Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 24 (2013) 158 – 166

Procedia Computer Science

17th Asia Pacific Symposium on Intelligent and Evolutionary Systems, IES2013

The Automated Fault-Recovery for Four-Legged Robots using Parallel Genetic Algorithm

Hyunsoo Park, Kyung Joong Kim*

Dept. of Computer Engineering, Sejong Univ., Seoul, Republic of Korea

Abstract

It is expected that robots could operate autonomously in extreme environments without human intervention. However, it is not a trivial task to design robots robust to all kinds of faults. On the other hand, Biological entities have capability to create new behaviors overcoming unexpected damage on their body. This is also one of desirable properties for industrial robots operating remotely in extreme conditions. In this paper, we propose to use a bio-inspired learning algorithm to generate new behaviors on a four-legged robot against unexpected body damages. Since the learning algorithm can be accelerated using parallelism on multiple machines, it is possible to adapt to changes (damages) quickly using remote computational resources. Experimental results show that the robot could adapt to damages on different part of robot’s body successfully.

Keywords: fault-tolerant; robot; parallel genetic algorithm; neural network; physical-based simulation

1. Introduction

Animals’ brain is resilient to physical damage on their body allowing organism to survive for long-time. For example, a lizard uses four legs with one tail to generate its movements and the animal could recover its motion from losing one leg or tail by slightly adapting its behavioral patterns. This property is also desirable for mechanical

* Corresponding author.
E-mail address: hspark@sju.ac.kr (Hyunsoo Park), kimkj@sejong.ac.kr (Kyung Joong Kim).

robots operating in extreme environments without human operator [1]. Since humans are not close to the robot, the machine's failures significantly reduce the probability of successful mission completion.

Animals' brain is resilient to physical damage on their body allowing organism to survive for long-time. For example, a lizard uses four legs with one tail to generate its movements and the animal could recover its motion from losing one leg or tail by slightly adapting its behavioral patterns. This property is also desirable for mechanical robots operating in extreme environments without human operator [1]. Since humans are not close to the robot, the machine's failures significantly reduce the probability of successful mission completion.

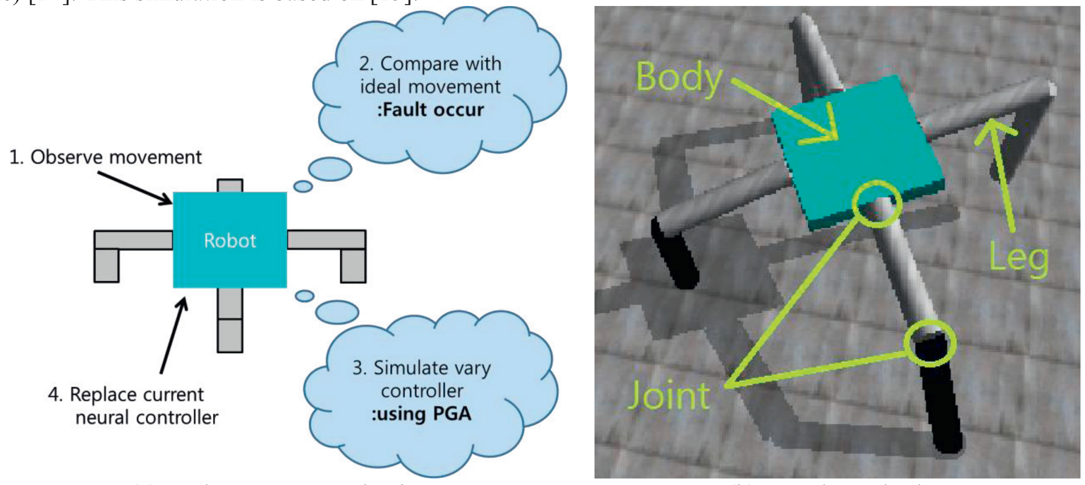
The easiest way to make the robot as fault-tolerant is to have redundant hardware [2]. However, redundancy is costly expensive and it often makes the control system complex. It is not possible to consider all possible faults prior to damage but stores all the solutions for known faults. To overcome those limits, it needs a mechanism to recover from unexpected damages [3]. In this approach, the robot executes machine learning algorithms to learn a new behavior if it detects broken part of body. Because the old control mechanism designed for intact body would fail to work correctly, it is necessary to learn new skills suitable for the new body. This allows the robot lose its functionality gracefully instead of radical performance degradation.

