# Learning optimal snake robot movement via evolutionary algorithms

**Zeyad Alghamdi, Calvin Ferraro, James Yu**

ASU CIDSE
zai@asu.edu, csferrar@asu.edu , Jamesyu1@asu.edu

**Abstract**

Snakes are an unconventional choice for robots because their movement is not easy to model. Due to the difficulty in their movements, it would be simplest to design a machine learning algorithm that would develop behaviors on its own. Thus, genetic algorithms are a good option for generating behaviors. A genetic algorithm only requires a method of encoding desired behaviors into a chromosome and a method of quantitatively determining how good a behavior actually is. Ideally, the end behavior in this work will be one that is already seen in nature, since genetic algorithms are derived from Darwin's theory of evolution.

## Introduction

It has been extensively studied by herpetologists that there are over 3500 discovered snake species around the world [1]. We thought that it would be an interesting field of work to explore how the results of an evolutionary algorithm compare to the results of biological evolution in the real world. As we can see Figure 1, common snake movements are broken down into four categories, each with their own movement pattern. One of the more interesting movement types is called sidewinding locomotion. The sidewinding movement is described as basically anchoring the body around the head, tail, and middle part, having a shape like a sine wave or the letter S. These three anchor points help the snake walk itself forward. We want to show how we can use Machine learning to explore if one of these four categories of motion is really the best kind of motion for a snake.

## Methodology

The simulations in this work were done using VREP. Included in the VREP package is a prebuilt snake mobile robot containing four links and a camera. The control for the snake was encoded into variable parameters described in the *Physics Parameters and Sinusoidal Waves* section of this summary, and the values for these parameters are obtained from the Genetic Algorithm described after that. Each snake in a population was simulated in VREP sequentially, and each population, organized into generations, was simulated sequentially as well. After each snake was simulated, its movement along the Y-axis was recorded as its fitness value. The Y-axis is used because when the robot is placed in the environment, the default direction that the robot is facing is the Y direction. The program included in the project repository simulates the snakes for a set number of generations with a set population size.

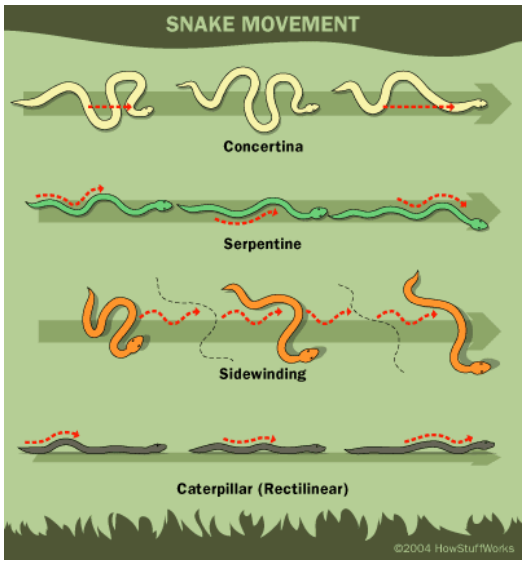


Figure 1: Common snakes movements categories[2]

VREP natively uses Lua to control its robots. However, since Lua is not a very common language, there is a remote API included that allows more common languages to run VREp simulations remotely. We used a remote Python API to send the commands that we wanted to run to VREP's local server. These commands included actions such as setting the angle for each joint of the snake and resetting the environment so that the snake returns to its initial pose once the individual has completed its simulation.

### Physics Parameters and Sinusoidal Waves

Upon observing real world snakes, we found that snakes move using special muscle groups, but in the field of robotics, we need to use discrete joints and links instead of continuous muscles. For every joint we have a vertical and a horizontal movement function, denoted as VJF and HJF respectively below. An illustration of the functions is shown in Figure 2.

$$VJF = AV * sin(S * [t + PV]) \quad (1)$$

$$HJF = AH * cos(S * [t + PH]) \quad (2)$$

In Equations 1 and 2 above, we have the standard sinusoidal wave equations, one sine and one cosine. The amplitude (AV, AH) determines the magnitude of the wave. This can be seen in Figure 2 by the difference between the maximum value and the zero value of the waves. These variables are named respectively such that AV is the vertical amplitude and AH is the horizontal amplitude. The speed *S* represents the frequency of the wave. Frequency represents the number of times a wave passes a fixed point in each second. The time *t*, in terms of the mathematical model, represents the distance along the wave, or the time that has passed since the wave was in its zero position. In terms of our snake domain, this represents the time that has passed in the simulation. The final variables in our equations represent the phase (PV, PH) which are named respectively for the vertical and horizontal phases. Phase represents the initial offset of the wave in the time direction. This only modifies the starting position of the wave and does not affect any other behaviors of it as time *t* approaches infinity.

