Fachhochschul-Masterstudiengang
INFORMATION ENGINEERING & -MANAGEMENT
4232 Hagenberg, Austria

Evolving Aquatic Robots for Payload Transportation

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science in Engineering

Eingereicht von

Ing. René Draschwandtner MSc

Begutachter: FH-Prof. DI Dr. Stephan Winkler

August 2015

Eidesstattliche Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum Name, Unterschrift

Contents

Acknowledgements

This work was carried out during my research visit at the department of Computer Science and Engineering, Michigan State University. I want to thank the Austrian Marshall Plan Foundation for funding my visit with the *Marshall Plan Scholarship* (MPS) and the Upper Austrian Government for granting me the *Internationalization Program for Students* (IPS).

I offer my sincerest gratitude to my supervisor, Prof. Philip K. McKinley, at Michigan State University. Without his guidance, encouragement, support and inspirational ideas this study would hardly have been completed. I am deeply grateful to my colleagues in Prof. Philip K. McKinley's research group Anthony Clark and Dr. Jared Moore. Both introduced me to the field of evolutionary robotics and provided practical guidance for my research activities. My sincere thanks also goes to my advisor at University of Applied Sciences Upper Austria, FH-Prof. Stephan Winkler, for his guidance in the initial phase and the end of my research project.

Kurzfassung

In einer aquatischen Umgebung wird der Nutzlasttransport typischerweise von Tauchrobotern, die mit Steuergeräten ausgestattet sind, durchgeführt. Diese Masterarbeit untersucht Steuerstrategien für den Nutzlasttransport mit schlangenähnlichen Robotern. Selbstentwickelte und evolvierte Steuermechanismen werden verwendet, um ein effektives Roboterverhalten zu erzeugen. Der Roboter wurde nach dem Vorbild einer Wasserschlange konzipiert und besteht aus einer definierten Anzahl an starren Körpern (rigid bodies) sowie motorisierten Gelenken. Der Nutzlasttransport ist in dieser Masterarbeit wie folgt definiert: Fähigkeit des schlangenähnlichen Roboters ein kugelförmiges Zielobjekt in einer aquatischen Umgebung auszuliefern. Im Unterschied zu konventionellen Tauchrobotern, welche externe Arme und Propeller besitzen, verformt der vorgestellte Roboter lediglich seine Gestalt um vorwärts zu schwimmen und ein Objekt zu greifen. Dieser Ansatz bietet mehrere Vorteile gegenüber konventionellen Tauchrobotern. Erstens, die Wahrscheinlichkeit der Umwelt Schaden zuzufügen wird durch die Absenz von Propeller und anderen externen Geräten reduziert. Zweitens, der Roboter verfängt sich weniger leicht in umhertreibenden Gegenständen. Drittens, die Umwelt wird mit diesem Ansatz geschont, da der Roboter weniger Lärm verursacht. In dieser Masterarbeit werden alle Experimente mit Simulationen durchgeführt. Die komplexe Aufgabe des Nutzlasttransports wird in folgende Teilaufgaben untergliedert werden: (1) annähern an das Zielobjekt, (2) greifen des Zielobjektes, (3) ausliefern des Zielobjektes. Diese Teilaufgaben werden zunächst individuell untersucht und nachfolgend kombiniert um die Nutzlasttransportaufgabe durchzuführen.

In der Natur generieren Wasserschlangen einen Vorwärtsantrieb indem sie sinusoidale Wellen entlang ihres Körpers propagieren. In dieser Masterarbeit verwenden die wasserschlangenähnlichen Roboter sinusoidale Bewegungsmuster um Geschwindigkeit aufzubauen. Die Parameter der sinusoidalen Bewegungsmuster werden mit genetischen Algorithmen optimiert, um eine maximale Durchschnittsgeschwindigkeiten zu erzeugen. Zusätzlich verwendet der Lenkalgorithmus das so erzeugte Moment, um den Roboter in Richtung des Zielobjektes zu steuern. Ein entwickelter Greifalgorithmus wird ausgeführt, sobald sich der Roboter in der Nähe des Zielobjektes befindet. Sobald das Objekt gegriffen wurde, führt der Roboter erneut ein Schwimm-

/Lenkverhalten durch. Zum Schluss setzt der Roboter das Objekt innerhalb einer Zielregion ab.

Eines der Ziele dieser Masterarbeit ist die Bewertung der Effektivität von evolutionären Algorithmen zur Optimierung der Nutzlasttransportaufgabe. Viele wissenschaftliche Arbeiten im Bereich evolutionary computation optimieren eine eng definierte Aufgabe. Im Gegensatz dazu kombiniert diese Masterarbeit selbst entwickelte Lösungen, mit Ergebnissen von evolutionären Läufen um ein komplexes Verhalten zu erzeugen, dass sich aus mehreren Teilaufgaben zusammensetzt.

Abstract

Payload transportation in an aquatic environment is usually performed by underwater robots equipped with controller boards. The focus of this thesis is an investigation regarding control strategies for the payload transportation task as executed by a snake-like robot. Both engineered (hand-coded) controllers and evolved controllers are employed to produce effective robotic behaviors. The robot has been designed to resemble a water snake and comprises a number of rigid bodies (links) and actuated hinge-joints. For this work, the payload transportation task is defined as: the ability for the snake-like robot to deliver a spherical target object while operating in an aquatic environment. In contrast to conventional aquatic robots, which are are often driven by propellers and utilize a robotic arms to grasp objects, the snake-like robot swims and grasps objects by deforming the robot's body shape. This behavior is useful in the real world for several reasons. First, the absence of propellers and external devices will reduce the likelihood of damaging the environment and the robot itself. Second, the robot is less likely to become tangled in debris while moving in the environment. Third, the robot is less disruptive to the natural environment due to low acoustic noise. All experiments in this thesis are conducted in simulation. Payload transportation can be broken into the following sub-tasks: (1) approach the target, (2) capture the target, (3) deliver the target. These three tasks are first investigated individually and then combined to solve the entire payload transportation problem.

In nature, water snakes propel themselves by propagating sinusoidal waves along their bodies. Likewise the snake-like robots in this thesis employ sinusoidal locomotion patterns in order to generate forward velocity. Sinusoidal parameters are optimized with a genetic algorithm to produce maximum average speed. Additionally, the steering algorithm utilizes forward momentum to direct the robot toward the target object. An engineered "grasping" algorithm can be executed when the robot is near the object. Once the robot has grasped the object, it executes a similar swimming/turning behavior with the object in tow. Finally, the robot releases the object at a destination zone.

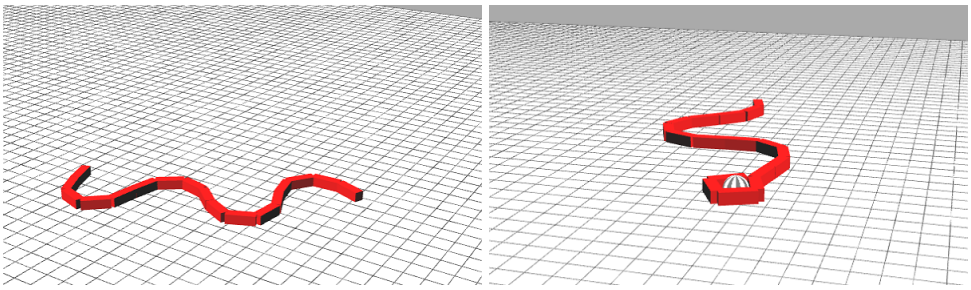
A goal of this research is to assess the effectiveness of evolutionary algorithms for optimizing payload transportation. Whereas many evolutionary

computation studies focus on solving single tasks, this thesis combines engineered solutions with results from evolutionary runs to realize a complex behavior including multiple sub tasks.

Chapter 1

Introduction

Conventional robots in an aquatic environment are often driven by an external propeller and employ an exterior apparatus to capture a target object. This thesis proposes a robot capable of both locomotion and grasping of a target object without specialized hardware for either task. The robot mimics serpentine locomotion by deforming its body which comprises a number of rigid segments connected by motorized joints (e.g. using a servo motor[1]). Such a system, as depicted in Figure 1.1, is presumably easier to construct than a device with an external propeller and specialized grasping apparatus.



(a) (b)

Figure 1.1: Example of a snake-like robot in a simulation environment. (a) shows the robot while performing serpentine locomotion. (b) illustrates forward movement with a grasped object.

Problem

The proposed robot requires complex control strategies in order to combine locomotion and grasping tasks. Many researchers investigated robotic snake-like locomotion patterns for generating forward propulsion ([56] and [38]) or turning gaits ([66], [37], [65] and [23]). Others, studied steering mechanisms for such robots ([41], [66], [23] and [61]), but a higher level control

[1] A servo motor is a motor with precise control of its angular position.

for combing different locomotion patterns remains unexplored.

Sims [58] first employed evolutionary computation for the design of aquatic robots. In that study, morphology and a brain (e.g. implemented with an artificial neural network) are evolved together in order to fulfill the tasks: (1) swimming, (2) walking, (3) jumping, and (4) following an object. Together with early work of Brooks [5] and others, the field of evolutionary robotics was formed. This area of research has been extended by many others (e.g. Lessin et al. [35]) to evolve behavior like forward locomotion, turn left, turn right. Bongard [4] found that the evolution of behavior should involve both a robot's body and its brain. Furthermore, Bongard [4] argues that complex behavior should be evolved step-wise by starting with simple tasks and progressing to more complex tasks. Creatures with evolved morphology often outperform creatures with a fixed morphology, although they use the same behavioral strategy to fulfill a given task [34]. However, these researchers evolve complex behavior which could be hand-coded by a human engineer in less time.

Purpose

This thesis focuses on control strategies for the motorized joints in order to generate effective locomotion and grasping behaviors. Solutions for simple tasks are combined to generate complex behavior. Engineered solutions in conjunction with solutions found through computational evolution. Specifically, problems that are difficult (or even impossible) to solve for a human engineer (e.g. multidimensional parameter optimization for non-linear functions) are solved by genetic algorithms. Tasks that can be implemented faster through human intuition than by computational calculation utilize engineered solutions.

Evolutionary robotics is not limited to computer simulations [39]. A robot's sensory data can be presented either to an onboard controller or transferred to an external personal computer which employs evolution. Therefore, the concepts presented in this thesis may be used to design real world robots. Empirical studies for evolved real world robot controllers are shown in [49].

The goal of this thesis is to explore the integration of engineering and evolutionary computation for so as to enable a robot to fulfill the task of capturing and transporting an object to a target destination in an aquatic environment. This complex task is divided into three subtasks: (1) approach the object, (2) grasp the object, (3) deliver the object. This thesis should answer following research questions:

1. How can snake-like locomotion be implemented in order to perform propulsion?
2. How must the robot's morphology be reconfigured to capture and transport a target object?

