Optimizing Humanoid Robot, Nao, Walk Using Particle Swarm Optimization

Syed Hasan Faaz Abidi
School of Science and Engineering
Habib University
Karachi, Pakistan
sa06195@st.habib.edu.pk

Habib Shahzad
School of Science and Engineering
Habib University
Karachi, Pakistan
hs05888@st.habib.edu.pk

Syed Hammad Ali
School of Science and Engineering
Habib University
Karachi, Pakistan
sa04324@st.habib.edu.pk

***Abstract*—This paper introduces the use of a multi-objective Particle Swarm Optimization (PSO) to evolve the walk of a Nao humanoid robot. The quality of the walk is determined by the speed, distance, stability, and, non-divergence of the robot. We aim to find the optimal angles of the joints of the robot that yield to the best walk. After evolving, we got a set of joint values that were able to make Nao walk.**

***Index Terms*—PSO, nao, humanoid, particle, robot, walk**

I. Introduction

In recent years, there has been an increasing interest in using humanoid robots in a variety of settings, from industrial and service applications to search and rescue missions. One of the key challenges in using humanoid robots is designing them to move effectively in unstructured environments. This is particularly challenging for walking, which is a complex task that involves the coordination of many joints and muscles.

In this paper, we explore the use of particle swarm optimization (PSO) to optimize the walking of a humanoid robot. PSO is a computational technique that is inspired by the social behaviour of animals, and has been shown to be effective in optimizing complex functions. We apply PSO to the problem of designing a humanoid robot walk that is efficient and effective in unstructured environments. We present the results of our approach with and show that PSO can be used to generate high-quality solutions for this complex problem.

The rest of the paper is organized as follows: Section II provides the technical background of the paper covering the details of the Nao robot used in the paper, the definition and the math behind PSO, and the previous work in the domain. Section III gives the description of the problem including the Nao walk, and the parameters to be optimized in our solution. Section IV details how the problem was formulated including the walking mechanism, the particle representation, the PSO and hyper-parameters, and lastly the fitness function used. Finally, sections V and VI detail the experimental setup and the results we obtained through our methods respectively. Section VII provides the conclusion to our work, and section VIII entails the future work that can be done by building on to our methods.

II. Technical Background

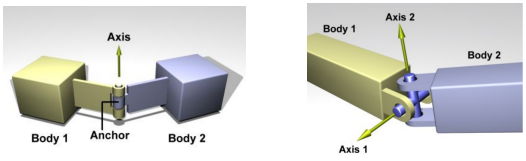
Before going into the details of the problem, it is crucial to know a few things that are foundational to both the problem and solution proposed in the paper.

A. Nao Robot and RoboCup

Nao (pronounced now) is an autonomous, programmable humanoid robot developed by Aldebaran Robotics. It is a small open source robot which is used by many universities as a research platform and educational tool. It resembles the human body in shape and is built to mimic human motion and interaction. It's packed with sensors and it can walk, dance, speak, and recognize faces and objects. Now in its sixth generation, it is used in research, education, and healthcare all over the world.

RoboCup is a soccer league where Nao humanoid robots play soccer against each other. It investigates a variety of scientific research issues in the field of robotics, including dynamic walking, running, and kicking the ball while preserving balance, visual perception of the robot, self-localization, and interaction with other robots.

If we talk more about Nao, it has a total of 22 joints. Most of them are hinge joints, which means that these joints are connected to such body parts that can move along a single axis only. The other type is universal joints. In these joints, the connected body parts can move along different axes. The difference is also evident in the fig. 1.



(a) Hinge (b) Universal

Fig. 1. Types of joints

More details about the joints can found in the official Robocup manual [6]. Depending upon the walk, Nao can utilize either some or all joints. Table shows the joints that we used with their perceptor aliases.

