Evolving ice skating behavior with a Nao robot in a simulated environment using HyperNEAT

Bas Bootsma (0719080) Roland Meertens (3009653)
Tom de Ruijter (3016269)

January 22, 2012

1 Introduction

...

Literature background...

In section 2 *Initial Experimental Setup* most of the details of the project will be explained and our .... In section ...

2 Initial Experimental Setup

2.1 NEAT

NeuroEvolution of Augmenting Topologies (NEAT) technique developed Ken O. Stanley in 2002, which makes use of genetic algorithms for evolving artificial neural networks [3]. The technique both alters the weights and the structure (topologies) of the network to find a balance between the fitness and the diversity of the solutions. NEAT has several properties which make it an interesting technique.

- **Complexifying**: Complexifying is the incremental increase of complexity over time. This means that NEAT starts out with simple topologies and gradually increases complexity over generations.
- **Speciation**: With speciation topologies which are similar are grouped together in niches. This protects for example new topologies, which generaly have worse fitness when compared to already existing topologies, thus allowing the new topologies evolve in a protected niche instead of immediatly discarding them.
- **Historical marking**: Each gene is assigned a unique historical marker which allows later generations to track origin of their genes.

2.2 HyperNEAT

HyperNEAT is the successor of NEAT and provides several extra properties which we believe are beneficial for generating ice skating behavior.

- **Geometric relationships**: HyperNEAT is aware of the geometric relationships of the input. In our case this means that it not only knows the different joints, but also the relationships between the joints. For example, it might be able to exploit the relationship between the ankle and knee joint by posing extra restrictions to prevent the Nao robot from falling over.
- **Symmetrical and recurrent patterns**: HyperNEAT is able to find symmetrical and recurrent patterns. In our case this is important, since ice skating behavior contains symmetrical and recurrent elements, which we hope HyperNEAT is able to find.

3 Hypothetical Experimental Setup

In the previous section we have explained the major details of our project and we also have explained how we did not get HyperNEAT to work successfully. In the case that we did got HyperNEAT working successfully we would have liked to perform several experiments. In the following sections the hypothetical setup, the experiments and the related HyperNEAT configurations are described (and the related expected results).

3.1 Setup

As we already explained we would have wanted to use a different approach, namely using a direct feedback loop from the simulator to the HyperNEAT algorithm. This means that for each time step t the current joint values of the robot in the simulator are read and feed as input into the HyperNEAT algorithm which produces the joint values for time step $t + 1$. Trying to evolve a complete behaviour (i.e. a motion file) has deemed to prove too difficult. However the described approach allows for a more direct mapping between the evolving behavior and the HyperNEAT algorithm in which the problem can be solved in smaller steps. A similar approach was used in evolving a walking gait for a quadruped using generative enconding [2].

* Able to use the scaling abilities of HyperNEAT; * Something about how long (settings) each run would take, etc.

3.2 Experiments

In order to investige our research question, whether it is possible to learn ice skating behavior to a Nao robot in a simulated environment, we would have to perform several experiments. As we have already explained ice skating behaviour consists of two phases, namely the startup phase and the recurrent phase. Since we are primarely interested in the recurrent phase all the experiments focus on trying to evolve an behaviour which has the property of a recurrent motion. This means that any evolved behavior not only has to work for the duration in which the algorithm runs, but also for durations larger than what is being evolved for. ...

3.2.1 Condition: With Initial Velocity

For the baseline condition we are interested in the recurrent phase of an ice skating behavior. Therefor we want to investigate whether it is possible to evolve an ice skating behavior from a

standing position with an initial velocity. This means that the Nao is in the initial position (i.e. all the joints are initially at 0) and a specific force has been applied to the robot which results in a certain velocity.

@TODO: THis probably means that the robot only has too small movements...

3.2.2 Condition: From Resting Position

If the evolved behaviour from the condition with an initial velocity for the Nao robot, works out out well, we want to investigate whether it is possible to evolve a similar behaviour from a resting position. In this condition the initial position of the Nao robot is the same as the condition with the initial velocity except no velocity is given. This means that the algorithm has to evolve a behaviour for both phases. This condition will probably take much more computational time to

In the case that both previous conditions produced no suitable ice skating motions there was also a different approach we could have taken. That is adding an already evolved behaviour to the popuplation. In this case there were two behaviours possible, namely a standard walking behaviour and a human ice skating behaviour. Even if both previous conditions did produce suitable ice skating motions it would still have been interesting to investigate whether adding such behaviours by default to the population would help to evolve a suitable ice skating behaviour.

3.2.3 Condition: Walking Behaviour

By default a walking motion is already provided by Webots containing the exact joint values at each time step. We could have used the motion file to train the algorithm with a fitness function based on how much the output differs from the already known joint values. After a certain number of generations or until the algorithm has perfectly learning the motion we could then save the CPPN and ... load it in

3.2.4 Condition: Human Ice Skating Behaviour From a Kinect

3.3 Fitness Functions

Predictions about the experiments...