3. Which parts of the robotic design process should employ evolutionary computation?

These questions are answered with the research methodology described in Chapter 2 by conducting an extensive literature review of biological and robotic snake-like locomotion as well as grasping in Chapter 3. Chapter 3 analyzes gaits proposed in the literature by using a visual implementation presented in Chapter 4. Relevant machine learning methods are then described in Chapter 5. Chapter 6 presents the employed simulation environment in this thesis. Experiments are described in Chapter 7. The conclusion in Chapter 8 evaluates the outcomes of the experiments.

Chapter 2

Methodology

This thesis is based on empirical research conducted by means of experimentation and observation. *In silico*[1] experiments are executed in a simulation governed by a 3-D rigid body physics engine. Recorded simulation data is used to perform a quantitative analysis with statistical methods. Furthermore, a qualitative assessment is conducted by observing the visualization of a simulation. The research process is divided into the following consecutive steps (based on [30]):

1. Specifying the problem and defining its scope.
2. Investigating solutions to related problems occurring in natural systems through literature review.
3. Formulating algorithms to leverage/mimic biological solutions for the technological problem in experiments. The MATLAB script Shape Visualizer, described in Chapter 4, is used for algorithm design and assessing algorithm effectiveness.
4. Implementing experiments on a local computer. Experiments can either (1) be divided into a part that evolves parameter values for a controller used in the physics simulation and a part that controls the physics simulation with the evolved parameter values, or (2) use hand-coded parameters for a controller in the physics simulation. In this thesis, a controller is responsible for the adjustment of animat (artificial animal) locomotion parameters. A local computer with a *Windows 7* operating system is used to encode and test the experiment. *Microsoft Visual Studio 2012* with *Python Tools for Visual Studio 2.1* is used as an integrated development environment for *Python 2.7* encoded experiments. Open Dynamics Engine (ODE), described in Section 6.1, is applied to perform physics simulations. Neural network controllers are created with the library *MultiNEAT*. *DEAP* (*Distributed Evolutionary Algorithms in Python*) is used for the evolution of a controller's

[1] *In silico* is a term to express a simulation executed on a computer.

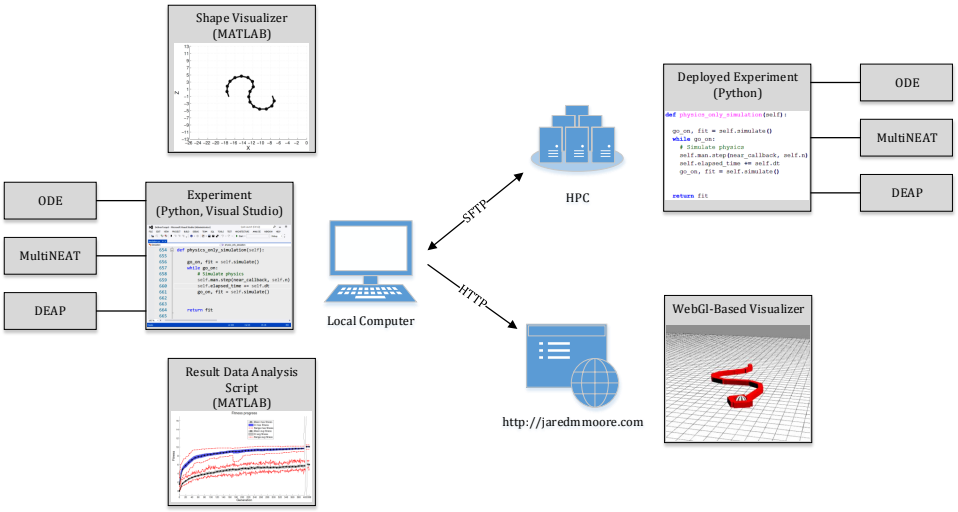


Figure 2.1: Overview of involved components. Shape Visualizer is used for the algorithm design on a local computer. Experiments are implemented in Python by using the libraries ODE, MultiNEAT and DEAP. Evolutionary experiment runs are deployed on a HPC through a SFTP connection. Result data is retrieved after the evolutionary runs finished. The result data is analyzed with MATLAB scripts and visualizations are generated by using the WebGl-Based Visualizer at http://jaredmmoore.com/WebGL_Visualizer/visualizer.html.

parameter values with highly standardized algorithms.

5. Conducting experiments that consist of evolutionary runs with 18 replicates on a high performance computer (HPC). Multiple replicates (with different random number generator seeds) are used to produce statistically significant evolutionary runs.
6. Analysis and processing of the generated result data on the HPC. MATLAB is used to transform and prepare recorded simulation data. Furthermore, it is used for the calculation and visualization of performance indicators. Visualization tools, described in Chapter 6.3, are utilized to evaluate the behavior of an animat in the simulation environment.
7. Interpret and summarize the analyzed data with statistical methods which may inspire other experiments.

Figure 2.1 depicts a system overview with all involved components.

Evolutionary runs are described in detail in Chapter 7. The following sections contain short descriptions of tools mentioned in the research process that are not further explained in other parts of this thesis. A complete list of software tool versions can be found in Appendix A.

MATLAB
MATLAB is an environment for numerical computation, visualization and programming which can be used to analyze data, develop algorithms and create applications [75]. It contains built-in mathematical functions that enable fast development of statistical tests, equation solvers and graph functions [27]. Data is stored in variables and handled with matrix manipulation functions. A sequence of commands can be stored in MATLAB scripts. Functions take input parameters, execute a sequence of commands in their own variable environment and return a result. Data visualization can be programmed with built-in two- and three-dimensional plotting functions. It provides easy data import, data transformation and export functionality and is widely used by engineers and scientists [75].

Python
Python is a high-level programming language that supports multiple programming paradigms like object-oriented and functional programming [72]. It is an intuitive scripting language that utilizes an interpreter at run time in order to generate an intermediate byte code that is executed by a virtual machine. Data types are inferred at run time and memory is managed automatically. Therefore, it can be used to rapidly develop highly readable algorithms. Python is used by a large number of developers in the scientific computing community [43].

Visual Studio with Python Tools for Visual Studio
Visual Studio is an integrated development environment (IDE) used to develop computer and web applications on the operating system Microsoft Windows [70]. It supports various programming and scripting languages as well tools for project management, software lifecycle management, cloud applications and phone development [59]. An intuitive interface with full tool chain support, static code analysis and debugging capabilities for many programming languages enable a researcher to rapidly develop reliable applications. Python Tools for Visual Studio (PTVS) is a plugin for Visual Studio that turns it into a Python IDE [69]. PTVS allows developers to use Visual Studio's capabilities for Python programming. It provides automatic syntax and hierarchic analysis of Python code [55] in conjunction with an interactive Python console.

High Performance Computer
HPCs offer speed and capacity significantly greater than machines built for commercial use [17]. They generally utilize a massive number of processors in order to solve computationally intensive problems. Processors can either be distributed over many local machines, where each machine solves a self contained sub task and reports its result back to a server, or placed in

proximity (e.g. in a computer cluster) to work together without immense data transfer overhead.

The HPC used to perform experiments for this thesis utilizes multiple nodes consisting of many processors in spatial proximity in order to reduce networking overhead. Each node performs one replicate of the evolutionary run at a time, while each processor of the node is used to run one simulation task at a time. One generation of the evolutionary run scatters its simulation tasks over all processors of a node and gathers its results when the simulation is finished. Depending on the utilization of the HPC and the number of replicates, one node may compute all replicates serially or all replicates may be computed in parallel over all nodes. In detail, the HPC used to perform experiments consists of 10 nodes where:

- nodes 1 until 6 each have 32 processors with $2400MHz$ per processor, and
- nodes 7 until 10 each have 64 processors with $2100MHz$ per processor

with 64-bit registers. The HPC uses the Linux distribution *SMP Debian 3.2*.

Chapter 3

Animat

The investigated snake-like[1] animat moves in a simulated aquatic environment. This thesis investigates "underwater" locomotion with neutral buoyancy as in [42], where authors focused on momentum generation for eel-like robots. The robot of Crespi et al. [11] is a near surface swimmer and generates locomotion patterns with a non-neutral buoyancy. While [56] and [38] investigate the physics of locomotion generation, other researchers ([41], [66], [23] and [61]) have studied gait generation for steering a robot in a terrestrial environment. This chapter utilizes the developed Shape Visualizer presented in Chapter 4 in order depict the robot's shape deformation.

3.1 Components and Layout

As described in Section 3.2.2, terrestrial motion patterns can be applied to aquatic locomotors. The animat is capable of moving in three degrees of freedom (DOF), without touching the floor of the environment. Its body consists of n identical rigid links connected with $n-1$ actuated joints as shown in Figure 3.1. The animat's undulatory motion can be compared with the dynamics of a free-moving serial chain. As shown, components are defined in the global coordinate system (X, Y, Z). The dimensions of a link are given in $(l \times h \times w)$, $1.5 \times 0.5 \times 0.5$ *units* in this study. Joints are indicated in Figure 3.1 to show their role and function in the animat's body in the simulation. Adjacent links are placed consecutively after each other, which would lead to collisions when a joint is rotated. To this end, adjacent links are defined to be allowed to intersect with each other's body. This behavior emulates the flexible nature of natural organisms, whose bodies are not simple 3D primitives.

Joints are used to describe the angular relation between two adjacent links. As indicated in Figure 3.1, two links rotate around the Y axis which

[1] Snake-like refers in to the behavior of elongated creatures that derive their movements primarily from their main body form rather than limbs or fins.

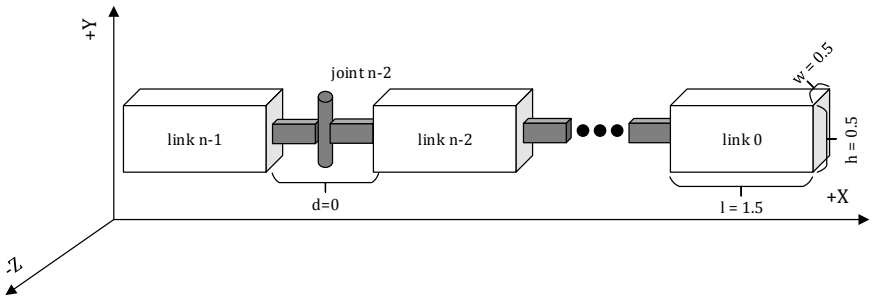


Figure 3.1: Animat consisting of n links and $n-1$ joints in a three dimensional global coordinate system (X, Y, Z). A joint connects two adjacent links. Link and joint indexing starts from greatest X position and ends at the smallest X position. The dimensions of each link are defined to be $(l \times h \times w)$ $1.5 \times 0.5 \times 0.5$ *units*. A joint is defined to have no distinct dimensions $(d = 0)$, but is shown here to emphasize its role in the animat's model.

describes the characteristics of a hinge joint. This thesis will use the term joint as a synonym for hinge joint. Figure 3.2 shows the actuation of a 2-link joint. The initial link position in 3.2(a) defines the initial angular relation between both links with 0° in the joint. A rotation with value $+\theta$ leads to a rotation of $+\theta/2$ for $link0$ and $-\theta/2$ for $link1$ by the definition of a positive counter-clockwise rotation. The angle θ is restricted to be $\theta \in [-90°, +90°]$. Each joint in an animat's body defines its own local coordinate system in order to describe the relation between two adjacent links.