In this paper, we propose to use Parallel Genetic Algorithm (PGA) to generate a new compensatory behavior for unexpected damage on robot's body (Fig. 1. (a)). Genetic Algorithms (GA) mimics the nature's process to reproduce new genes adaptable to changing environments [4]. It has been widely used to solve engineering optimization problems and known to be strong for global search [5][6]. And there are many works automatically generate robot controller using GA or GA like evolutionary techniques [7][8][9][10][11]. Initially, it generates a population of solutions represented as a bit-string randomly and evaluates their goodness. Based on the fitness to the problem, they can reproduce their offspring (slightly modified solutions) for the next generation using mutation and crossover operations. It is possible to accelerate the speed of the genetic algorithm using parallelism with a cluster of computers [12], or General Purpose Graphic Processing Units (GPGPU)[13]. It allows the rapid adaptation of robot controllers for the emergency situations.

2. Robot

2.1. Robot body and controller

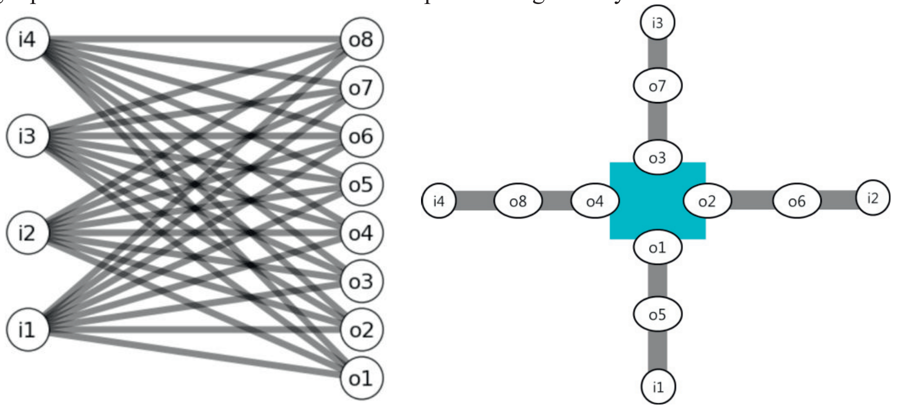
A four-legged robot was used for our experiments. The robot has four touch sensors and it returns positive value if the leg touches down on the ground. The robot has one box-shaped body and four legs with eight joints in total (Fig. 1. (b)). Simulation environment have no obstacle plane ground and physics engine ODE (Open Dynamics Engine) [14]. This simulation is based on [15].


1. Observe movement
Robot
2. Compare with ideal movement :Fault occur
3. Simulate vary controller :using PGA
4. Replace current neural controller
Body
Leg
Joint

(a) Fault recovery mechanism (b) Four-legged robot

Fig. 1.Fourlegged robot and Fault recovery mechanism

A simple neural controller gets inputs from the touch sensors and outputs the actions of the eight motors attached to the joints. In total, the neural network has 32 weight parameters. The genetic algorithm encodes the vector of weights as an array of real values. If the robot detects damage on its body, it executes the parallel genetic algorithm (PGA) to learn new behaviors quickly. It means robot generate new neural controller using evolving technique to determine weight parameter of each links in order to adapt to damaged body.