There is one more variable that is used in the simulations that does not appear in the mathematical model that we have described thus far. This is referred to as the camera phase PC. The camera is mounted on the front of the first link and is given its own phase variable because it only has one degree of freedom as opposed to the two degrees that the other joints have.

## Genetic algorithm

The snake movement domain is the perfect space for a Genetic Algorithm. It is not readily apparent how a snake should move in order to create an optimal movement, and, judging by nature, there may not be one optimal movement for every snake. However it is very easy to judge the fitness of any given snake. We decided to go with a fairly standard algorithm for our Genetic Algorithm for the sake of simplicity and ease of implementation. To relate the parts of the Genetic Algorithm to the snake movement, the chromosome is represented by a list of floating point numbers that are the parameters in Equations 1 and 2, and the fitness is the forward distance travelled by the snake. Each population is simulated by simulating the individuals sequentially.

Our implementation is made up of two parts, the Individual class, and the Genetic Algorithm class. The Individual represents one snake by containing a python list that represents the chromosome of the Individual and a float that represents the value assigned as the fitness of the

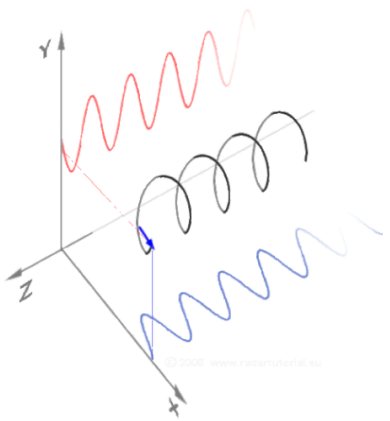


Figure 2: This set of graphs shows the three dimensional sinusoidal curve and its sine and cosine projections onto two dimensional planes [3]

Individual. The Individual class also contains a method to mutate the Individuals chromosome. This works by taking in a threshold value called the mutation rate. This is a value from zero to one that is compared to a random value, also from zero to one. If the random value is greater than the mutation rate, then the chromosome will be mutated by picking one index of the chromosome array and reassigning it to a random value within its predefined range.

The Genetic Algorithm class (GA) contains most of the logic of the algorithm. When initialized, the GA populates the first generation by creating chromosomes with randomized parameters. Each new generation is generated by creating a new population of individuals-as is typical in Genetic Algorithms. The first Individual in each generation is the best individual that has been generated up to this point. We then take the top half of the last population and cross them over repeatedly until we have enough individuals to fill out the population. This may result in population members not being crossed over or other population members being crossed over multiple times. This randomness helps the Genetic Algorithm reflect its roots in biological evolution as this is how it may occur in biology. Individuals are crossed over by choosing parameters from each of the parents. Each parameter has a 0.5 chance that it will be passed on to the newly crossed over individual. After the individual is crossed over, it receives the chance to be mutated based on the mutation rate. If mutation occurs, one parameter is taken at random and reassigned to a new random value. This allows for individuals to explore new methods of movement by having a parameter that may not have been seen in a previous generation.

## VREP Python API

The implementation for the python API was done through the provided scripts found within the VREP installation folder after first downloading VREP. Within the VREP version 3.6.2 MAC PRO EDU download folder (the location varies slightly based on operating system), there contains a script called vrep.py found within the programming/remoteApiBindings/python path which serves as the script containing the dependencies required for implementing a python api.

After importanting this vrep.py script and remoteApi.dylib file into a src, all that was left was to create the actual client script that would remotely run VREP simulations. This was achieved by creating a client script found within the same environment with the vrep.py script and remoteApi.dylib file that would connect to the VREP simulation server, obtain handler ID's for a snake robot placed in the VREP simulator environment and control the joint motors (via a joint target position method) in order to test out the fitness of an individual within a population of encodings provided by the previously mentioned genetic algorithm class.

The VREP Python API utilizes the GA class by creating an instance of the class at the beginning of execution and continuously calling its methods to get a population of encodings to be used in the algorithm for snake movement and to also update the fitness of the individuals within a population, and lastly, when all individuals within a given population have been simulated, create a new population considering mutations and crossover of the fittest snakes to begin testing a new generation of snake movement encodings.

A continuous API was desired in order to continuously restart and run multiple simulations in one script execution, so an API configuration txt file, which was also found in the VREP installation folder, was modified to have the VREP simulator serve as a server that listens to a specified socket.

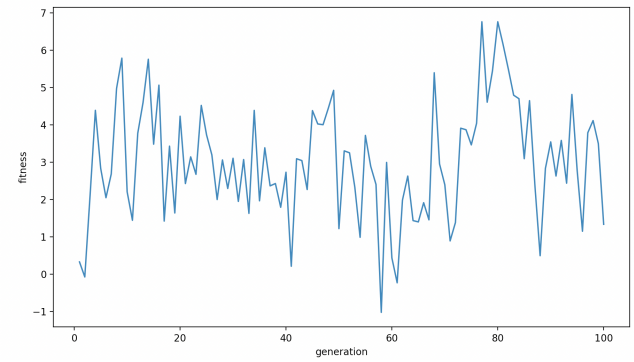


Figure 3: The average fitness of the snakes per generation for an experiment where the values for each snake movement parameter are limited between 0 and 40. The experiment ran with a 50% mutation rate with a population size of 10 over 100 generations.