In order to perform a rotation as shown in Figure 3.2, a joint has to apply a torque τ (respectively $-\tau$) on $link0$ and $link1$, as shown in Figure 3.3. In detail, the torque is calculated with $\tau = r \times F$, where r is a vector that defines the COM (center of mass) displacement from the joint origin, in this case $l/2$, and F the force vector. The force vector is calculated with $F = m*a$, where m is the mass of the link and a is the vector of acceleration. Hence, a certain force is required to rotate a link around the angle $\theta/2$. This force is usually limited by hardware properties, e.g. the maximum torque of a servo motor. The aim of this thesis is not to build a fully realistic hardware simulation. To this end, maximum forces have been set high enough to allow smooth animat motion in the physics simulation.

The animat is defined to have a uniform form mass m distribution. In other words, each link is defined to have the same mass m, while joints are defined to have mass $m = 0$. This leads to a constant density, according to the equation $\rho = \frac{m}{V}$, because V is defined to be $1.5 \times 0.5 \times 0.5$ for each link. As previously mentioned, the animat is defined to have neutral buoyancy. Buoyancy, or upthrust, is the contradicting force to gravity in a

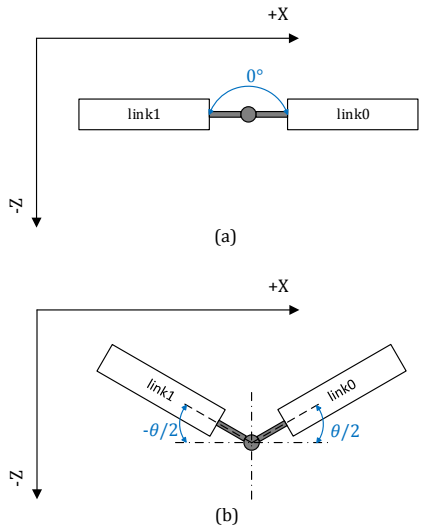


Figure 3.2: Top view of a joint actuation in an animat consisting of two adjacent links and one joint in the global coordinate system. (a) shows the initial link position and the initial joint position at 0°. The joint is then actuated by a value θ which leads to a rotation of both links (b).

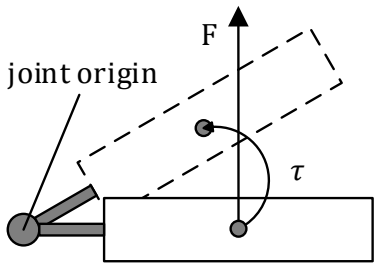


Figure 3.3: Top view of a link that is rotated around Y axis at the joint origin with torque τ and its resulting force F. The dashed rectangular represents the estimated link position after applying torque τ on the initial link position indicated with a continuous line.

liquid. A body moves upward if buoyancy is greater than gravity and vice versa. Neutral buoyancy generates neither upward nor downward thrust. A body stays at the same vertical position if no external force is applied.

(a) Lateral undulation

(b) Concertina locomotion

(c) Sidewinding locomotion

(d) Rectilinear crawling

Figure 3.4: Four most common snake gliding types. (a) Eastern garter snake slithers through a muddy area [74] published in public domain. (b) Corn snake moving in concertina mode [40] licensed under CC BY. (c) Sidewinding rattlesnake [71] reproduced with permission from AAAS. (d) Great basin gopher snake rectilinear locomotion [76] reproduced with permission from Filip Tkaczyk.

3.2 Locomotion

Snake-like creatures employ many unique locomotion patterns adapted to specific environments. Movements can be categorized as creeping on the ground, jumping up in the air, or gliding in the air by aerodynamically deforming the animal's body. The four most common gliding types of movement identified in [24], [61], [36], [56] can be classified into:

1. **Lateral undulation**: Lateral undulation is achieved by propagating waves along the animal's body, see Figure 3.4(a).
2. **Concertina locomotion**: Concertina locomotion is obtained by stretching and curving body parts. The curved body part stays at the same position, while the stretched body part is moved forward, see Figure 3.4(b).
3. **Sidewinding locomotion**: The lifting and curving of an animal's body parts, while the lifted body part is moved parallel to its lateral orientation, is called sidewinding locomotion, see Figure 3.4(c).
4. **Rectilinear crawling**: An animal performs rectilinear crawling when it uses the edges of its body to pull itself forward, see Figure 3.4(d).

This thesis investigates animats in an aquatic environment and will focus on lateral undulation.

3.2.1 Lateral Undulation

Lateral undulation is the wave-like movement of body parts in order to generate forward propulsion. This type of motion is also called serpentine movement/crawling [24], [36], [61], [56] for terrestrial snake movement. Lateral undulation employed by aquatic swimmers is often referred to as anguilliform locomotion [11], [42] or simply swimming [65] for eel, lamprey and other underwater snake-like animals.

Gray [22] first conducted a qualitative analysis of lateral undulation with several eel-like fish. He found that sinusoidal waves are propagated along the fish's body from head to tail with a wave length slightly less than the fish's body length. Each body part makes similar "tracks" [24] or, in other words, passes the same point in an n-dimensional coordinate system. Irregularities in the environment, e.g. caused by ground friction (terrestrial) or hydrodynamics (aquatic), push against the body which leads to forward propulsion. Lateral undulation is therefore not suitable to generate forward movement on a slippery surface [37] or in vacuum. Figure 3.5 depicts the principal movement of a water snake-like animat with lateral undulation. The body moves sinusoidally while generating propulsion. Each time step shows the current spatial position. It moves $distance_1$ from time step $t = 0$ to $t = 1$ and $distance_2$ from $t = 1$ to $t = 2$.

3.2.2 Undulatory Propulsive Force Generation

Forward motion cannot be generated by just moving an animal's body parts according to a sinusoidal wave. As previously stated, either ground friction or hydrodynamics are required in order to generate positive propulsive force in the transverse direction.

Aquatic drag forces have similar influences on the undulatory snake-like mechanism as anisotropic[2] ground friction properties [36]. This means that undulatory motion properties in a terrestrial environment can be transferred to an aquatic environment and vice versa.

Gray [22] explains the generation of propulsive force in an aquatic environment in transverse body direction by the displacement of water. Figure 3.6 depicts the animat's body surface which generates propulsive force. Water is deflected along the animat's body, generating two forces F_d and F_p. The water deflection force (F_d) is parallel to the body and the pressure force (F_p) directs normal to the body, representing the pressure against the water

[2] An environment where property values, e.g. viscosity or density, differ between two or more measurements (e.g. spatially displaced) is called anisotropic.

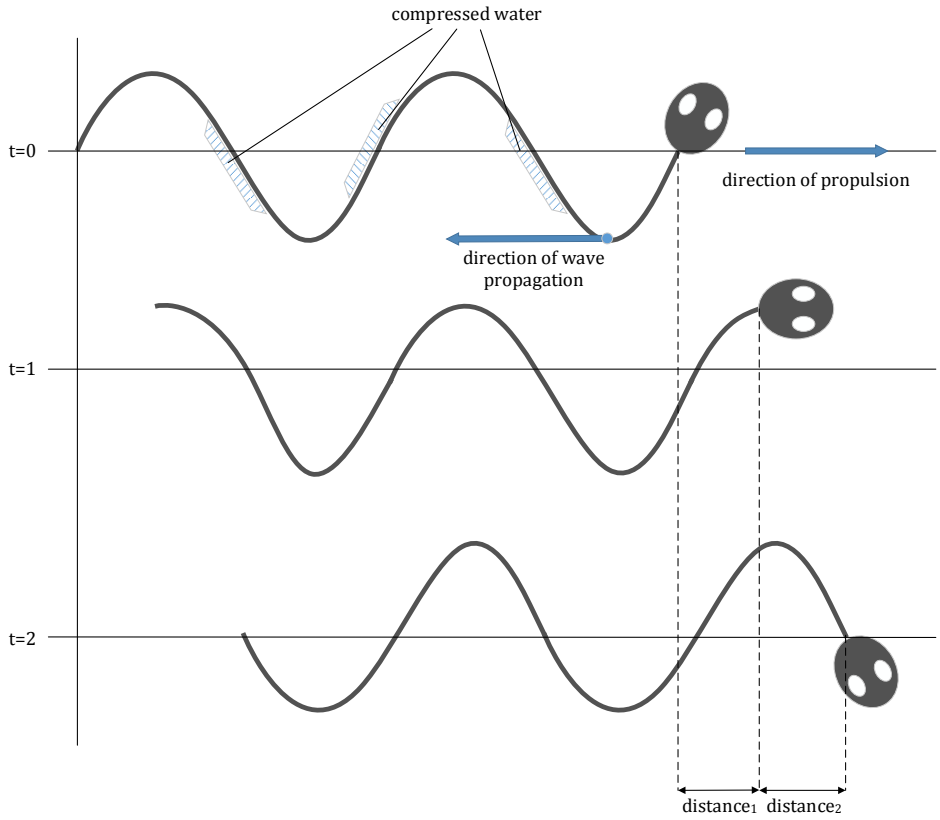


Figure 3.5: Lateral undulation of a water snake-like animat in an aquatic environment. Compressed water is painted as blue grid at $t = 0$. The sinusoidal wave travels from the head (right) and to the tail (left). This generates a propulsion to the right. $t = 0$, $t = 1$, $t = 2$ define time steps in the animal's movement. The animal's body moves from the left to the right with distances $distance_1$ and $distance_2$ between the time steps.

generated by a propagating wave. The accumulation of F_{dp} (force component in F_d) and F_p results in propulsive thrust.