3.4 HyperNEAT Configurations

* Close to the original ...

3.5 Expected Results

4 Proof of Concept

5 Introduction

As described in section 2 hyperNEAT does not generate any suitable values. As a way of demonstrating that (except for the neuro evolution algorithm algorithm) our implemented setup works an alternative evolutionary algorithm is implemented. The fitness functions described in section 2 are implemented in Java, which brought the choice of our evolutionary algorithm to the easily usable and implementable Genetic Algorithms and Genetic Programming component (JGap).

JGap is provided as a Java framework and provides basic genetic mechanisms that can be easily used to apply evolutionary principles to problem solutions [1]. JGap is free of charge and can be downloaded at http://jgap.sourceforge.net/.

As described in section 3.1 the representation the motion of our Nao robot is represented as a two-dimensional array with double values. As JGap works with a genome consisting of an array of alleles consisting of double values a simple algorithm is written to converge the one-dimensional output of JGap into a two-dimensional array.

During the generating of the next generation of chromosomes the existing chromosomes are crossed over. It is very probable that a chromosome generates a motion in which the first half is a very suitable motion, but leads to the Nao falling over halfway during this motion. This leads to our choice of the single-point crossover parameter which randomly selects a point to start the crossover.

Continuously generating new generations of chromosomes by crossing over leads to a very small amount of alleles surviving and leads to sparsity of diversity in the population. Due to this a mutation parameter is introduced. The downside of mutation is that a mutated chromosome is more likely to introduce a wrong parameter at the start of a file. This makes mutation dangerous and leads to our choice of mutating only 1/12 of chromosomes.

After each generation the chromosomes used in the next generation are selected. As there is not one definite motion the algorithm has to converge to even sub-optimal chromosomes have to survive. Our algorithm selects the best 90 percent of chromosomes each generation and uses these to generate the next generation. These chromosomes are manipulated, and the best chromosome is preserved to prevent it from dying after all manipulations.

There are a lot of different motions that our Nao can perform to achieve a forwards motion. This means that even a motion that is not ideal is able to evolve into a motion that has a high fitness. With this in mind, ideally the population size is very large, as this gives more way to innovative solutions. Unfortunately the computation time of the fitness function is very large, as described in section 3.3. To speed up the simulation a population size of only 30 is chosen.

As there is no clear goal that our Nao has to reach while moving forward and our algorithm is not able to run forever an amount of maximum generations is selected as termination condition. As there was not a lot of time to run our algorithm the termination condition taken was 200 generations.