No.	Description	Perceptor
1	Left Shoulder Pitch	laj1
2	Left Shoulder Yaw	laj2
3	Left Shoulder Roll	laj3
4	Left Arm Yaw	laj4
5	Left Hip YawPitch	llj1
6	Left Hip Roll	llj2
7	Left Hip Pitch	llj3
8	Left Knee Pitch	llj4
9	Left Foot Pitch	llj5
10	Left Foot Roll	llj6
11	Right Hip YawPitch	rlj1
12	Right Hip Roll	rlj2
13	Right Hip Pitch	rlj3
14	Right Knee Pitch	rlj4
15	Right Foot Pitch	rlj5
16	Right Foot Roll	rlj6
17	Right Shoulder Pitch	raj1
18	Right Shoulder Yaw	raj2
19	Right Shoulder Roll	raj3
20	Right Arm Yaw	raj4

Considering the scope of this paper, we can say that perceptor aliases are merely for telling the simspark server which joint to move with the given angle.

B. Particle Swarm Optimization

Particle swarm optimization is a bio-inspired algorithm which is applied to optimization problems to search for an optimal solution in a solution space. Individuals or particles are represented as n-dimensional vectors where n is the dimension of the problem. It starts with the initialization of a population of particles randomly distributed in the search space. The position of particle i at time t is denoted as x_i^t and the velocity is denoted as v_i^t. Each particle is assigned a objective value or fitness value which is calculated by the fitness function. At every step the positions and velocities are updated as:

$$v_i^{t+1} = \omega v_i^t + r_1 c_1 (pbest_i^t - x_i^t) + r_2 c_2 (gbest^t - x_i^t) \quad (1)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (2)$$

where r_1 and r_2 are random numbers uniformly distributed between 0 and 1. $pbest_i^t$ is the i^{th} particles best position and $gbest^t$ is the global best of the whole population. ω is the inertia weight constant which determines the speed and direction of search. c_1, c_2 are cognitive and social coefficients which control the trade off between exploration and exploitation. Using this mechanism, the particles learn from self-cognition and social interaction to approximate the optimal solution of the problem.

C. Previous Work

There has been some previous work in the field of optimizing humanoid robots; however, most of the research done is on the Genetic Algorithms, and where PSO is used, it is not used the Nao robot that we are experimenting with.

In the paper [1], The genetic algorithm (GA) is used to generate the fitness-based best sitting pose for the robot to fit comfortably on the sittable-object using an initial collection of random valid sitting poses as the input generation (i.e. box and ball). Poses are converted into numerical stability levels using fitness criteria that represent pose stability (i.e. how practical the position is based on real-world physical limitations). The suggested approach's feasibility is tested in a simulated setting using the V-Rep simulator, which demonstrates how the GA can generate a fitness-based ideal sitting-pose. The simulation's results are implemented using the real "NAO" robot.

In paper [2], the work done by the researchers addresses the issue of creating fast and stable walking for humanoid robots. They do this by employing a model-free method in which each joint is represented by a sinusoidal function of time. Using a genetic algorithm, the parameters of the functions for all actuated joints are optimised. Under V-REP, experiments were conducted with a NAO robot in a simulated environment. The optimised robot could walk in a straight line at 54cm/s for up to 200 metres without falling. Individuals' adaptability to varied situations, such as walking up and down ramps, was also assessed through experiments. For the robot walking uphill, various movement patterns, a slower pace, and more erect positions were observed.

In paper [3], the work offers a novel walking gait generation technique for biped robots based on inverse kinematics. The suggested technique uses the PSO (Particle Swarm Optimization) algorithm to get optimal values for the five walking algorithm parameters. In the DCSELAB at Vietnam National University, a small humanoid robot is used to test the proposed experimental method. Simulation and experimental data indicate that using the PSO algorithm with an effective fitness function can greatly cut learning time. In addition, a rapid walking pace is achieved.

III. Problem Description

The problem that we are targeting is Nao's walk. Starting from random movements of Nao's joints, we are aiming to make it walk and then evolve that walk as much as possible. So, the problem comes down to finding best suited movements for each of Nao's joint. However, optimizing joints' value is only possible if we have a proper locomotive mechanism which takes joints value and apply them to Nao. Therefore, the problem is to first figure out a walking mechanism and then use that to evolve Nao's joints.

A. Nao Walk

The first thing to note in the motion of walking is that it is periodic. Walking happens when a set of specific motions happen periodically. So, a significant part of the problem was to figure out a mechanism which can use joint values and make the Nao move in a way which maximizes the possibility of walking or locomotion.

