BAST: A biological evolutionary autonomous robot

Cassiano Silva de Campes
College of Information and Communication Engineering
Electrical and Computer Engineering
Sungkyunkwan University
Suwon, South Korea

Abstract

The Biological Evolutionary Autonomous System Technology - **BEAST** - is a robot that uses Genetic Algorithm to learn by itself how to walk in any circumstances. **BEAST** applies advanced techniques to divide the tasks into three different groups: (1) search, (2) execute, and (3) learn. It uses Genetic Algorithms to mimic the human behavior in learning process, allowing the quadruped robot to walk in an unknown environment iteratively. **BEAST** uses an *Client-Server* approach to reduce the computation requirements on the robot side. Future work is envisioned for using different techniques that could result in a fast learning process as well as less computational power necessary to drive the calculations for the local optimal result.

1 Introduction

1.1 Genetic Algorithms

In 1992, Holland [4] introduced the stochastic search technique called Genetic Algorithm (GA). GA was inspired by biology where the evolutionary theory persists and thus, every new adaptation generates a possibly stronger model by the natural selection. Holland could model complex computational problems into *genes* (a representation of the problem in a computing system) and thus could prove that by techniques inspired by genomes, such as crossover, he could prove that GA is a good candidate to find good solutions in a feasible time [1].

Genetic Algorithms are based on the natural selection, an observation introduced by Darwin [2] where the species are naturally selected based on its survival characteristics, and thus, new species are produced, what leads to a competition, natural selection and survival. It is a well known cycle that has been studied by him. Figure 1 presents the natural selection mechanism.

Furthermore, with the evolution in computing power with high performance parallel CPUs added to high capacity main memories, GA became a preferred approach to solve complex systems that in the past were considered impossible to be solved within a lifetime. GA are massively applied to generate high-quality solutions for problem optimization, such as path planning, and search problems. It relies on biological inspirations where the possible operations in a GA are *mutation*, *crossover* and *selection* [3].

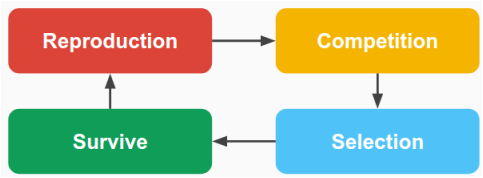

Figure 1: Darwinian natural selection theorem

1.2 Problem Modeling

One of the major challenges about applying GA to real world is how to model the problem domain in a computational environment. In our approach, we model the gene as a given command that is going to be performed on the **BEAST** robot. A gene chain is composed by several commands that are going to be executed by the robot. Each command is independent itself, what makes it easy to be modeled and applied in an GA.

2 BEAST: the GA-based robot

This section we explain the design and implementation of **BEAST**, our GA-based autonomous robot. It is divided into *Client* (Subsection 2.1) and *Server* (Subsection 2.2) sides where the former is the robot itself that executes the commands, and the last is in charge of dealing the heavy computation, respectively.

2.1 Client-side: the **BEAST**

The *Client*-side is considered a passive module, that is, it executes the commands according to the requests from server and once executed, return the results back to the server. The **BEAST** (presented in Figure 2) is a four-limb robot that uses an ultrasound module to detect distance from a reference point. The details are explained in subsection 2.3. The communication between *Client* and the *Server* is through serial connection using a baud-rate of 9600 bits per second. This is a limitation in our project, however in section 4, we evaluate the possibility of using wireless communication to avoid cables connecting the robot to *Server*.

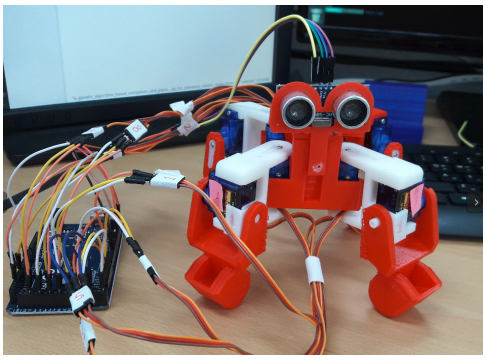

Figure 2: **BEAST**: the developed robot

2.1.1 Arduino Mega

To control the hardware and develop the necessary software to control the robot, we used the Arduino Mega development board. Arduino - an open source development board - was firstly introduced in 2005. It was based on the Atmel AVR 8-bit micro-controller, and currently the board ranges from 8-bits to 32-bits data/instructions size according to the applications needs. The board is simple and powerful at the same time, allowing the fast prototyping and evaluation of the feasibility of hardware and software projects in embedded systems industry. More informations about the board used can be found at `https://goo.gl/acJ2zw`.