Liljebäck et al. [36] analyzed the propulsive force generation of a snake-like robot in a terrestrial environment. The robot consists of n links and $n - 1$ connecting joints and moves along a sinusoidal wave. The authors examined the contribution of each link to the total propulsive force and deduced that forward propulsion in the longitudinal direction is generated by link movement in the transversal direction. They also showed that the magnitude of the propulsive force increases by increasing the link velocity in transversal direction or by increasing the link angle offset, as long as it stays under 45°. Furthermore, they conclude that a link produces the highest

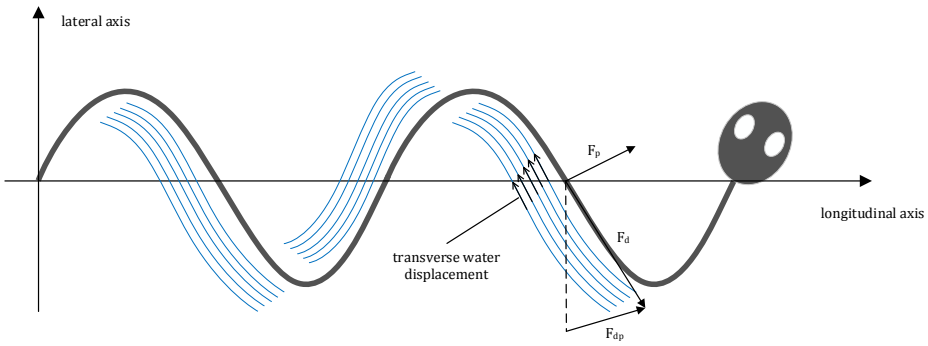


Figure 3.6: Propulsive force generated by moving the body according to a sinusoidal wave. Water is deflected along the body and generates the force F_d. Pressure is generated due to pushing against compressed water (F_p) which directs normal to the animat's body. Force F_{dp} is the force component in F_d that directs into the same direction as F_p.

propulsive force from a given link velocity if the link angle is $\theta_i = \pm 45°$ to the forward direction.

3.2.3 Joint Angle Calculation for Generating Propulsion

Hirose [24] studied the body movement of biological snakes which led to the mathematical formulation of the *serpenoid curve*. The planar *serpenoid curve* describes a snake's spatial and temporal body movement with:

$$x(s) = \int_0^s cos(a * cos(b * \sigma) + c * \sigma)\ d\sigma, \tag{3.1}$$

$$y(s) = \int_0^s sin(a * cos(b * \sigma) + c * \sigma)\ d\sigma, \tag{3.2}$$

where $(x(s), y(s))$ represents a point in the coordinate system, and s is the arc length from the robot's origin point. The variables a, b and c are defined as scalars. It is important to note that Equations 3.1 and 3.2 describe the robot at a certain time instance.

Saito et al. [56] derived a discrete approximation of Hirose [24]'s continuous *serpenoid curve* for n-link robots. The authors state that a snake's undulatory locomotion can be imitated by changing the relative angles of the robot's links with:

$$\phi_i(t) = \alpha * sin(\omega t + (i-1)\beta) + \gamma, \tag{3.3}$$

where i represents the index of a robot's joint as $i \in 1, ..., n-1$ and $n = number\ of\ links$. The angular frequency ω determines how fast a *serpentine*

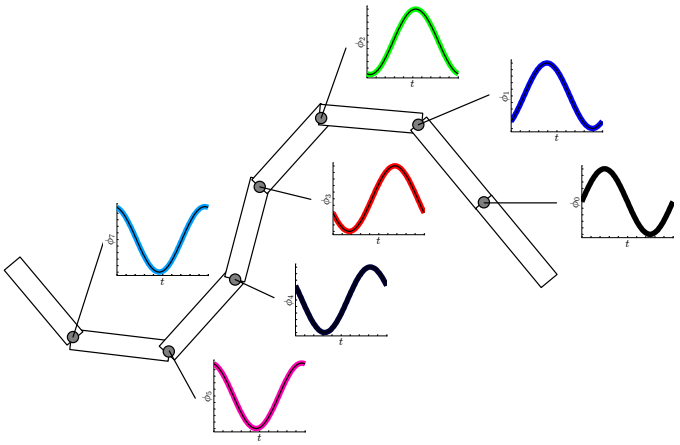


Figure 3.7: 8-link animat with corresponding temporal sinusoidal joint angle paths at a certain time step. The current shape of the animat is the result of applying ϕ_i to the corresponding joint at $t = 0$. Each joint has a distinct joint angle course.

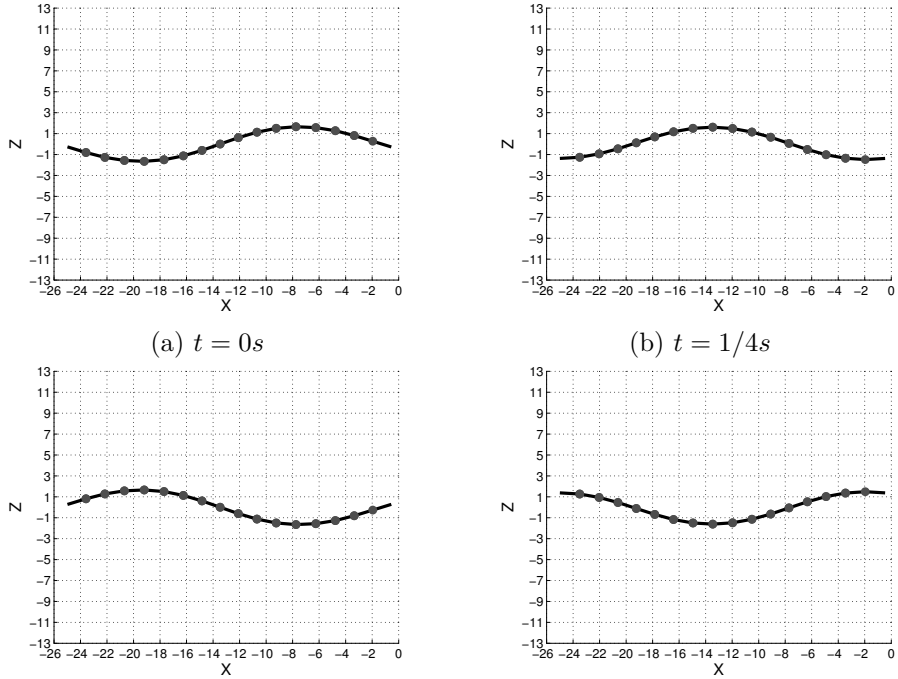
curve propagates along a robot's body and is therefore mainly responsible for forward locomotion speed. Furthermore, the variable α specifies the amplitude of a joint's angle, which is highly dependent on the robot's surrounding environment. A joint's phase shift in respect to the first joint is calculated with joint index -1 $(i - 1)$ times phase shift constant (β). Saito et al. [56] found that β values can be roughly approximated with $2\pi/n$. The offset variable γ biases joint angle values and can be used to change the robot's direction of motion. In detail, this variable can be used to veer the robot to the left or to the right. Setting $\gamma = 0$ leads to straight forward motion.

Variables α, ω and β are highly interdependent. E.g., increasing the value of ω while keeping α very small would not lead to an observable increase of a robot's forward velocity. For example, Yang et al. [65] show experimentally the relation between α, ω with respect to a underwater snakelike robot's forward velocity. Yang et al. [65] and McIsaac and Ostrowski [42] outline the impact of the phase shift constant β on the robot's forward velocity respectively acceleration. Saito et al. [56], Hasanzadeh and Tootoonchi [23] and Liljebäck et al. [37] present methods to optimize α, ω and β variables.

This thesis employs Equation 3.3 [56] in the notation

$$\phi_i(t) = \alpha * sin(\omega t - i\beta) + \gamma, \tag{3.4}$$

where i defines the joint index with $i \in 0, ..., numberofjoints - 1$. Calculated ϕ_i angles are applied to the robot's $n - 1$ joints. Figure 3.7 shows an animat's shape at $t = 0$. The robot is actuating each joint according to its joint angle course over time (as pointed out in the colorized course next to the joints) in a sinusoidal manner. This is shown in Figure 3.8. Links are



(a) $t = 0s$ (b) $t = 1/4s$

(c) $t = 2/4s$ (d) $t = 3/4s$

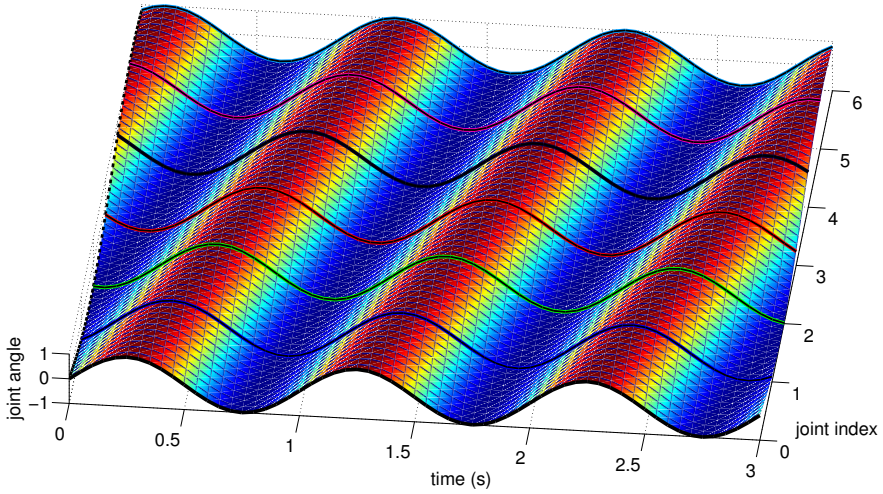
Figure 3.8: 17-link animat that performs a forward locomotion gait with parameters $\alpha = 0.1$, $\omega = 2\pi$, $\beta = 2\pi/16$ and $\gamma = 0$ for Equation 3.4 in a vacuum environment. (a)-(d) depict the time progress starting from 0 seconds with step size 1/4 seconds.

actuated by the ϕ_i calculation in a vacuum environment. One sinusoidal wave propagates along the animat's body from right to left, producing a propulsive force in the positive X direction.