B. Evolving Parameters

Finding the best values for the joints was the heart of our problem. There are total of 22 joints as shown in the table above. Considering all these joints, the solution space that they make is huge. Finding a set of values from all possibilities was our biggest challenge. To make things slightly simple we limited our joints to 20 instead of all 22. We left 2 joints out for the reasons discussed later in this paper. Still the solution space consisting of 20 joints is huge. Each joint is a dimension itself and we are solving for a 20th dimensional vector. We tried to evolve all 20 of these joints.

IV. Problem Formulation

A. Walking Mechanism

As discussed above, walking is a periodic motion. We need to our joints to move in a cycle. Keeping this in mind, we formulated a mechanism that follows a counter approach. Counter is, basically, our time parameter. We created a cyclic movement of our walk using this counter. The core idea was check the value of the counter. If the counter value is below certain value then move all the joints with certain values. And if the counter is between another range, then move all the joints with certain values. Lastly, if the counter exceeds a certain threshold, reset it. This created our entire walking cycle. The time of counter for the entire cycle and its halves was manually tuned by us.

We divided the entire walking cycle in to two halves. Ideally, each half is for moving each leg i.e two. For example, for a certain set of joint values, there is a possibility that in the first half, right leg will move forward and left will mainly stay at rest, and, in the other half, left leg will move forward and vice versa. This motion in loop will make the Nao walk. However, since, values of joints are being evolved from randomness, there might be a global maxima where this convention is not true, and both legs will move significantly in each half. Therefore, our approach to map joints value to walking motion will also perform well in such instances as it doesn't eliminate such possibilities.

B. Particle Representation

The particle in PSO is a n-dimensional vector and we can easily map the representation of joints into a vector.

Joint.	Min Value	Max Value
hj1	-120.0	120.0
hj2	-45.0	45.0
raj1	-120.0	120.0
raj2	-95.0	1.0
raj3	-120.0	120.0
raj4	-1.0	90.0
laj1	-120.0	120.0
laj2	-1.0	95.0
laj3	-120.0	120.0
laj4	-90.0	1.0
rlj1	-90.0	1.0
rlj2	-45.0	25.0
rlj3	-25.0	100.0
rlj4	-130.0	1.0
rlj5	-45.0	75.0
rlj6	-25.0	45.0
llj1	-90.0	1.0
llj2	-25.0	45.0
llj3	-25.0	100.0
llj4	-130.0	1.0
llj5	-45.0	75.0
llj6	-45.0	25.0

Lets say that we have a list of joints such that $joints = [joint_1, joint_2, \ldots, joint_n]$. We will construct a vector v such that v_i is a randomly generated angle uniformly distributed between the range of minimum value of $joint_i$ and the maximum value of $joint_i$ (from the table above). This vector v will be the particle representation. We also not that in the list of joints, $joint_i$ is a right joint and $joint_{i+1}$ is the corresponding left joint and $joint_{i+2}$ is another right joint and $joint_{i+3}$ is another left joint and so on.. This essentially helps us in mapping the particle vector into a joint configuration which can be given as a parameter to the nao robot. We can easily map the vector into a configuration, such that
$config_0 = (v_0, v_1)$,
$config_1 = (v_1, v_0)$,
$config_2 = (v_2, v_3)$,
$config_3 = (v_3, v_2)$,
$\ldots$,
$config_{n-1} = (v_{n-1}, v_n)$,
$config_n = (v_n, v_{n-1})$.
This mapped configuration is helpful in periodic walking (as discussed in the previous sections).
However, for this problem we only use 10 joints which are associated with the leg of the robot.

C. PSO and hyperparameters

To evolve values of 10 joints of Nao, we applied PSO. The reason choosing PSO was natural as the solution of problem is continuous, eventhough it is bounded by the joints maximum and minimum values. We mostly applied classic PSO settings in which we generated random particles with random positions and velocities. The position was these particles represents values of our joints. Position of each particle was a $10th$ dimensional vector (also discussed in particle representation).

Moreover, velocity was also a $10th$ dimensional vector generated randomly in range $[-1, 1]$.