2.2 Server-side: Genetic Algorithms

Our genetic algorithm is based on `best genes crossover`, where the half best genes are selected to suffer crossover with the half worst genes and finally a probability function is assigned to generate random mutations. Thus it is possible to create new generations that mixes good and bad genes. Following, the remainder half worst genes perform crossover among themselves to generate the new half genes to complete our full next generation. The new generation is created as follows:

1. generate the first generation with n genes
2. execute the genes to get the fitness;
3. sort genes from the best to the worst fitness values
4. select the half best genes and perform crossover with the half worst genes
5. select the remainder half worst genes and perform crossover among them
6. new generation is created

The pseudo-code on listing 1 explains how we developed the above mentioned steps to implement our GA to be used in **BEAST** robot. The complete source codes and references for this project can be found at `https://goo.gl/L9KqLA`.

```
1
2  new_generation() {
3  cross_idx = random()
4
5  int idx_gene
6
7  for (begin_best && begin_worst until end_best && end_worst)
8  if (idx_gene < cross_idx)
9  best[idx_gene] = worst[idx_gene]
10 else
11 worst[idx_gene] = best[idx_gene]
12 end_for
13 idx_gene = 0
14 for (begin_worst until end_worst)
15 if (idx_gene < cross_idx)
16 worst_begin[idx_gene] = worst_end[idx_gene]
17 else
18 worst_end[idx_gene] = worst_begin[idx_gene]
19 end_for
20
21 send_to_robot()
22 }
23
24 sort_genes() {
25 int i, o
26 for (begin_gene until end_gene -1)
27 for (begin_gene until end_gene - (idx -1))
28 if (fitness[idx] > fitness[idx + 1]) {
29 int t = fitness[idx]
30 fitness[idx] = fitness[idx + 1]
31 fitness[idx = 1] = t
32 }
33 end_for
34 end_for
35 }
36
37 store_fitness(int genes_idx, int fit_val) {
38 fitness_list[idx] = fit_val
39 }
```

Listing 1: Pseudo Genetic Algorithm

Function `new_generation()` in listing 1 is executed on the *Server* side due to the heavy computational needs. Once the new genes are generated, they are sent to the **BEAST** robot through the serial communication interface. Thus, the robot is able to execute the new commands, generating the fitness values, and sending it back to the *Server*. Our approach exhaustively use the *Client-Server* communication to avoid heavy computations on the robot side, reducing complexity.

Figure 3 shows the block diagram that explains how the Genetic Algorithm is executed in *Server* and *Client* sides. The heavy computation is performed in the *Server*

by generating the first generation, and then sending the genomes to the *Client*. The *Client* in turn executes the commands, and stores the fitness and sends it to the *Server*. The fitness is used by the Server to decide the best genomes to be used for crossover operation. A mutation of 1% is configured in our algorithm to give randomness possibility. After the new generation is generated, the genomes are sent to the *Client*, and the algorithm repeats until it reaches the maximum number of iterations configured.

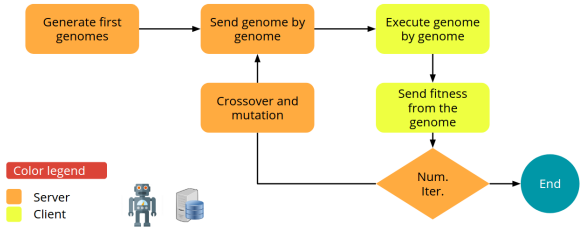

Figure 3: Block diagram representing the communication between *Server* and *Client*

2.3 Fitness Parameter

To evaluate the performance of the **BEAST** robot we used the distance measurement as the fitness value to evaluate the genes generated by the GA. However, we have noticed that depending on the robot position and rotation angle, the distance may be interfered. To solve this problem, we also use a gyroscope to give more measurement precision to our fitness.