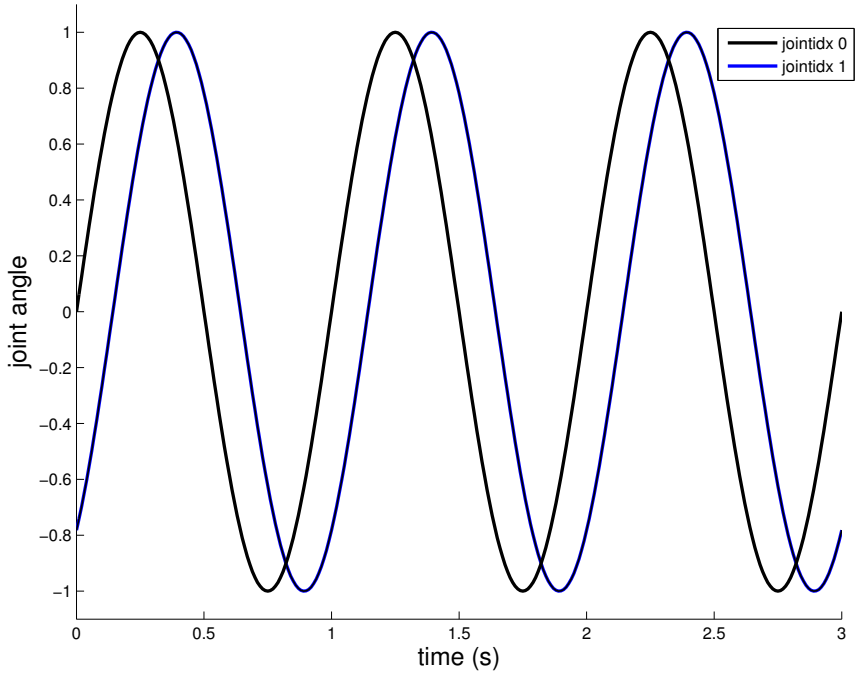
It is noteworthy that Equation 3.4 discretizes the spatial part (k) of a two-dimensional traveling sinusoidal wave in the form

$$y(t, x) \; = \; A * sin(\omega t \; - \; kx) \tag{3.5}$$

where t defines a certain time instance and x a certain spatial point. The variable A describes the amplitude of the wave, ω the angular frequency responsible for temporal propagation and k the wave number (also known as spatial frequency) responsible for spatial propagation. The relation between Equation 3.4 and Equation 3.5 is shown in Figure 3.9 (a). Quasi-continuous x and discretized i values propagate along time. This Figure depicts the joint angle progress of eight joints. Figure 3.9(b) shows the shift ($i * 2\pi/8$) of joint index 0 and 1 on the projected $angle(y) - time(x)$ plane. The start angle values at $time = 0$ for joint indices 0 and 1 are 0 and -0.71 respectively. By measuring all start angle values, one can deduce the initial shape of the



(a)



(b)

Figure 3.9: Joint angle progress according to Equation 3.4 with $\alpha = 1$, $\omega = 2\pi$, $\beta = 2\pi/7$ and $\gamma = 0$ and seven joints. (a) shows the joint angle (y-axis) over time (x-axis) and space/jointindices (z-axis) in conjunction with a two-dimensional quasi-continuous function according to Equation 3.5 with $a = 1$, $\omega = 2\pi$ and $k = 2\pi/7$. The dashed black line at $time = 0$ represents the starting angle for quasi-continuous joint indices. Each discrete joint index $i = 0, ..., 6$ is colorized. (b) depicts the joint angle progress over time for joint indices 0 and 1.

robot (black dashed line in Figure 3.9 (a)) as exemplified in Figure 3.7.

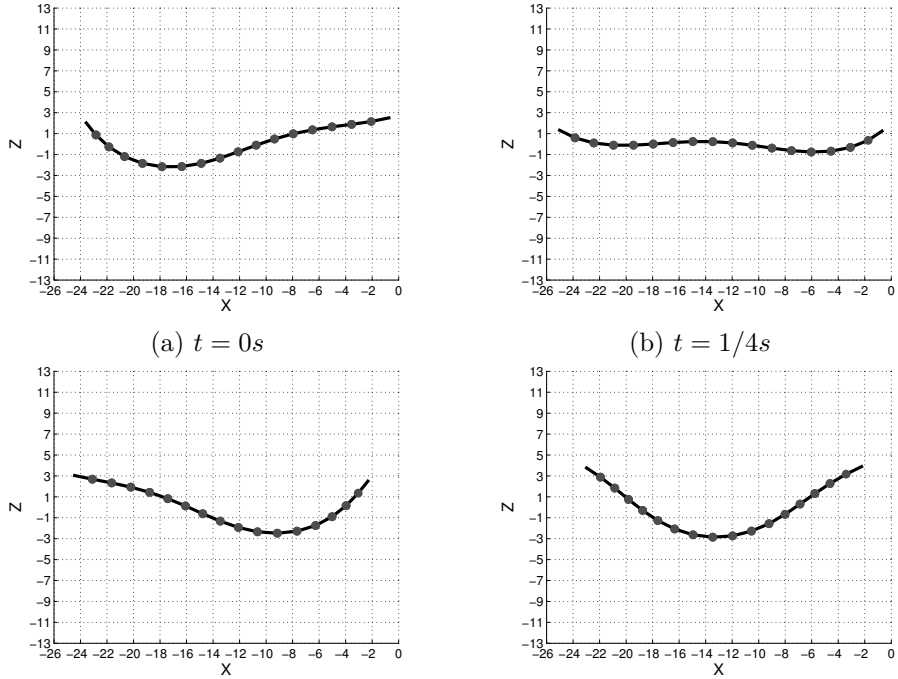
It is important to note that β can be interpreted as temporal sinusoidal phase shift between joint indices as shown in Figure 3.9 (b) and as spatial angular displacement, e.g. black dashed line in Figure 3.9 (a). By using the approximation formula $2\pi/n$, it is guaranteed that only one spatial sinusoidal wave is propagated along the robot's body [56]. Hence, it is useful to compute β in multiples of $2\pi/n$, because one can see immediately the number of propagating body waves from the optimized value (e.g. a value of 1 produces one distinct wave, a value of 2 two distinct waves and a value of 0.5 half a wave).

An ANN can be used as an alternative to the discrete approximation of a continuous *serpenoid curve* [56]. As outlined in Chapter 7, however one loses the direct control with three parameters of the animat's shape. Furthermore, an ANN requires a clock input signal in order to produce an oscillating output. The clock input signal could be omitted by using a Central Pattern Generator (CPG) [26]. CPGs are able to produce a sinusoidal rhythm which could represent the angle per joint per time step instead of calculating a sinusoidal body wave. They are commonly used to generate cyclic gaits, e.g., as used in [11] and [65]. CPGs such as ANNs cannot be used to directly modify joint actuation characteristics. An alternative to ANNs and CPGs is presented in [28], where authors adjust the β value from Saito et al.'s approximation formula in order to form an asymmetric snake body shape. Joint actuation is handled similar to the method described in this chapter (sinusoidal actuation) which means that no advantage would be gained for this thesis.

3.3 Turning Gaits

Turning gaits for the animat are limited. In fact, relatively few researchers have studied turning gaits in snake-like locomotion compared to regular forward movement.

As previously described, the term γ in Equation 3.4 can be used to turn the animat to the left or to the right. The adjustment of variable γ is called *circular motion* [41], or the *symmetrical line modulation method* [66] and has been studied in [66], [37], [65] and [23]. The variable γ adds a constant bias to each calculated joint angle $\phi_i(t)$. This means that all joint angles in an animat's body skew to the left or the right depending on the sign of the variable γ. Due to the fact that γ is added to the sinusoidal motion, turning is possible only if forward momentum is generated. This effect can be seen in Figure 3.10. The same sinusoidal wave used in Figure 3.8 propagates along the animat's body. An additional bias of $\gamma = 0.05$ leads to a positive joint angle tendency, which causes the links to turn more in the positive Z direction than in the negative Z direction. This behavior produces a veer



(a) $t = 0s$ (b) $t = 1/4s$

(c) $t = 2/4s$ (d) $t = 3/4s$

Figure 3.10: 17-link animat that performs circular motion with parameters $\alpha = 0.1$, $\omega = 2\pi$, $\beta = 2\pi/16$ and $\gamma = 0.05$ for Equation 3.4 in an vacuum environment. (a)-(d) depict the time progress starting from 0 seconds with step size 1/4 seconds.

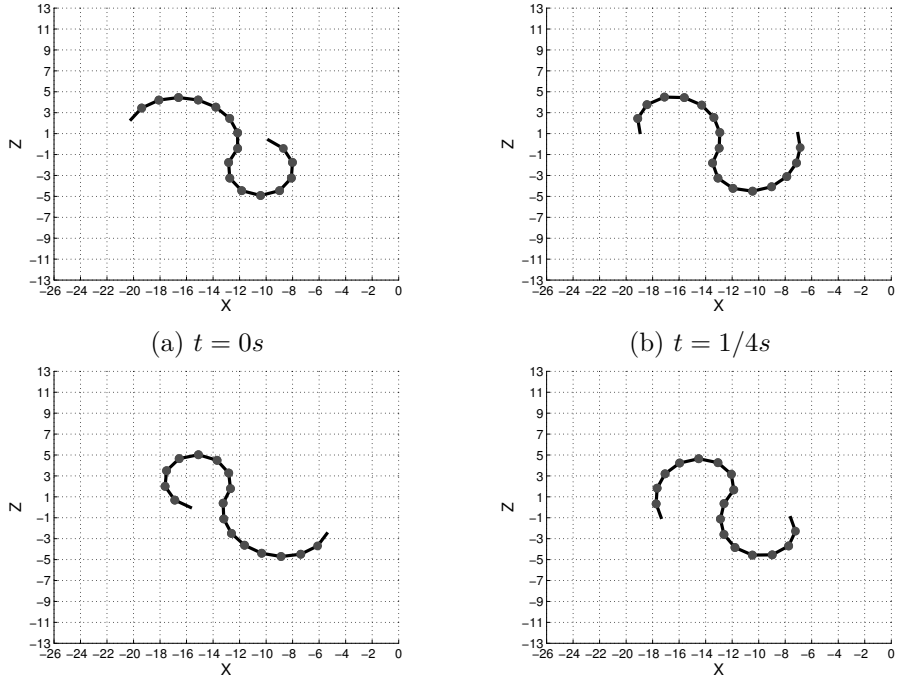
in positive Z direction if the animat is placed in an aquatic or terrestrial environment. Consequently, the direction of turn can be changed by using a negative sign for the γ value.