(a) Neural controller (b) Placement of sensor and actuator

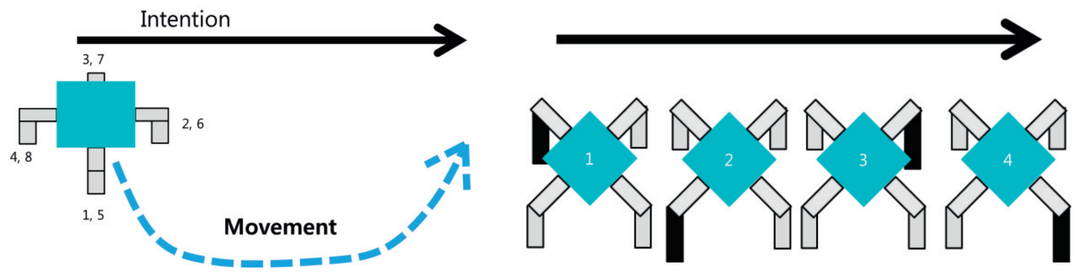
Fig. 2. Relation of neural controller and sensor/actuators (output neuron number is corresponding to joint number)

Fig. 2 represents the topology of neural networks and robot body's points corresponding to sensors and motors. Sensors are labeled i1-i4 and motors are labeled o1-o8. The sensors are attached to the end of legs and motors are placed in the joints connecting robot parts. If a leg touches down on the ground, attached sensor transmits positive value to the neural controller.

2.2. Intact behavior evolved

We assumed High level of robot's intention is represented by its trajectory. Thus, this trajectory represents baseline movement of robot's "Go forward" intention. In this paper, we defined robot's forward direction is figure's right direction or X-axis positive direction. Robot remembers this behavior, when simulation robot compares with this (baseline) behavior and current (simulated) behavior in same intention. If these two trajectories are different significantly than robot thinks some fault is occurred and try to adapt this fault using genetic algorithm with simulation environment.

The intact behavior of the robot is evolved using genetic algorithm. Fitness function measures the distance traveled to specific direction by the controller. It is interesting that the evolved controller shows curved trajectories with horse's gallop gaits.



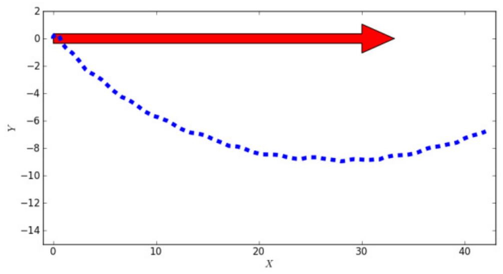
(a) Intention and real movement trajectory (b) Sequence of robot legs touched on the ground

Fig. 3. Movement of evolved intact body controller

Most of cases evolved robot’s behavior is similar horse’s gallop. Generally, Horse’s gaits are classified into walk, trot, canter and gallop [16]. Gallop is the fastest gait. On the other hand, it is unstable than others gaits. When initial stage of evolution, the robot is in low speed and its movement is different to the gallop rather than it is similar to walk, trot or canter. It is more stable gaits. The robot touches down on the ground using different legs in turn. It seems like horse riding (Fig. 3. (b)).

In this case, stability can measure by how many legs on ground at each time. For example, when the robot move high speed (perform gallop gaits) than its only one leg on ground at a time, even between each footfall no leg on ground in short time. Contrary, when the robot move low speed (perform walk gaits) always three leg on the ground and only on leg move. If there are some obstacles or disturbance, the robot can perform walk gaits easily than gallop.

Because we define fitness that how far robot traveled to specific direction (Intention) and there are no obstacles and disturbance, most of evolved controllers perform gallop gaits not walk gaits. Biological entity like horse evolved in long time to perform this gait in order to move high speed. Therefore maybe gallop gait is the most efficiency movement in these conditions.


(a) Intact body trajectory

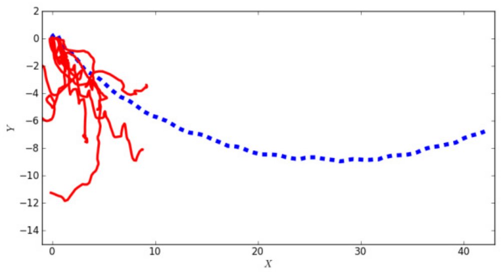

(b) Damaged body trajectories

Fig. 4. Baseline trajectories (a) dotted line trajectory of robot’s body center, red arrow mean robot’s intention; (b) Damaged body trajectories with the best controller