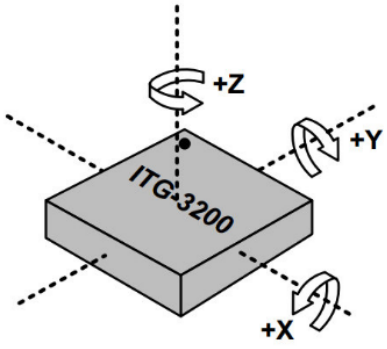

Figure 4: 3-axis gyroscope is used in this project

Figure 4 presents the 3-axis that is used to get the rotational information of the robot. The better aligned the robot is, the better is the fitness. This way, we have our fitness function based on both distance and gyroscope information. Every gene stores a command that is executed by the robot and thus, it may result in a random movement that can either reduce or increase the distance between the robot and the target reference. This way, we denote the fitness value as the average distance traveled by the robot (distance difference) to the reference point [5].

At runtime, when the robot is learning how to walk by executing the genes received from the *Server*, the fitness is calculated after each step and later the average value is sent back to the *Server*. The distance is measured using the ultrasonic module HC-SR04 sensor (more informations are available at `https://goo.gl/PRMjTe`). Figure 5 shows how the distance influence the fitness value.

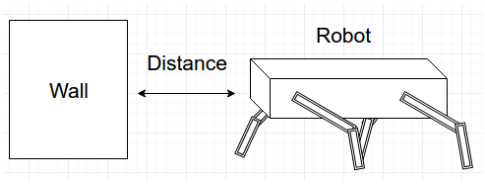

Figure 5: Fitness as a function of distance

Although at this point of the research we were not able to provide a feasible fitness simulation to increase the result speed, our results satisfies the expectations according to the available resources. We could provide good results in a reasonable execution time even when we need the robot to execute all genes generated.

3 Evaluation

We implemented the proposed project based on a *Client* and an *Server* architecture as explained in Section 2. The Server, which is in charge of the heavy computing processing, is an Intel Xeon E3-1270 3.6GHz Octa-core 64-bit Processor equipped with 32GB main memory. The robot parts were printed using the 3D printer Cubicon located on the Learning Factory Laboratory at Sungkyunkwan University. The robot has four limbs and each limb contains 2 micro-servo motors acting as the joints, resulting in a total of eight step motors connected to an Arduino Mega micro-controller. The control of the motors is performed using Pulse Width Modulation(PWM) to precisely infer in the rotation of the limbs.

Figure 6 shows the prototyping circuitry necessary for our project. For this implementation, we were not able to develop wireless communication and thus, the communication is done through serial signals to the *Server*. This resulted in several wires to connect the parts of the robot. Figure 7 shows the circuitry schematic used. The connections are carefully placed. Other option could be the use of Arduino Nano, however, this board does not have enough PWM signals (only 6) to control all the necessary servo motors in our project. It was inevitably to use Arduino Mega to accomplish our goals.