Beyond veering, turning in a small area is a highly desirable behavior. This movement complements other behaviors, increasing the overall agility to the robot. McIsaac and Ostrowski [42] describe a turning gait without forward movement generation for an odd number of animat links which they called, a *spinning gait*. The *spinning gait* adjusts the variables β and γ depending on the joint index i with [41]:

$$\beta(i,n) = \begin{cases} i - n/2 & i \leq n/2 \\ n/2 + 1 - i & i > n/2 \end{cases}, \tag{3.6}$$

$$\gamma(i,n) = \begin{cases} +1 & i \leq n/2 \\ -1 & i > n/2 \end{cases}, \tag{3.7}$$

where n is the number of links. This approach treats the animat's body as two parts. Each body part propagates its own sinusoidal wave from inside



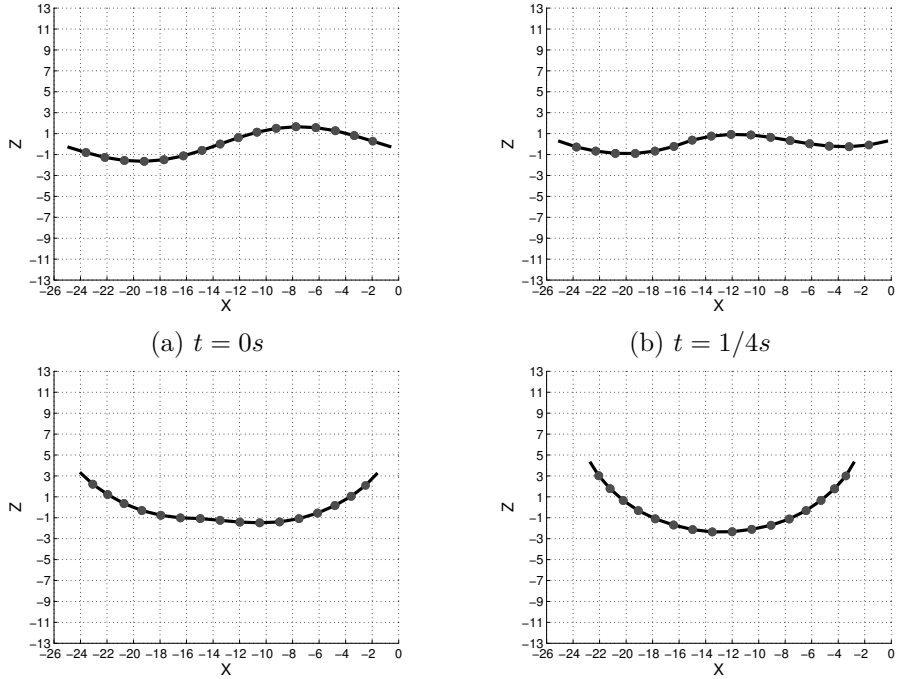
(a) $t = 0s$ (b) $t = 1/4s$

(c) $t = 2/4s$ (d) $t = 3/4s$

Figure 3.11: 17-link animat that performs a spinning motion with parameters $\alpha = 0.1$, $\omega = 2\pi$, $\beta = 2\pi/16$ and $\gamma = 0.3$ for Equation 3.4 in an vacuum environment. (a)-(d) depict the time progress starting from 0 seconds with step size 1/4 seconds.

joints to outside joints. Both sinusoidal waves generate a force directed toward the middle of animat. These two forces would eliminate each other if γ would be set to 0. Setting γ at the first sinusoidal wave to a positive value in connection with a negative γ at the second sinusoidal wave leads to accumulated force in the same circular direction. This produces the principal motion depicted in Figure 3.11. One sinusoidal wave propagates from the middle to the left. The other sinusoidal wave propagates in the opposite direction. Together, these waves generate a force which leads the animat to spin in counter clockwise direction if placed in an aquatic environment. Only a half wave is observable per body part, because $\beta = 2\pi/16$ leads to one full wave along the whole body. The direction of motion can be reversed by changing the signs of the two γ values simultaneously.

Mclsaac and Ostrowski [44] present the so called *coil* turning gait, which requires forward momentum. The snake coils around itself while in active forward motion. This result is achieved by propagating a fixed angle value ϕ along the animat's body. In other words, if ϕ_i reached a trigger value, it is held for a certain time. Afterwards, ϕ_i follows the sinusoidal curve again.



(a) $t = 0s$ (b) $t = 1/4s$

(c) $t = 2/4s$ (d) $t = 3/4s$

Figure 3.12: 17-link animat that performs a coil motion with parameters $\alpha = 0.1$, $\omega = 2\pi$, $\beta = 2\pi/16$ and $\gamma = 0$ for Equation 3.4 in an vacuum environment. (a)-(d) depict the time progress starting from 0 seconds with step size 1/4 seconds.

This produces the principal coil motion depicted in Figure 3.12. Aquatic or terrestrial drag forces turn the animat in the positive Z direction, assuming that the animat has forward momentum in the positive X direction. After a specified time the animat's joints start to consecutively actuate again, beginning from joint index 0.

Additional turning gaits such as the *amplitude modulation method* and *phase modulation method* have been investigated by Ye et al. [66]. These gaits have not been used this thesis, because they would require additional joint angle adjustment for little gain in agility.

3.4 Object Grasping Motion

The definition of the animat's snake-like layout, as well as its body actuation like a serial linked chain, limits the possibilities of grasping a target object. Research in robotic grasping mainly focuses on emulating the two functions: (1) restraining, and (2) manipulating of human hands [3]. In detail, a so-called robotic *fingertip grasp* [3] holds an object with the ends of the robot's

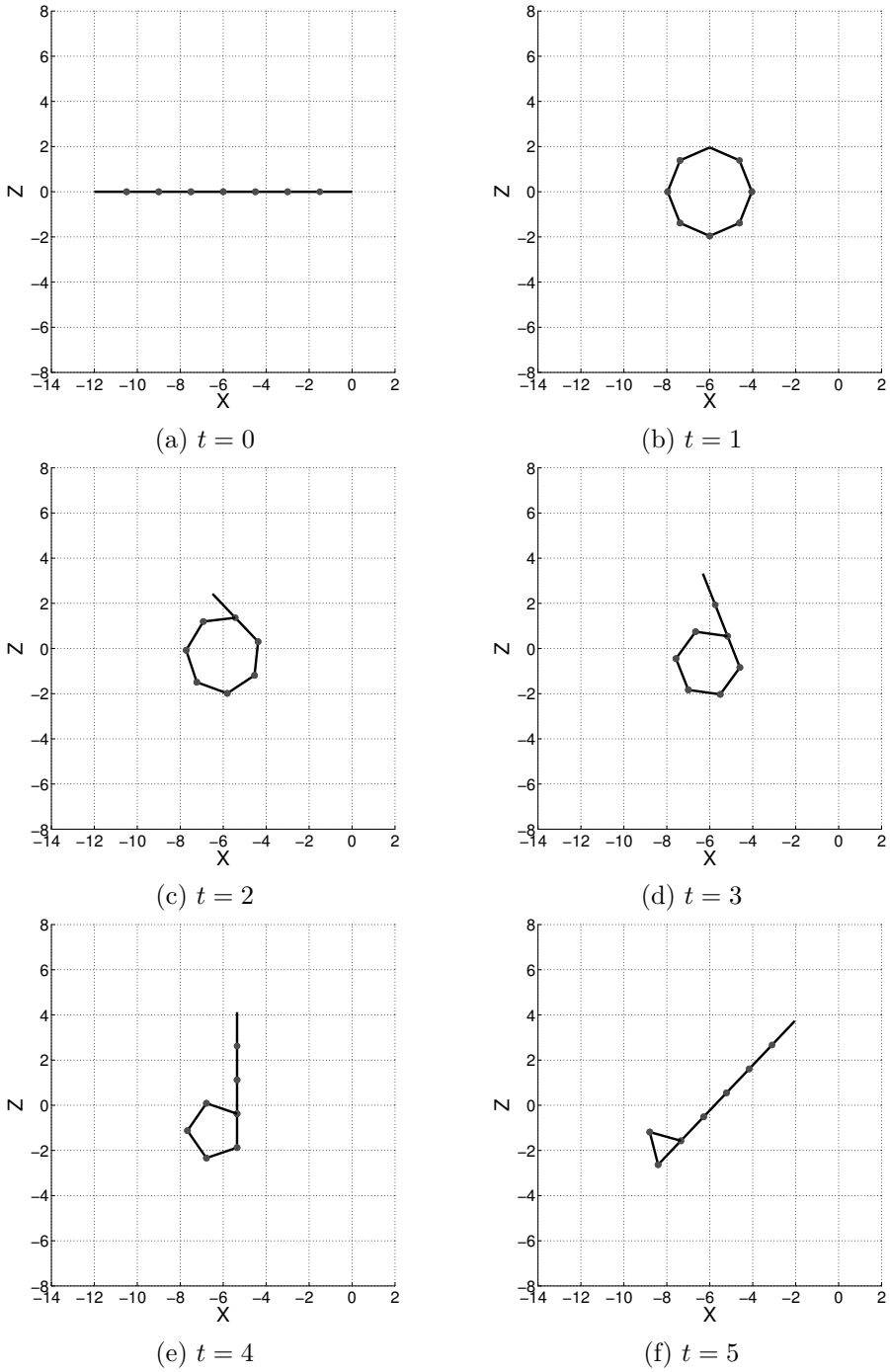
Algorithm 3.1: Move body links properly to envelope an object with $|graspjoints| + 1$ links

```
1: GRASP(|joints|, |graspjoints|)                    ▷ joint_i ≙ θs_[i]
2:     θs ← 360°/|joints| + 1)                       ▷ set all θs - form a circle
3:     moveanimat(θs)                                ▷ e.g. simulation step

4:     for i in |joints| − 2 and i < |joints| − |graspjoints| do
5:         θs_[0:i] ← 0                              ▷ set θs [0:jointindex)
6:         θs_[i:|joints|] ← 360°/(|joints| + 1 − i)  ▷ set θs [i:|joints|)
7:         moveanimat(θs)
8:     end for
9: end
```

fingers [33]. The proposed animat could utilize this approach to attach to an object by using the start and the end link to clamp a target object, but it would not be able to subsequently move with target object. Hence, an object enveloping/surrounding mechanism as described in [62], [24], [33], [50] and [10] is used to grasp the object with one part of the animat's body while retaining the other body part for actuation. Trinkle et al. [62] utilize the two end parts of a 3-link chain like robot to grab a polygonal object. The robot's middle part is mounted to another device which stabilizes the grasping motion. Hirose [24] describes a gripper made out of two separate chains to fully surround an object of arbitrary shape. Lenarčič et al. [33] adapt the previously mentioned idea of a robotic finger, which consists of 3 links and 3 joints where one joint is connected to a fixed surface. In contrast to the *fingertip grasp* the links are used to partially surround and clamp a circular object. Pandolfi et al. [50] presents a biologically inspired method that surrounds an object with one part of the robot's multi-link body as many times as possible.

Algorithm 3.1 is proposed in order to fully surround a target object while retaining $|graspjoints| + 1$ links for actuation. The algorithm starts with forming a circle around the animat's COM. The target object is captured if it is positioned within the circle. Afterwards, each iteration straightens one more joint by setting $\theta = 0°$. Hence, each iteration has to increase θ values of the "grasping" joints in order to remain a circle. This means that the circle is tightened with each iteration. The target object moves within the tightened circle assuming $m_O << m_A$ where m_O is the target object's mass and m_A is the animat's mass and the circumference of the target object is embraceable by $|graspjoints| + 1$ links. Figure 3.13 depicts the animat's body movement when applying Algorithm 3.1.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$ (d) $t = 3$

(e) $t = 4$ (f) $t = 5$

Figure 3.13: 8-link animat that applies Grasping Algorithm 3.1 with $|graspjoints| = 3$ starting from an initial position (a) at $t = 0$. (b) shows the formed circle at $t = 1$. (c)-(f) depict the tightening of the circle in which a target object can be placed.