Curved trajectory is the one thing we can’t predict before experiments. We define robot’s intention that the robot move as far as possible along x-axis. But, robot’s initial orientation is not parallel to x-axis. Basically, robot don’t have front, back, right and left, but if assume the robot’s finally movements are same to horse’s gallop then there are two possible front side of the robot. First front side is between joint 3, 7 and 2, 6 and second front side is between joint 2, 6 and 1, 5. If one of these two sides become (perform a role) a robot’s front side then evolved final trajectory of the robot is curved line. Become which side become front side is depend on initial simulation state. In this experiments condition the side between joint 2, 6 and 1, 5 become front side.

Fig. 4. (a) shows robot’s intention and baseline trajectory used in this work. Straight arrow represents robot’s intention and dotted line represents robot’s movement trajectory. Robot’s intention is go forward as fast as in limited time. But, robot has limitation by structure of body, controller, initial condition in simulation and etc. so, robot couldn’t move straight forward, instead it take curved trajectory. In this trajectory, robot move forward distance (X-axis) is 42.441. X-axis means robot’s front and backward directions, Y-axis means left and right side directions.

Fig. 4. (b) shows example trajectories of the best controller with various damaged body. With intact body the best controller could travel along X-axis over 40. But, with damaged body (broken joint) robot couldn’t travel even over 10.

3. Fault recovery algorithm

In this paper, we use typical Real-Value Genetic Algorithm (RVGA) to create new neural controller. We modify the evaluation stage to use parallel processing. Because the evolution requires complicated robot simulation with a physics engine, therefore the evaluation stage becomes a bottleneck for rapid adaptation.

3.1. Initialization

Initially, it creates randomly a set of the array each encoding different neural network. Each array is consist 32 real-values range in [-1, 1]. These values are means neural controller's weights. Generally, entire set of array called population.

3.2. Crossover and mutation

We use crossover and mutation for generate next offspring. In crossover stage, we choose randomly two arrays (neural controller) from population. These selected arrays call parents, and generate offspring controller from combining two parents randomly. We use single point crossover method. (1) Randomly select array's index range in [0, 31]. (2) Slice each parent at selected index and join one parent's front part and other parent's real part. In this manner, create offspring same number of parents. If total parents size is N, than creates N offspring. In mutation stage, randomly add small number N(0, 1) to each element in very small probability.

3.3. Evaluation

In this work, the goal of the recovery mechanism is to make its trajectory with the damaged body as close as possible to the intact one. The similarity between new trajectory and the original one is calculated using sampling-based method. It chooses 100 points (x, y coordinates) to get Euclidian distance between them. bt_i means baseline trajectory's sampled point, and ct_i means current trajectory's sampled point.

$$fitness = \sum_{i=0}^{100} distance(bt_i, ct_i)$$

Evaluation stage is very time-consuming job. Because of, each evaluation is complicated physics-based robot simulation. Thus we try to parallel processing to accelerate this stage. But, other stage is relatively consume trivial time in modern computer therefore other run serial.