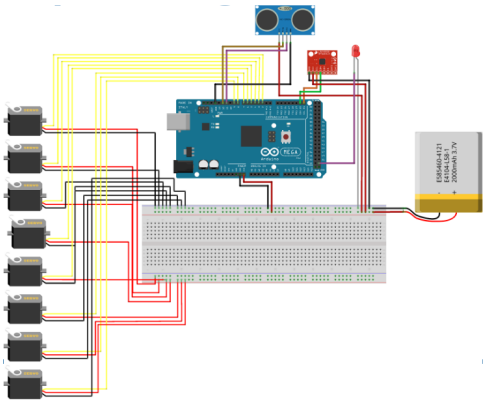

Figure 6: Components organization used for the project

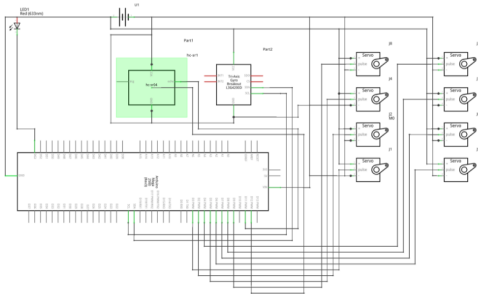

Figure 7: Circuit schematic for the robot

To evaluate our robot we then executed all the necessary GA algorithm, were the communication between the *Client* and *Server* were performed. Every iteration the robot sends the fitness values to the *Server*, and thus the *Server* generates the new genes to be executed on the *Client*. Once the number of iterations reach the maximum configured value, the *Server* then stores the best gene reached so far, send it to the *Client*, and then the *Client* executes the best gene in a infinite loop.

As explained, we added a gyroscope to give to the robot more degrees of freedom to create a powerful fitness function. For this reason, the fitness is the sum of distance and the gyroscope information. At the initialization phase of the robot, the gyroscope is calibrated and then, the calibrated position is considered the position zero in al three axes *X, Y*, and *Z*. This way, during the algorithm execution, the difference between the actual axis values from the reference value will be counted for the fitness value. Thus, as less rotation variation from the reference, better is the fitness.

We plot a graph that is the result of the GA iterated several times. The X-axis represents the number of iterations executed and Y-axis is the average fitness value. As we can expect, as the number of iterations increase, better results we obtain. However, due to our project being executed in a real hardware, we have suffered from slow

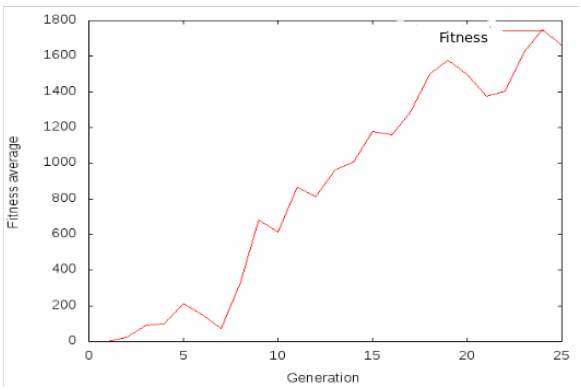

Figure 8: **BEAST** fitness evaluation

execution time due to the motors rotation that has a time limitation.

4 Conclusion

Very complex engineering problems are very hard to be solved due to required huge computing performance and enormous execution time. Evolutionary systems are a good approach to find reasonable optimized solutions (but not always the optimal) for such problems. Genetic algorithms are a good example of such approach, where inspiration by biology enforces the concept of genes that are related to perform a given task. The genes are crossover-ed and mutated to produce new generations iteratively and thus, trying to reach a best solution.

Our **BEAST** robot uses such advanced techniques in problem optimization by using genetic algorithms to find a solution in its domain. Our experiments have demonstrated that the robot is able to walk within few iterations. Nonetheless, we also demonstrated that after several iterations, the **BEAST** learned to walk in a admissible fashion (similar to quadrupeds animals).

Finally, we could conclude in a real system the usage of genetic algorithms applied to a quadruped robot. Although our approach uses serial connection (that needs a wired connection to the robot), we expect to improve the interface to a wireless mechanism, allowing the robot to be fully autonomous. Further work is considered to use different GA approaches to enhance the processing time. Nonetheless, we expect to implement a fitness simulator based on real experiments, this way we can simulate all the movements computationally and later transfer the results to the robot to be executed on real platform.

References

[1] AHUACTZIN, J., TALBI, E.-G., BESSIERE, P., AND MAZER, E. Using genetic algorithms for robot motion planning. *Geometric Reasoning for Perception and Action* (1993), 84–93.

[2] DARWIN, C. On the orngin of species, 1859.

[3] EIBEN, A. E., RAUE, P.-E., AND RUTTKAY, Z. Genetic algorithms with multi-parent recombination. In *International Conference on Parallel Problem Solving from Nature* (1994), Springer, pp. 78–87.

[4] HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[5] LIPPMANN, R. An introduction to computing with neural nets. *IEEE Assp magazine 4*, 2 (1987), 4–22.