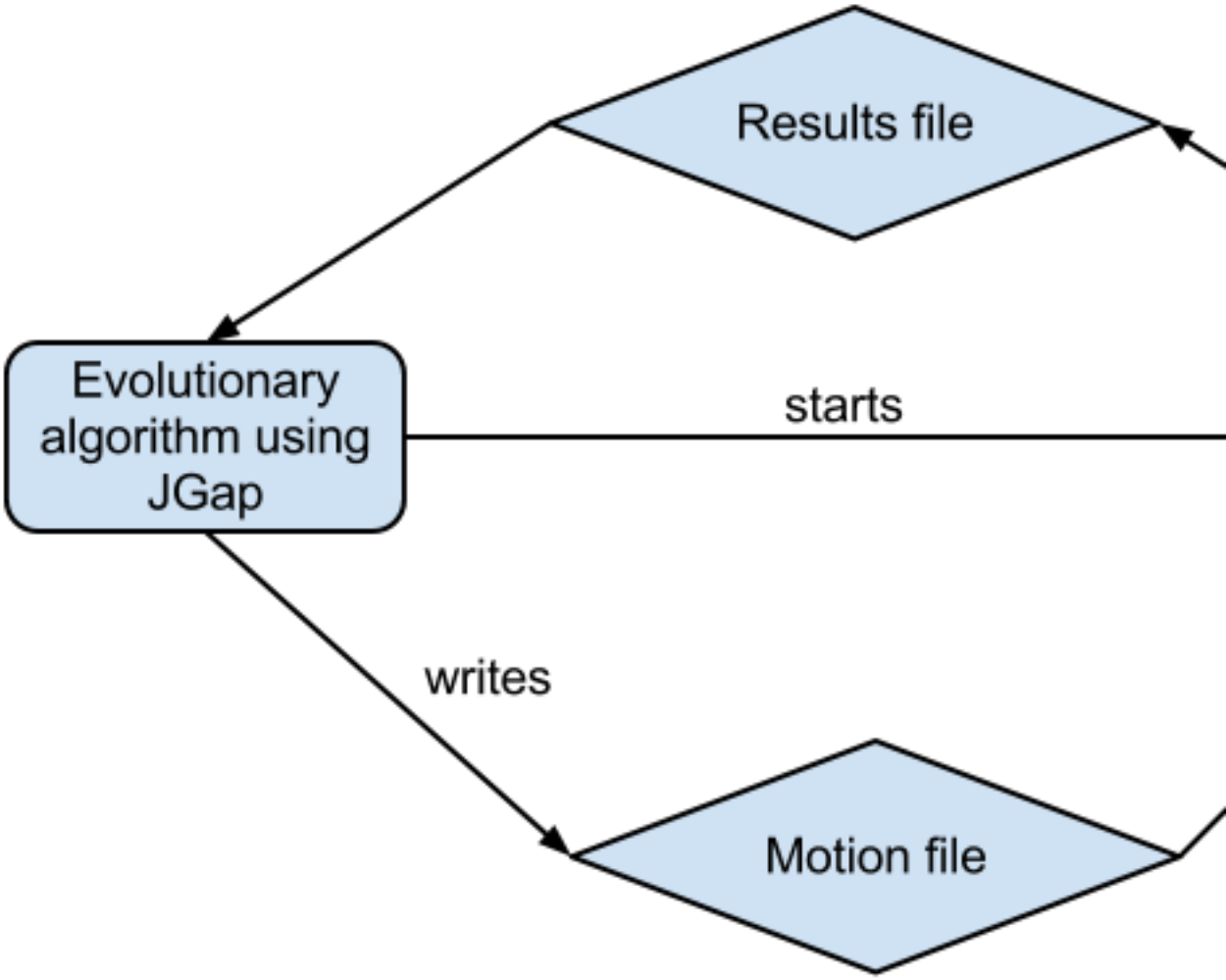

Figure 1: Setup used for running our evolutionary algorithm

6 Analysis of the results

6.1 Arm moving

-¿ Graph-like thing that describes how the fitness improves over the different generations

During the running of the evolutionary algorithm with the arm moving fitness function no real milestones can be found. The algorithm starts with moving its arm randomly, and keeps selecting a better arm-raising motion every generation.

Unfortunately our final chromosome still has values that are terribly off compared to the ideal values. Apart from this problem the final result looks like a very solid arm movement. The conclusion of our arm moving fitness function is that evolutionary algorithms are very capable of producing the simple movement of one joint.

6.2 Moving forwards for four seconds

-¿ Graph-like thing that describes how the fitness improves over the different generations

In the first five generations the evolved behaviours consist of our Nao performing some random movement which leads to a strange forwards motion. Starting from generation six the algorithm develops a rolling motion that ends at the four second mark. As our distance-measure point is the middle of our robot this makes a lot of sense. By performing a roll a little bit of extra distance is gained during the four seconds in which the motion is performed. Starting at iteration 120 a new behaviour emerges: sidewards shuffeling. This behaviour proves to be the best reachable behaviour in 200 generations as no better behaviour is found.

Although our best chromosome after 200 generations is capable of traversing a large distance in four seconds it does not look like a very stable walking gait. Evolutionary algorithms thus are very capable of producing an advanced movement like this when they have enough time. The other conclusion is that the produced gait is not stable enough to use in a real robot, it thus is necessary to improve our fitness function.

6.3 Repeated motion for twelve seconds

-¿ Graph-like thing that describes how the fitness improves over the different generations

In the first five generations the evolved behaviours consist of our Nao performing some random movement leading to eventually falling on its back. In generation six our algorithm discovers that by turning around at the start of the motion it gains a little more distance (as the measure point for distance is the middle of our robot). In the same generation it starts to walk by using its arms as support (just like a gorilla). It also generates a motion that prevents the Nao from falling, consecutive generations improve the motion and keep preventing our Nao from tumbling over (except for generation 60).

Although our best chomosome after 200 iterations is capable of traversing a large distance in twelve seconds by repeating the four seconds long motion three times it unfortunately does not look like a very stable gait. Evolutionary algorithms thus are also very capable of producing an advanced repeated movement like this when they have enough time. Just like the fitness function

of the four second long motion (in section 6.2 this motion is not stable enough to use in a real robot and our fitness function has to be improved.

6.4 Ice skating

-¿ Graph-like thing that describes how the fitness improves over the different generations -¿ Describing the steps in the evolutions -¿ show that even the final results can be improved As our algorithm does not converge to an ice-skating motion the only conclusion is that our fitness function has to be adjusted to account for more than just the accelerometer data.

7 General conclusion for all results

As shown in the previous section (section 6) our setup is able to generate very nice behaviours as long as the fitness function is chosen correctly. Especially our arm moving fitness (section 6.1) shows that the generated behaviours still are not optimal as the generated values are off from their ideal values. This problem can probably be easily solved by having a larger population and more generations, which is not possible in a small computational time of less than a day.

8 Conclusion

References

[1] The jgap website, Januari 2013. http://jgap.sourceforge.net/.

[2] J. Clune, B.E. Beckmann, C. Ofria, and R.T. Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2764 –2771, may 2009.

[3] Wikipedia. Neuroevolution of augmenting topologies. `http://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies`, 2012. [Online; last accessed 16-October-2012].