Below is our implemented PSO algorithm for evolving Nao walks,

Algorithm 1 Paricle Swarm Optimization for Evolving Nao Walk

1) Initialize Population with n particles with randomly generated positions and velocities.
2) Run loop for $iterations$ times
   a) Iterate over all particles
      i) Evaluate each particle. For evaluation, run $rcssserver3d$ [5] and RoboViz Simulation [6], and spawn an agent with the given position vector (joint configuration).
      ii) Let it run for 20 seconds.
      iii) Then terminate the server and close the simulation.
      iv) Observe and update the fitness value of the particle along with personal best and personal best position
      v) Compare particle's fitness with global best
         A) if particle fitness is more than the global best than update global best with particle fitness value, and global best position with particle's position
         B) else continue
   b) Iterate over all particles
      i) Update particle's velocity with the formula given in section II.
      ii) Update particle's position
3) Repeat until Convergence

The hyper-parameter were not enough in our implementation which is a good thing as we didn't need to figure out tuning them for better results. The cognitive factor, c_1, and the social factor, c_2, were manually tuned in a way that at the early iterations the values of these two were low as we wanted the PSO to be more explorative. Moreover, there was inertia constant, ω which play a key role in position updates and support exploration as well. We also tuned it high in the starting iterations. As the iteration increased, we increased c_1 and c_2, and decreased ω.

The population size was constant throughout which was 15 and the number of iterations were around 500.

D. Fitness Functions

The fitness is a multi-objective function such that given a nao robot, it is trying to achieve three objectives which are distance, stability, and, non-divergence. We can compute a score for these factors by waiting for a certain amount of time (letting the robot) walk in the field and calculating the scores accordingly.

We can determine the position of the robot using a localizer that use twos poles to compute the position of the robot in real time.

Lets say initially $fitness = 0$

1) Distance:

This objective is to ensure that the robot covers the maximum distance.

If $p_0 = (x_1, y_1)$ is the initial position of the the robot and $p_{20} = (x_2, y_2)$ is the position of the robot after 20 seconds then the distance would be $distance(p_{20}, p_0)$ such that each position is simply just an (x, y) co-ordinate in the field and the euclidean distance between the two positions can be calculated by this formulae:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3}$$

The score will updated such that $fitness = d \times 30$

2) Stability:

This objective is to ensure that the robot does not fall down. if the robot falls down then we assign a penalty to the robot and update the score such that

$fitness = fitness - (100 \times penalty)$ where penalty is a time dependant variable.

3) Divergence:

This objective is to ensure that the robot walks in a straight line.

The divergence error or the straight line error is simply the difference between y_2 and y_1 such that $error = abs(y_2 - y_1)$

We update the score such that

$fitness = fitness - (error \times 10)$

V. Experimental Setup

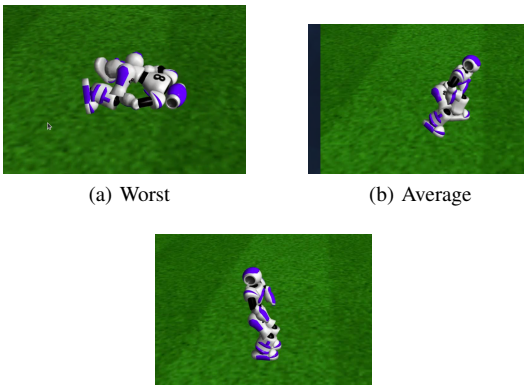
Linux was used as our primary OS to conduct all this experiment of running the algorithm and testing it through RoboViz [6] simulation. Simspark was also essential to run and execute our agent. In addition, we created a script that automated the entire process of evolving joints using PSO. This was essential as each particle took 20 seconds for evaluation only. The script was written in a way that it automatically starts the rcssserver3d and terminates once the agent had been evaluated. This was a big relief in our long and tiring evolving process.

Evaluating a single agent was done for 20 seconds. In, 20 seconds, we evaluating our agent on the basis of three aspects: distance covered, stability, and walking in a straight line. If an agent is walking with stability, cover a substantial distance, and is walking in a straight path, then it would evaluated with good fitness score.

VI. Results

The results of the problem we started with in the beginning was surprising. We started training with pure randomness and the final evolved values of joints we have evolved are able to make our Nao robot walk. The final walking Nao is checking all the points. It has a stable walk (such that it's not falling), it is covering a good distance in time span of 20 seconds. And lastly it tries to be in a straight line and diverts very little.

