable.

In the second iteration, the seven balance parameters mentioned in Section 4.4 were locked for the optimization. In addition, one parameter in particular, the swing parameter, appeared to increase the robot's walk speed. The swing parameter controls the amplitude of the swinging foot, and vi-

[1] Video of walking using the original parameters can be viewed at: `http://youtu.be/grlceQkBTxw` and walking using the optimized walk parameters can be viewed at: `http://youtu.be/nGc127yYoSs`

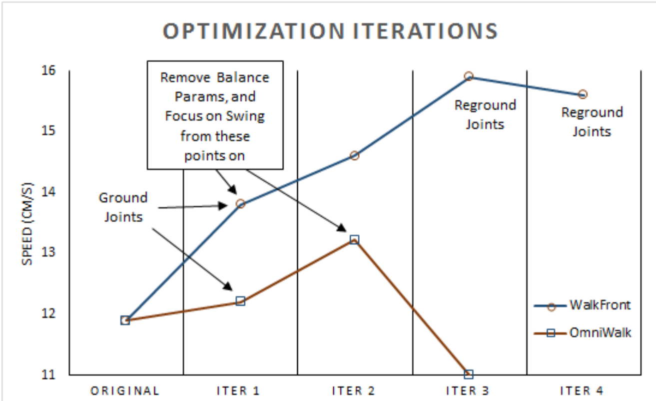


Figure 4: The performance of parameters through iterations of the GSL algorithm, and the guidance used to achieve these results.

sual analysis showed that increasing this parameter caused the robot to swing its lifted foot farther and take larger steps. Therefore, the optimization was guided by increasing the initial variance for the swing parameter to be used in the CMA-ES optimization. This led to a maximum walk speed of 13.2 cm/s. However, the new parameter sets were not as stable as the original. In the third iteration, the re-optimized parameter sets became too unstable to measure.

Therefore, for the second run of GSL, WalkFront was used as $\mathsf{Fitness}_{\mathsf{sim}}$. As before, we grounded the simulation, removed the seven balance parameters and focused on the swing parameter. This resulted in a large improvement from the original parameter set, reaching a speed of 13.8 cm/s. The optimization did not seem to be stagnating at this point, so the second iteration repeated the same methodology as the first, resulting in a walk that moved 14.6 cm/s.

Further investigations at this point indicated that due to the changes in the robot's parameter set, the joint models used in simulation were no longer approximating the walk on the real robot accurately. Therefore, in the third iteration, the grounding step was repeated with the current best parameters, and the optimization was repeated. This step resulted in another large improvement to the walk speed, reaching 15.9 cm/s.

However, while the walk parameters from the third iteration produced a fast walk, the robot fell periodically. In the fourth iteration, the grounding and optimization steps were repeated, but a faster parameter set was not found. Instead, parameter sets were discovered at a speed of 15.6 cm/s, in which the robot did not fall at all in our evaluation.

The step size was not initially optimized and was held at 67% of the maximum because changing its value quickly led to parameters that were unstable on the real robot but worked well in simulation. In addition, putting the step size at its maximum prevented nearby parameter sets from being stable enough to evaluate. To combat this problem, the optimization was guided away from this parameter. However, since the parameter set found in the fourth iteration was as stable as the original parameter set, we were able to increase the step size back to 100%. At this step size, the original parameter set measured at a walk speed of 13.5 cm/s, while the optimized parameters measured at 17.1 cm/s, which is a 26.7% improvement in the walk speed.

# 5. RELATED WORK

Robotic bipedal locomotion has been a hot topic of research in the robotics community in recent years. Much of the research in this area uses the concept of the zero-moment point to predict a stable trajectory for a robot to follow. Such works include [7], [13], [27], and [32].

There has also been work on using machine learning to improve robot performance. Several RobotCup teams have improved the walk speed of the quadrupedal Sony Aibo using machine learning [15, 24, 14]. More recently, there has been progress in learning on bipedal robots, including learning decision tree models to score penalty kicks [11], learning to walk by demonstration [18], and improving walk stability by learning from human feedback [19].

Simulation has long been used as a tool for modern research and development. Thomke [29] describes how simulation has been used for crash testing in the automobile industry for over a decade. Building real models for crash testing is much more expensive and time consuming than building a model in simulation. Additionally, real models are limited in value because they are often built after the automobile's design can no longer be changed and there are a large number of possible crash scenarios. In these cases, it is engineers who learn from the simulation and indirectly apply their findings in real designs.

Many learning algorithms, when first developed, are first tested in virtual environments. Simulation has been used in robotics research to develop and test new algorithms such as path planning models [33]. Also, simulation has been used in the development of learning algorithms for applications such as for modeling robots with many sensors and actuators [21] and, in 1990, for learning evasive maneuvers in flight simulation [8]. Intuitively, learning in simulation lends itself well towards active learning, when a database of training data is not available.