Chapter 4

Shape Visualizer

This chapter presents the developed MATLAB script Shape Visualizer which models a mutable robotic shape after the animat's layout definition in Section 3.1. The visualization of an animat's shape by using only its joint angle values is beneficial for the evaluation of theoretical motion patterns, as investigated in Chapter 3. E.g., collisions can be detected before the simulation with a physics engine has been started.

The animat's body can be approximated as a n-link serial chain. Therefore, the proposed shape visualizer geometrically calculates the link positions $(x\_start, z\_start, x\_end, z\_end)$ in a two-dimensional coordinate system of a loose n-link serial chain and $i = n - 1$ joints with an uniform mass distribution. The rotation of a point $P = (x, z)$ around the Y-axis by the angle θ is performed with:

$$P_{rotated}(P, \theta) \;=\; R_y \;*\; P \;=\; \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \;*\; \begin{pmatrix} x \\ z \end{pmatrix}, \tag{4.1}$$

and the translation by the vector $V = (v_x, v_z)$ is calculated as:

$$P_{translated}(P, V) \;=\; P \;+\; V \;=\; \begin{pmatrix} x \\ z \end{pmatrix} \;+\; \begin{pmatrix} v_x \\ v_z \end{pmatrix} \;=\; \begin{pmatrix} x + v_x \\ z + v_z \end{pmatrix}. \tag{4.2}$$

Figure 4.1(a) depicts an initial shape of an 8-link chain in a planar coordinate system with X- and Z- axis for the shape analyzer algorithm. As shown, the algorithm assumes a link length of 1.5 *units* which results in a total length of 10.5 *units* for a 8-link chain. The 8-link chain contains seven joints indexed with $i = 0, ..., 6$. Link and joint indexing starts from the most positive X position and continues to the most negative X position. Positions $link\_start_i$ and $link\_end_i$ overlap with the $link\_end_{i-1}$ respectively $link\_end_{i+1}$ position, except $link\_start_0$ and $link\_end_7$. Those represent the start (respectively, the end) of the chain and are therefore not connected to other links. The initial shape in Figure 4.1(a) is useful for describing a chain, but it is not a requirement for the algorithm to work properly. A

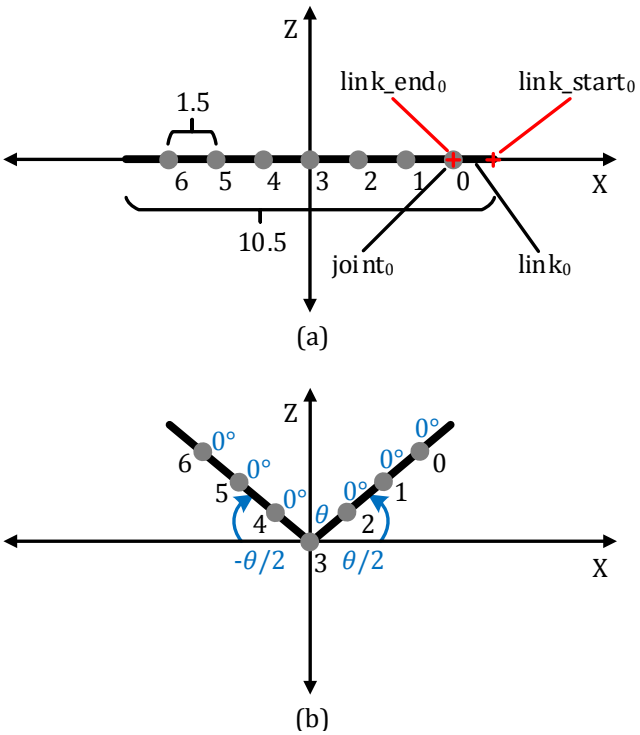


Figure 4.1: Geometric transformation of an 8-link chain from an initial position (a) and applying a angle θ value to joint index 3 (b) under the assumption of a link length of 1.5 *units*. (a) exemplifies the position of $joint_0$, $link_0$, $link\_start_0$ and $link\_end_0$ as well as the indexing of the joints. (b) depicts the applied angle θ value to joint index 3 while all other angles keep the value 0°.

chain's initial shape in the two-dimensional coordinate system may be defined differently.

Each joint is positioned between two adjacent links ($link\_end_i = link\_start_{i-1}$) and defines its own coordinate system. The joint's coordinate system is used to indicate the angle θ between two adjacent links. An angle θ is defined as $\theta \in [-180°, +180°]$. A θ value of 0° indicates that both links form a straight line, which leads to a maximum distance between $link\_start_i$ and $link\_end_{i-1}$. Setting the angle θ to 360° (or its integer multiples) leads to an overlapping of both links. Positive angle values lead to a link rotation in positive Z direction and negative angle values result in a rotation in negative Z direction, by assuming the initial link positions in Figure 4.1(a).

Figure 4.1(b) shows the result of setting the angle at joint index 3 θ_3 to a positive value, while keeping all other joint angles at 0°. It can be seen that the algorithm rotates the leading link, here $link_3$, with $+\theta/2$ and the

trailing link, here $link_4$, with $-\theta/2$ in order to apply a total angle of θ. Consecutively, the algorithm iterates through all leading links (here $link_2$, $link_1$ and $link_0$) and through all trailing links (here $link_5$, $link_6$ and $link_7$). In each iteration the algorithm performs a rotation with $+\theta/2$ respectively $-\theta/2$ as well as a translation in order to keep the serial chain connected.

The halving of θ at $link_3$ and $link_4$ describes a special case and is the consequence of two reasons: (1) the angle θ has been applied for the middle joint of the chain, namely joint 3, and (2) the algorithm assumes equal mass distribution over all links. In order to perform a loose serial link coupling, the algorithm calculates the angle θ proportion from for all leading and all trailing links, by weighting its mass with the equations:

$$\theta_{leading}(\theta, |trailing\_links|) = \theta * (|trailing\_links| + 1)/|links|, \quad (4.3)$$

$$\theta_{trailing}(\theta, |leading\_links|) = \theta * (|leading\_links| + 1)/|links|. \quad (4.4)$$

Equations 4.3 and 4.4 can be interpreted as "the lighter part of the chain is rotated with a greater angle around its joint" and vice versa. Consequently, if angle θ in Figure 4.1(b) would have been applied at another joint, then weighting Equations 4.3 and 4.4 would have resulted in unequal angle values other than $\theta/2$.

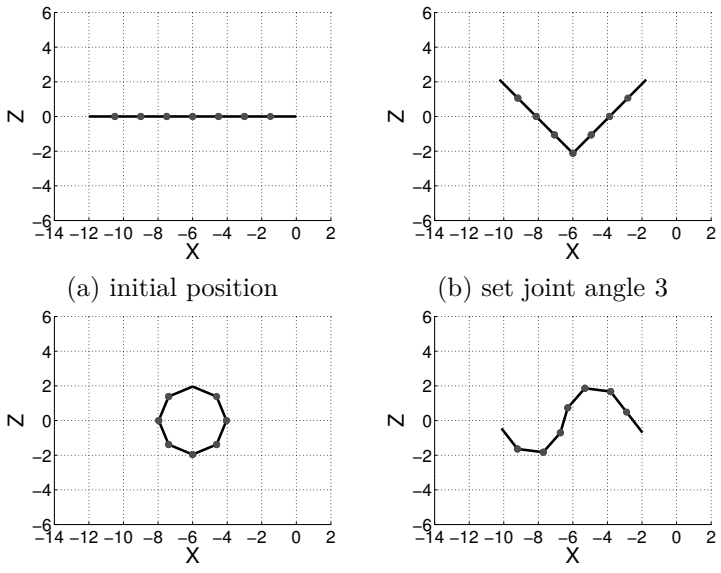
Figure 4.1(b) depicts the geometric displacement of a chain's links when applying an angle value $> 0°$ on joint 3. Due to demonstration purposes, it does not show a displacement of the center of mass (COM) which shall be compensated in order to show the accurate motion of a loose n-link serial chain. Consider that, if the COM would displace without terrestrial or hydrodynamical friction just by moving angles, then it would lead to a propulsion in an vacuum environment. This thought experiment is physically invalid and must be compensated by the algorithm. Therefore, the algorithm assumes that the COM stays always at the same position as defined by the initial chain position. This is implemented by storing the COM position (COM_{before}) before the links are rotated around a joint. After the rotation has been executed, the new COM (COM_{after}) is calculated and reset to COM_{before} by translating all link positions with $COM_{before} - COM_{after}$.

Algorithm 4.1 summarizes all steps for calculating the chains's new shape. The algorithm takes the current start and end positions of all links *links* and the desired angle change per joint $\Delta\theta_i$ as parameters. An angle change of $0°$ means no change. Hence, the angle of the joint stays the same as it was prior the algorithm call. Angle changes of $\neq 0°$ force a rotation of the joint.

The visualization is done by drawing lines from *linkstart* to *linkend* for all links. Figure 4.2 depicts the visualization of three different shapes in (b), (c) and (d) by using (a) as initial position. COM stays at the initial position $(-6, 0)$ in Figure 4.2(b), (c) and (d). Hence, no propulsion force is generated. Point $(-6, 0)$ is the result of averaging X and Y positions of all

Algorithm 4.1: Transforms the chain's shape with $\Delta\theta$ joint angles

1: ShapeTransformation($links, \Delta\theta s$) ▷ $link_i = (linkstart_i, linkend_i)$
Returns the updated links
2: **for each** $\Delta\theta_i$ **in** $\Delta\theta s$ **do** ▷ $i..jointindex$
3: $COM_{before} \leftarrow getCOM()$
4: $\theta_{leading} \leftarrow \Delta\theta_i * (|trailing\_links| + 1)/|links|$
5: $\theta_{trailing} \leftarrow \Delta\theta_i * (|leading\_links| + 1)/|links|$
6: $adjlink_{leading} \leftarrow RotateLeading(link_i, \theta_{leading})$
7: $adjlink_{trailing} \leftarrow RotateTrailing(link_{i+1}, \theta_{trailing})$
8: $leading\_links \leftarrow RotateTranslate(link_{i-1}..link_0, \theta_{leading})$
9: $trailing\_links \leftarrow RotateTranslate(link_{i+2}..link_{last}, \theta_{trailing})$
10: $links \leftarrow Concatenate(leading\_links, adjlink_{leading},$
11: $adjlink_{trailing}, trailing\_links)$
12: $COM_{after} \leftarrow getCOM()$
13: $links \leftarrow Translate(links, COM_{before} - COM_{after})$
14: **end for**
15: **return** links
16: **end**



(a) initial position (b) set joint angle 3

(c) circle (d) sine curvature

Figure 4.2: Four different chain shapes. (a) uses joint angles $\theta s = (0°, 0°, 0°, 0°, 0°, 0°, 0°)$ and defines the initial position for (b), (c) and (d). (b) sets joint angle θ_3 to 90°. (c) forms a circle by setting all joint angles to $360°/(|links + 1|)$. (d) utilizes the function $\theta_i = 1 * sin(0 + 2\pi/7 * i) + 0$ to calculate joint angles.

link positions $link_{start}$, $link_{end}$. In general, the visualization enables the user to test motion behavior and shape deformation algorithms. As previously mentioned, the visualization is also used to identify possible link collisions before starting a physics simulation.