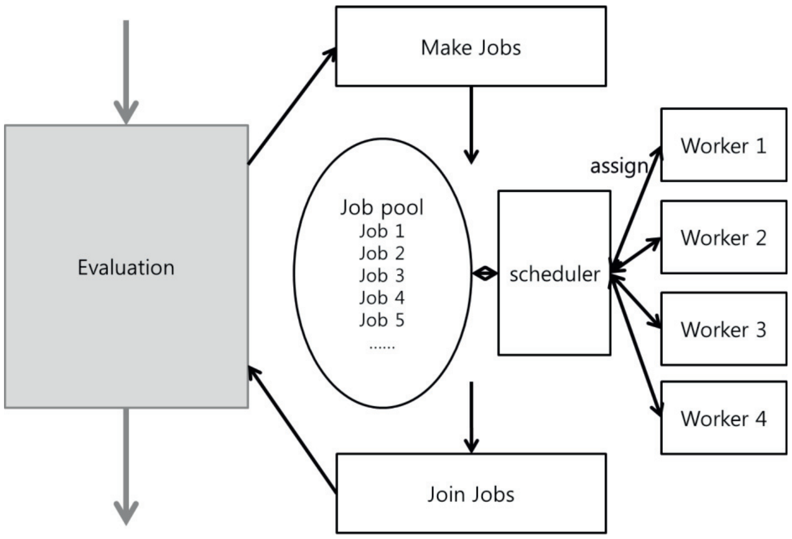


Fig. 5. Parallel processing procedure for evaluation

Evaluation stage is very easy to split independent jobs. It is not need to interact others that to evaluate each individual (neural controller). So, we apply classical fork-join model. Each neural controller runs on separated process with argument and finally it returns fitness value only. We use Python programming language and "Parallel Python" [17] module that parallel processing framework supporting simple clustering. Python is script language, it support vary useful modules (library), easy to understand and fast development. Parallel Python is one of parallel processing framework, it support process based parallel processing, fault-tolerance, scalability, scheduling. Its usage is very easy and it is reliable. Our purpose is not optimized GA to parallel processing than excess use of resources with at least effort. Thus, we use various machines in cluster.

Fig. 5 shows parallel processing steps. (1) Make job pools in main process. Work processes are running in other machines. (2) Scheduler assign job to worker process, one job per one core. If job is fail then scheduler reassign job to other server. Of course, if worker process job is finished, than scheduler assign other job to worker process. (3) Job running is complete then, job returns fitness value to main process. (4) Finished job wait end of evaluation stage, when all jobs are complete. It is synchronization overhead. Each evaluation executes independently but, it is need to complete all evaluations to proceed to next stage (selection). Therefore, population size has to large enough to minimize overhead. If population size is too small, then many processes wait until the last evaluation is complete.

3.4. Selection

In this paper, selection method is simple. Choose best half population from entire population. Other half population did not selected is delete from population. To achieve this, just sort by fitness and delete half from end of population. It's some sort of elitist method.

4. Experimental results

In our experiments, the population size is 50 and the maximum number of generations is 300. In our work, we use five computers in total 14 cores to parallelize the genetic algorithm. It is 6-7 times faster than one core computer (single worker process). Table 1 summarizes the parameters.

Table 1. An example of a table

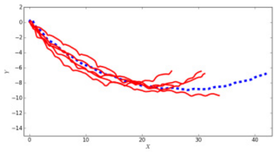
Condition	Values
Population	50
Generations	300
Crossover rate	0.9
Mutation rate	0.05
Cluster hosts	14 cores of various CPUs (6-7 times speedups)
# of experiments	5 times

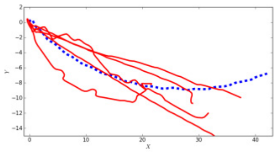
We give damage to each joint and run the PGA searching for a new controller suitable to the changed body topology. The distance traveled by the new controller is compared with the one from the intact body. We run the experiments five times for each joint. Its results are summarized in Table 2. Performance recovery ratio is measured by the distance traveled by the new controller compared to the original one. Fig. 6 shows the trajectories of five recovery runs (dotted line from the intact body and the solid lines are from new controllers recovered).

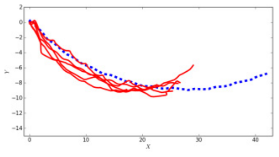
Table 2. Summary of the experimental results (1-4 are upper and 5-8 are lower joints)

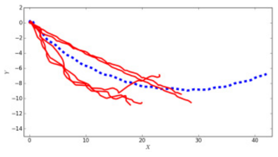
Joints #	Average distance traveled	Performance recovery ratio
1	29.388 ± 3.272	69.2 %
2	30.722 ± 5.425	72.3 %
3	26.182 ± 2.088	61.6 %
4	23.477 ± 4.133	55.2 %
5	17.724 ± 2.086	41.7 %
6	23.805 ± 4.388	56.0 %
7	24.475 ± 2.929	57.6 %
8	17.375 ± 4.220	40.9 %

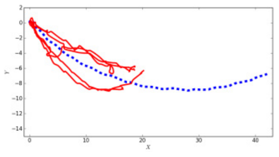
Experimental results show that the controller newly generated can recover the original performance about 40%-70%. It is revealed that severe upper joint failure (40%~57% recovered) is more difficult to be recovered than lower joint disconnection (55%~70% recovered). Upper joint is more important than lower joint because it result in losing entire leg. However, lower joint disconnection affects only half of leg. Fig. 6 shows trajectories of each condition.