There has also been work on learning to walk in simulation, such as using manifold learning to determine high-level walking decisions [23]. Our implementation is based on the framework created by the UT Austin Villa RoboCup 3D simulation team [17], which produced a fast and stable walk for the RoboCup 3D simulation league using machine learning.

However, the price of the convenience of simulation is its inaccuracy. As Gat puts it, "You can't do science about robots without firing up a robot" [6]. In his 1995 paper, Gat claims that in order for simulation to be useful, the results of simulation must be tested on the real robot. Still, the question remains on how best to make use of those tests.

Koos et al. [16] made a recent effort to answer this question by studying a quantity they labeled "Transferability", which is a measure on how well each component of a simulated model's performance transfers from the simulation to reality. Trials were run on the real robot during optimization to estimate transferability. Unlike this project, in which we made changes to the simulation to increase transferability, Koos et al. made changes to the optimization's objective such that it searches for parameters that improve both fitness and transferability. More examples of using both simulation and real trials in learning include [3], [34], and [12].

In our approach, the optimization was guided manually based on human observation. There has been much work stemming from the concepts of learning from demonstration [25] and using human feedback as a reward. In addition, there has been research in using human demonstration to teach motions to bipedal robots [26] and to teach robots to walk [18], and there has been research in using human feedback to improve walk stability [19].

# 6. ALTERNATIVE APPROACHES

Section 4 describes the method used for applying GSL to optimize a walk on a humanoid robot. However, other approaches were attempted as part of this work, and they are documented in this section.

## 6.1 Predicting Joint Commands

As described in Section 4.3, to ground the simulator, GSL learns a model of the dynamics of the joints on the real robot and uses those models to change the joints commands in the simulator. This model maps the state of the robot and the desired next state into a set of commands to be sent to the joints in the simulator. These commands are then combined with the original commands to create smoother curves. However, further analysis shows that, in the simulator, the joints do not exactly reach these desired targets. This difference is due to the physics of the simulator. Although it is not entirely realistic, the simulator does have the concept of torques and momentum, so the requested joint targets are not always reached.

Therefore, to counteract this problem, we tried to learn a model that predicts commands to send to the simulator in order to reach the desired joint targets. To learn this model, we run a similar technique to the grounding described in Section 4.3, running an exploration routine in simulation and storing the same state and action data. Then, a model was learned through the M5 algorithm. This model allowed us to predict the commands required to send to the simulator to achieve the desired targets.

Unfortunately, while this method did achieve its goal, it did not produce the desired result. The simulated robot was able to match the expected joint angles with an error of less than 0.1 radians for every joint on nine out of ten consecutive frames. However, the joint commands were too noisy compared to the original joint commands produced by the walk engine. This noise caused the simulated robot to fall more often than the real robot, even though it appeared to be more accurately matching the joint angles observed on the real robot. This may have been attributed to the error rate of the joint models and the independence of each joint's model from the others.

## 6.2 Predicting Joints on the Real Robot

In Section 4.3, we describe the method used for grounding the simulator to match the robot's dynamics in the real world. This approach focused on learning a model of the effects of actions, used to modify the simulator. However, an alternative approach is to instead learn the inverse model, learning what actions on the robot are required to achieve the results predicted by the simulator using the M5 learning algorithm once again. Instead of making the simulator act like the real robot, it might be possible to make the robot act similarly to the simulator.

To implement this idea, the robot took the desired angles produced by the walk engine and calculated the commands that would be required to meet these targets on the real robot. To reduce the noise produced by the mapping, the original and predicted values were combined by the function $0.6 * \text{original} + 0.5 * \text{predicted}$. However, this approach reduced the speed of the walk by approximately 30% and failed to increase the stability of the robot when evaluating parameters optimized in simulation.

# 7. CONCLUSION

Using the Grounded Simulation Learning algorithm, machine learning can be used in simulation to improve the values of parameters used on a real robot. In this paradigm, we iteratively ground the simulation, optimize in simulation, evaluate the optimized parameters on the robot, and use this evaluation to guide the simulation towards fruitful parameters in the next round of optimization.

This paper introduces GSL and validates it on the task of increasing the walking speed of a humanoid robot, the Aldebaran Nao. We grounded the simulation by using machine learning to build models for expected joint behavior. Testing the output of the optimization iteratively revealed which parameters were more or less appropriate to optimize. By focusing on those parameters, parameter sets were learned that improved the robot's forward walking speed by over 25%.

There are several avenues for future work in this area, the first of which is to test GSL on additional robotic tasks. Another avenue is to consider automating the selection of which parameters are added to the OpenParams for further investigation. However, this change may slow down the optimization and increase the number of evaluations needed to run on the robot. Finally, more exploration should be performed into the possible methods for *grounding* the simulation.

# Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at The University of Texas at Austin. LARG research is supported in part by grants from NSF (IIS-0917122), ONR (N00014-09-1-0658), and the FHWA (DTFH61-07-H-00030). We would like to give thanks to the members of the UT Austin Villa team and Todd Hester for their help and support on this project.

# 8. REFERENCES

[1] Open dynamics engine faq. `http://ode-wiki.org/wiki/index.php?title=FAQ`.

[2] SimSpark generic physical multiagent simulator system. `http://simspark.sourceforge.net/`.

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

[4] J. A. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *International Conference on Robotics and Automation*, pages 1615–1620. IEEE Press, 2001.

[5] S. Behnke. Online trajectory generation for omnidirectional biped walking. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1597 –1603, May 2006.

[6] E. Gat. On the role of simulation in the study of autonomous mobile robots. In *AAAI-95 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents.*, Stanford, CA, March 1995.

[7] C. Graf and T. Röfer. A closed-loop 3D-LIPM gait for the RoboCup Standard Platform League humanoid. In E. Pagello, C. Zhou, S. Behnke, E. Menegatti, T. Röfer, and P. Stone, editors, *Proceedings of the Fifth Workshop on Humanoid Soccer Robots in conjunction with the 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, 2010.

[8] J. Grefenstette, C. L. Ramsey, and A. C. Schultz. Learning sequential decision rules using simulation models and competition. In *Machine Learning*, pages 355–381, 1990.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009.

[10] N. Hansen. The CMA evolution strategy: A tutorial, 2005.

[11] T. Hester, M. Quinlan, and P. Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010.

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

[13] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *ICRA'03*, pages 1620–1626, 2003.

[14] M. Kim and W. Uther. Automatic gait optimisation for quadruped robots. *Australasian Conference on Robotics and Automation*, pages 1–3, 2003.

[15] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

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

[17] P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone. Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In *Twenty-Sixth Conference on Artificial Intelligence (AAAI-12)*, July 2012.

[18] Ç. Meriçli and M. Veloso. Biped walk learning through playback and corrective demonstration. In *In AAAI 2010: Twenty-Fourth Conference on Artificial Intelligence*, 2010.

[19] Ç. Meriçli and M. Veloso. Improving biped walk stability using real-time corrective human feedback. In *RoboCup Symposium 2010: Robot Soccer World Cup XIV,*, 2010.

[20] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004.

[21] J. M. Porta and E. Celaya. Reinforcement learning for agents with many sensors and actuators acting in categorizable environments. *Journal of Artificial Intelligence Research*, 23:79–122.

[22] R. J. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.

[23] S. Ramamoorthy and B. Kuipers. Trajectory generation for dynamic bipedal walking through qualitative model based manifold learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-08)*, 2008.

[24] M. Saggar, T. D'Silva, N. Kohl, and P. Stone. Autonomous learning of stable quadruped locomotion. In G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pages 98–109. Springer Verlag, Berlin, 2007.

[25] S. Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems 9*. MIT Press, 1997.

[26] A. Setapen. Exploiting human motor skills for training bipedal robots. HR 09-02, The University of Texas at Austin, May 2009. `ftp://ftp.cs.utexas.edu/pub/techreports/hr09-02.pdf`.

[27] J. H. Strom, G. Slavov, and E. Chown. Omnidirectional walking using zmp and preview control for the nao humanoid robot. In J. Baltes, M. G. Lagoudakis, T. Naruse, and S. S. Ghidary, editors, *RoboCup*, volume 5949 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009.

[28] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[29] S. H. Thomke. Simulation, learning and r&d performance: Evidence from automotive development. *Research Policy*, 27(1):55–74, May 1998.

[30] R. Tilgner, T. Reinhardt, D. Borkmann, T. Kalbitz, S. Seering, R. Fritzsche, C. Vitz, S. Unger, S. Eckermann, H. Müller, M. Bellersen, M. Engel, and M. Wünsch. Team research report 2011. Technical report, Leipzig University of Applied Sciences.

[31] D. Urieli, P. MacAlpine, S. Kalyanakrishnan, Y. Bentor, and P. Stone. On optimizing interdependent skills: A case study in simulated 3D humanoid robot soccer. In K. Tumer, P. Yolum, L. Sonenberg, and P. Stone, editors, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, volume 2, pages 769–776. IFAAMAS, May 2011.

[32] E. R. Westervelt, J. Grizzle, and D. E. Koditschek. Hybrid zero dynamics of planar biped walkers. *IEEE Transactions on Automatic Control*, 48:42–56, 2001.

[33] S. X. Yang and M. Meng. Real-time collision-free path planning of robot manipulators using neural network approaches. *Auton. Robots*, 9(1):27–39, Aug. 2000.

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