Below are some of the pictures for comparison between worst fittest, average fittest, and best fittest agents.



(a) Worst (b) Average

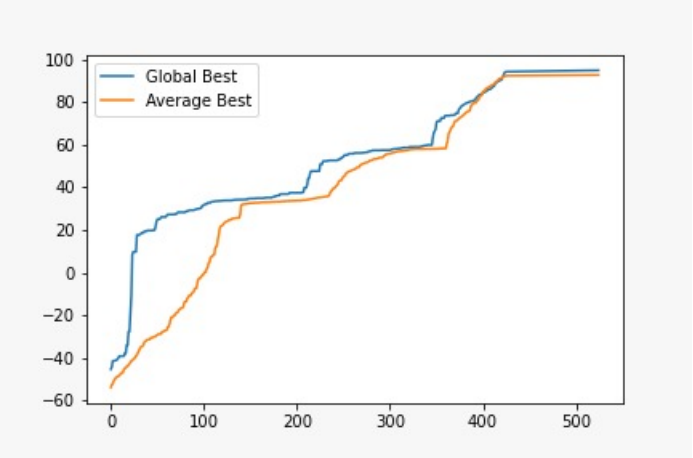
(c) Best

Fig. 2. Comparing Agents with different fitness scores

It is evident from the pictures above that evolving the joints with complete randomness gives the Nao a complete different posture. It may seem weird to us (humans) as it bends a little too much while walking, but, it makes sense. The posture was not the focus. The fitness functions were. It evolves to score good according the fitness functions that we set.

In addition to the Nao itself, we have an interesting learning pattern. Below, fig. 3, is the convergence graph for global best fitness and average best fitness.

Fig. 3. Global and Average Best Vs Iterations



We can see from the figure above that both global best and average best converges after around 500 iterations. However, there are converging patterns in between as well. For example, around 100-200, it seems like it's about converges, but it shoots up again. It is because of the changing hyper-parameters, we tweaked our hyper-parameters during the evolution process. One downside of PSO is that it can get stuck at the local optima as the entire population in PSO converges. Therefore, it was crucial to tweaked exploration and exploitation as much we could.

VII. Conclusion

To conclude the work that we did in this paper, the main takeaways are that PSO is very efficient for continuous problem. We also learned that the area of evolving walk is huge. And, even though our Nao is performing quite good, there is a lot of room for improvement. We can also conclude that PSO can be altered depending on the problem that one solving and making it more specific can yield better results.

VIII. Future Work

We may increase our solution space to include a higher number of joints and completely automate the entire process using some scripting and run the algorithm for a longer time to achieve better results.

Acknowledgment

We would like to thank Dr. Syeda Saleha Raza for her effort in teaching us the core concepts about computational intelligence and guiding us on nao locomotion and localization. We would also like to acknowledge that we forked the repository mylamour-github-robocup and used it as our base code to optimize our robot.

References

[1] Al-Hami, M. Lakaemper, R. (2014). Sitting pose generation using genetic algorithm for NAO Humanoid Robots. 2014 IEEE International Workshop on Advanced Robotics and Its Social Impacts. https://doi.org/10.1109/arso.2014.7020994

[2] Villela, L. F. C., Colombini, E. L. (2017). Humanoid Robot Walking Optimization using Genetic Algorithms.

[3] Huan, T. T., Anh, H. P. H., Kondo, Y., Han, X., Wei, X., Chen, Y. W., ... and Fang, X. (2015). Novel stable walking for humanoid robot using particle swarm optimization algorithm. hip, 6(6), 6-6.

[4] "Server · Wiki · RoboCup Simulation / SimSpark · GitLab", GitLab, 2022. [Online]. Available: https://gitlab.com/robocup-sim/SimSpark/-/wikis/Server.

[5] Simulation, "3D Soccer Simulation – RoboCupSoccer Simulation League", Ssim.robocup.org, 2022. [Online]. Available: https://ssim.robocup.org/3d-simulation/.

[6] "GitHub - magmaOffenburg/RoboViz: Monitor and visualization tool for the RoboCup 3D Soccer Simulation League", GitHub, 2022. [Online]. Available: https://github.com/magmaOffenburg/RoboViz.