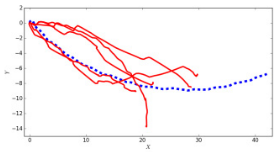

(a) Joint 1 broken

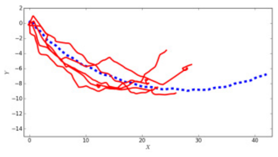

(b) Joint 2 broken


(c) Joint 3 broken


(d) Joint 4 broken


(e) Joint 5 broken


(f) Joint 6 broken


(g) Joint 7 broken

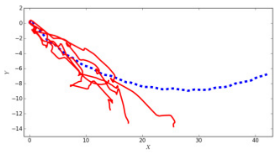

(h) Joint 8 broken

Fig. 6. Trajectories of new controller with the damaged body (dotted line is for the original intact trajectory)

Fig. 7 shows robot's movement over time. At first, one leg touches down and corresponding leg's motor pushes the ground. The leg in the opposite side touches down at the next time. Undamaged robot leg can push the ground to perform the gallop movement. If lower joint is disconnected, then left half of leg can touch down into ground and push back. It looks like the robot tries to imitate undamaged leg's functionality using the damaged leg. It is similar to human's crutches. However, when the upper joint is disconnected, the robot cannot push the ground, so it cannot imitate the gallop movement

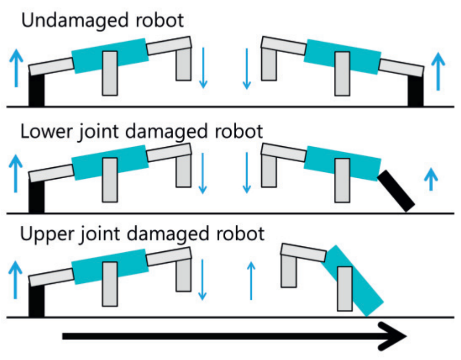


Fig. 7. The analysis of the behaviors of the robots with different levels of damages

There is another insight observable from the results. In the robot's evolved movement, a specific part is more important than other parts to perform behaviors. In this experiment, robot has "go forward" intention and the evolved robot actually shows big left curve. In this condition, some legs are more important than others. If performance recovery ratio is lower than others, it means that the part is more important than others.

When the lower joint 1~4 are broken, joint 3 (61.6 % recovered) and joint 4 (55.2 % recovered) are more important than the joint 1 (69.2% recovered) and joint 2 (72.3 % recovered) for the robot's current behavior. Because rear legs push the ground to move body forward, the damage on the rear leg reduces the moving force into the forwarding direction.

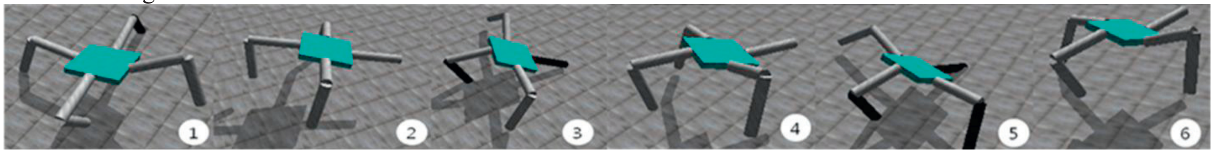


Fig. 8. Fault recovery steps (step 1: intact body, step 2: losing the right lower joint, step 3-step 6: the new behaviors for the changed body)

Fig. 8 shows the fault recovery process step by step. The step 1 shows the robot without damage and uses whole body to move forward. In the step 2, the robot loses his right lower leg. As a result, its original controller fails to produce successful behavior to go forward. In the step 3, the robot tries to find a new controller suitable for the damage body. The step 3~6 show a new body movement evolved on the damaged one. It uses the broken leg as a leverage to move forward.