## Results

The results of one of the experiments we performed for snake movement are shown below in the two figures. The average fitness of a snake over the 100 generations is shown below in Figure 3.

As shown by Figure 3, fitness seemed to move around with not much trend moving upwards. It seemed that as the generations progressed, the fitness seemed to be increasing and decreasing at a larger rate and this can be attributed to the fact that the fitness score of a snake is how straight it moved-which means that the difference between a snake doing well and doing not as well can be dictated by the direction it is moving. With some noticed inconsistencies of even running the same snake with the same parameters multiple times, the lack of overall fitness improvement may be attributed to the fact that a snake that performed well may not perform as well again by moving sideways (or even backwards) in a straight direction rather than forwards. The lack of progression seen in the average fitness for each generation can also be attributed to the mutation rate. In our experiments, we used a mutation rate of 0.5 which is quite low. Had it been higher, individuals may have been more similar to the previous generations and less susceptible to mutating into something that is not as useful.

The fitness of the best individuals at 10, 20, 50, and 100 generations over 100 generations is shown below in Figure 4.

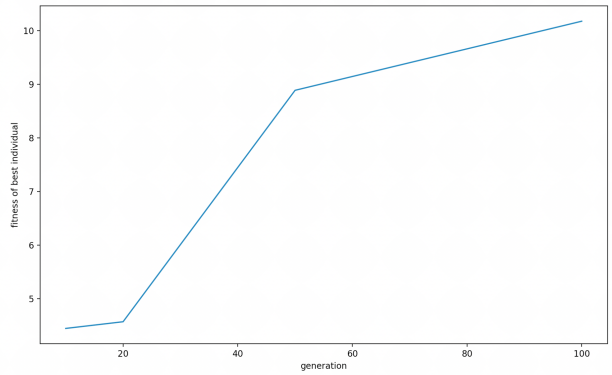


Figure 4: The fitness of the best individuals at 10, 20, 50, and 100 generations over 100 generations for an experiment where the values for each snake movement parameter are limited between 0 and 40. The experiment ran with a 50% mutation rate with a population size of 10 over 100 generations.

As shown by Figure 4, the best individuals at every generation seemed to constantly grow at an impressive rate over the generations-considering that the average fitness of snakes seemed to hover at around three. These results support the conclusion that, perhaps, the snakes were learning to move large distances of space while the

direction that it may be moving may have been inconsistant. It also supports the above claim that the average was held back by the low mutation rate.

## Lessons Learned

There were a lot of lessons learned from this group project, a lot of which involved utilizing the VREP simulation application, which is now called Coppeliasim. Since all of us were new to VREP, we all spent a few days watching youtube videos on how to make a python API for it and also what tools were contained within VREP. As we were exploring all the entities and models that were already provided by VREP, we learned just how powerful VREP is as a tool, and, after having learned how to simulate simple snake movements on VREP using the python API, we also learned just how user friendly VREP was especially considering how other simulation tools have a much higher learning curve.

There was also a lot of research that went in to actually figuring out how to integrate a genetic algorithm with the VREP simulator to simulate generations of snake movement encodings, and a lot of research also went in to understanding the math behind how a snake may move and what common snake movements are within nature in order to understand the results of our simulations and to properly simulate a robotic snake to move.

Beyond the technical aspects of the group project, there were a lot of lessons learned in working and presenting as a group as well. As a group, we learned how to delegate tasks such as generating a script for the genetic algorithm and generating APIs and running simulations in such a way that was non-intrusive to each other's tasks. Also, as a group we were forced to clearly structure our presentation, by putting time constraints on our presentation parts and working on handoffs to each other, in order to present our project and the different aspects of what went into it cohesively.

## References


*[1]. A. Kasturiratne et al., "The Global Burden of Snakebite: A Literature Analysis and Modelling Based on Regional Estimates of Envenoming and Deaths," PLoS Med, vol. 5, no. 11, p. e218, Nov. 2008, doi: 10.1371/journal.pmed.0050218*

*[2]."How Snakes Work," HowStuffWorks, Aug. 20, 2004. https://animals.howstuffworks.com/snakes/snake.htm (accessed Apr. 29, 2021).*

*[3]."7 Animated GIFs That Will Make You Instantly Understand Trigonom*etry," IFLScience. https://www.iflscience.com/physics/7-animated-gifs-that-will-make-you-instantly-understant-trigonometry/ (accessed Apr. 29, 2021).


The QR code and link below connect to the git repository containing the code for this project.

**bit.ly/3elgXBc**