5. Conclusion and future work

In this paper, we show that four-legged robot simulated in a physics-based simulator (Open Dynamics Engine) can be robust to damages on the joints of robot's body. The parallel genetic algorithm is adopted to accelerate the speed of the evolutionary search. It allows the rapid adaptation of robot's behavior using multiple remote resources. Robots continuously monitor its behavior and detect the faults. In case of the fault detected, the robot tries to adapt its behavior to the changed situation. It is rarely true that the controller originally designed for the ideal situation could work as the same way with the big changes.

It shows that the evolution can be accelerated about 6-7 times than the normal settings and the adaptation can recover the original performance about 40%-70%. From the different recovery ratio, it is possible to guess the relative importance of each part of the robot to generate the original behavior. This insight is useful to predict the robot's fault-tolerance on the different types of damages.

It is necessary to improve the way to detect the robot's broken part. Although robot has the vision sensor, it is expensive to monitor whole body's status. If the robot's part has different relevance to the behaviors currently used, it is interesting to co-evolve the basic behavior and the body topology robust to the damages.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2013R1A2A2A01016589, 2010-0018950).

References

[1] A. Ko, and H. Y. K. Lau, "Robot assisted emergency search and rescue system with a wireless sensor network," International Journal of Advanced Science and Technology, vol. 3, pp. 69-78, February 2009.
[2] R. Isermann, Fault-Diagonosis Systems, An Introduction from Fault Detection to Fault Tolerant, Spring, 2005.
[3] J. Bongard, V. Zykov and H. Lipson, "Resilient machines through continuous self-modeling," Science, vol. 314, pp. 1118-1121, November 2006.
[4] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.
[5] S. Goyal, R. Gupta, "Optimization of fidelity with adaptive genetic watermarking algorithm using tournament selection," International Journal of Advanced Science and Technology, vol. 30, pp. 55-65, May 2011.
[6] A. Brezulianu, M. Fira and L. Fira, "A genetic algorithm approach for a constrained employee scheduling problem as applied to employees at mall type shops," International Journal of Advanced Science and Technology, vol. 14, pp. 1-14, January 2010.
[7] D. Cliff, P. Husbands, and I. Harvey, "Explorations in Evolutionary Robotics," Adaptive Behavior, vol. 2, no. 1, pp. 73-110, June 1993.
[8] S. Nolfi, and D. Floreano, Evolutionary Robotics: the biology, intelligence, and technology of self-organizing machines, MIT press, 2004.
[9] J. A. Meyer, P. Husbands, and I. Harvey, Evolutionary robotics: a survey of applications and problems, Springer, pp. 1-21, 1998.
[10] H. Lipson, J. Bongard, V. Zykov, and E. Malone, "Evolutionary robotics for legged machines: from simulation to physical reality," Proceedings of the 9th international Conference on Intelligent Autonomous System, pp. 11-18, 2006.
[11] D. Floreano and L. Keller, "Evolution of adaptive behavior in robots by means of Darwinian selection," PLoS Biology, vol. 8, no.1, doi:10.1371/journal.pbio.1000292, January 2010.
[12] G. A. Sena, D. Megherbi, and G. Isern, "Implementation of a parallel genetic algorithm on a cluster of workstations: traveling sales problem, a case study," Future Generation Computer Systems, vol. 17, no. 4, pp. 477-488, January 2001.
[13] P. Pospichal, and J. Jaros, "GPU-based acceleration of the genetic algorithm," GECCO competition, 2009.
[14] Open Dynamics Engine, http://www.ode.org/ .
[15] Ludobot simulation, http://www.uvm.edu/~ludobots/ .
[16] E. S. Harris, Horse Gaits, Balance and Movements, Howell Book House, 1993.
[17] V. Vanovschi, Parallel Python Software, http://www.parallelpython.com/.