Chapter 5

Machine Learning Methods

This chapter describes the two machine learning methods, genetic algorithms and artificial neural networks, applied in this research; details of specific experiments are described in Chapter 7. Genetic algorithms are used in two ways. First, they are used to evolve the robot's actuation parameters directly. Second, the artificial neural networks (a statistical learning model explained in more detail in Section 5.2) are generated by the NEAT library which employs genetic algorithms for evolution.

5.1 Evolutionary and Genetic Algorithms

Evolutionary algorithms (EAs) mimic natural evolution with the goal to generate effective solutions for computational problems [19]. They utilize a population of individuals to test many possible solutions, refining them over time through a combination of selection, mutation and crossover. Individuals that outperform others, propagate their "genes" through the population. A fitness function measures individual performance (also called fitness) deciding which individuals of a population are eligible for recombination with other individuals and/or mutation. Recombination generates a new individual (child) from the characteristics of two or more parent individuals. Mutation alters a child's characteristics at the gene level. The children produced this way, comprise a new population and the process repeats over generations.

Recombination and mutation are called variation operators. They primarily create diversity in a population. Selection pressure increases the fitness among the individuals in a population. Evolutionary algorithms are stochastic, which means that the genes of more fit individuals have a higher chance to be propagated to the next generation.

Genetic algorithms (GAs) are a subset of EAs. A GA is generally considered as an optimization method [19], although it was originally conceived by Holland [25] to study adaptive behavior. The goal is to find a set of

Algorithm 5.1: Simple GA work flow from [45] and [1].

1: Start with a randomly generated population of individuals.
2: Calculate the fitness of all individuals.
3: Repeat until n offspring individuals are generated:
4: Select parent individuals from the population.
5: Apply recombination operator with probability p_c to two parents in order to generate an offspring.
6: Mutate the offspring with probability p_m.
7: Replace the current population with all generated offspring individuals.
8: Increase the generation counter and go to step 2 if the number of maximum generations has not been reached.

parameters $x_1, ..., x_n$ which minimizes or maximizes a function $f(x_1, ..., x_n)$ (also called fitness function). These functions are usually nonlinear implying that each parameter can not be treated independently [63]. Specifically, the interaction of parameters may affect the output of function f. Goldberg [21] identified three main types for solving such problems: (1) calculus-based, (2) enumerative, and (3) random. A GA overcomes the shortcomings of calculus-based methods which depend on the existence of derivatives in the solution space and enumerative methods which lack in efficiency, by using randomness to guide a highly exploitative search in the parameter $(x_1, ..., x_n)$ space. A simple GA work flow is specified in Algorithm 5.1.

An individual is represented by a chromosome. A specific position in the chromosome is called an allele and describes a parameter x. Commonly used chromosome encoding forms are bit-strings (binary encoding) and real value sets. In this thesis, individuals represent sinusoidal parameters i.e., amplitude, frequency, phase shift. Individuals in Chapter 7 employ amplitude, frequency, phase shift alleles either separately per joint (Sections 7.1 and 7.2), or combined for all joints i.e., each joint has the same amplitude, frequency and phase shift value (Sections 7.2, 7.7, 7.8 and 7.10). The individual's representation highly affects the implementation of the variation operators. For example, Recombination of binary representations might be implemented as *n-point* crossover. In the simplest form (*one-point* crossover), two (or more) individuals swap their bit-string starting from a specific index (determined e.g., via a random number generator) until the end of the string. In contrast, real valued representations can be implemented by arithmetically combining two (or more) values at an allele, e.g. through calculating of an average. Another common implementation is the simulated binary crossover (SBX) proposed by Deb and Agrawal [13]. It utilizes a polynomial probability distribution for the calculation of the child's allele value, with the result that the SBX provides the same search power as binary crossovers for binary representations. Experiments in this thesis employ either one-point crossover by

swapping real valued parameter strings similar to a bit-string as described in Sections 7.1 and 7.2, or SBX in Sections 7.2, 7.7, 7.8 and 7.10. Mutation at a binary encoded chromosome flips a few bits in a bit-string, (e.g., 0000100 might be mutated to 0100100). The mutated position in the bit-string is determined randomly according to a uniform distribution. It is common for real value encoded individuals to employ a random number distribution in order to change the value itself instead of determining the index of an allele, as employed in experiments described in Sections 7.1 and 7.2. Deb and Goyal [14] proposed a widely utilized implementation with a polynomial probability distribution, which is employed in experiments described in Sections 7.2, 7.7, 7.8 and 7.10.

As previously mentioned, selection is used to increase the overall fitness in a population. Therefore, the selection of an individual is dependent on its fitness. One common selection method is *tournament* selection, which does not require a global ordering of the population according to its fitness. Therefore, it is conceptually simple and executes quickly. A number k of individuals is randomly chosen (with or without replacement) from the population to participate in the tournament. The best (highest fitness) of the k individuals is selected for variation. This process is repeated until enough individuals for the next generation's population have been selected. Tournament selection is utilized in all experiments that perform evolution in this thesis. The fitness function reflects the distance traveled in a simulation (either the Euclidean distance or the distance in X direction).

GAs are a powerful tool to search for solutions to complex problems by employing concepts from natural evolution. This method can also discover innovative (within the boundaries of the solution space) approaches to problems that are either not solvable by calculus methods or require too much computation time for an exhaustive search.

5.2 Artificial Neural Networks

Artificial neural networks (ANNs) are learning models inspired by the biological nervous systems of animals and humans [31]. They are applied as function approximators with a number of input and output signals. Specifically, an ANN is implemented as an interconnected network consisting of input, hidden and output neurons. The structure of the network is called its topology. Each connection between neurons has an assigned weight that indicates the importance of the connection and a direction of signal propagation (e.g. from $node_N$ to $node_{N+1}$). Hence, each node has input and output signals. In each time step, each node processes its input signals by accumulating the product of the input signal and its weight. Afterwards, a neuron-specific activation function (sigmoid function is commonly used [57]) is applied to calculate the output of a neuron. If the neuron is an input or

hidden layer neuron then the output is the input to another neuron at the next time step. Otherwise, it is an output neuron and therefore produces an output signal for the neural network at a time step. On robots, output signals are typically used to drive actuators. A memory can be implemented with recurrent node connections, but it also makes the ANN time variant. To summarize, the connection weight, the connections between nodes, and the activation functions characterize the ANN and can be adjusted (or evolved) in order to generate a different output signal from a given input signal. Learning methods provide input signals to an ANN, evaluate the generated output and adjust the ANN characteristics according to a cost function (e.g. difference between input and output signal value).

This thesis utilizes the NeuroEvolution of Augmenting Topologies (NEAT) library which uses GAs in order to evolve the parameters of an ANN [60]. Neuroevolution searches for a behavior in the solution space and belongs to the class of reinforcement learning methods. Specifically, the cost function is derived from evaluation in an external environment (a physics simulation environment in this thesis).

In NEAT an ANN is encoded as a linear representation of the network connectivity in the chromosome. Specifically, the genotype consists of *Node Genes* that indicate all input and output Nodes of the ANN as well as *Connection Genes* that represent the internal connection of the ANN. *Connection Genes* include a inbound and an outbound pointer as well as a real valued weight, a boolean enabled/disabled bit, and a innovation number. Mutation occurs by adding a new gene to the *Connection Genes* list. Crossover is implemented by lining up the genes according to their innovation numbers and merging all other genes. Competing conventions[1] are avoided by using the innovation number as historical marker (homology). NEAT also includes a mechanism for the protection of innovation by defining "species" within a population that share the same fitness value. Therefore innovative structures with low fitness values can survive. NEAT's minimizing topology approach prevents the ANN from undesired growth.

The NEAT implementation *MultiNEAT* is a C++ library with Python bindings which provides an API. Experiments in this thesis call API functions to initially configure the library (e.g. shown in Section 7.2), force the internal evolutionary calculation of one generational step, retrieve the evolved ANN and allow the injection of the fitness value determined via an ODE simulation.

[1] Competing conventions describe the problem of individuals which produce the same output value, but have different encodings.

Chapter 6

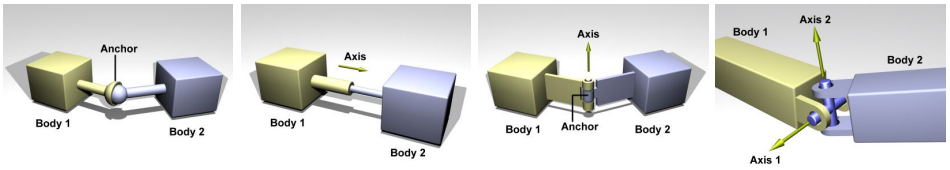
Simulation Environment

In order to reproduce behavior of a real world system on a computer, a physics-based simulation environment is used. This thesis utilizes the Open Dynamics Engine (ODE), a rigid-body dynamics library, to test hand-coded and evolved controllers for an animat in an aquatic environment. ODE is utilized by many institutions and research laboratories with applications in in robotics simulation (e.g., humanoid or biomimetic) [18] and comparison with other libraries like MATLAB [2]. Data recorded during simulation is used to analyze the motion and control of physical entities and to generate visualizations.

It is important to clarify and describe the influence of the term *reality gap* for work in the field of evolutionary robotics. The *reality gap* occurs when a well performing simulation designed controller is transferred to a real world robot. It may be observed that the controller has a *weaker* performance in reality compared to the performance in the simulation. Researchers address the *reality gap* by e.g., assessing an evolved controller with its transferability to the real world [29] or providing online self modeling mechanisms to adapt unforeseen conditions for aquatic robots [54].

6.1 Open Dynamics Engine

ODE is a free, open source physics simulation engine. It is written in C++ but it includes Python bindings [73]. ODE is often used to simulate articulated rigid bodies, e.g., legged creatures, in a virtual reality. The virtual reality is characterized by a user's physics settings and configurations. A user can, for example, create a terrestrial or an aquatic environment. The virtual world consists only of rigid bodies, which cannot *deform* or change in shape. Objects can optionally be connected by joints, and depending on the joint-type, the motion of two objects will be coupled. Objects in the simulation follow the Newtonian laws of physics [16]. This means that (1) if the net force applied on an object is equal 0 then its velocity remains constant,



(a) Ball-socket joint (b) Slider joint (c) Hinge joint (d) Universal joint

Figure 6.1: Four different ODE joints and their characteristics of force transfer between bodies. Reproduced with permission from [73].

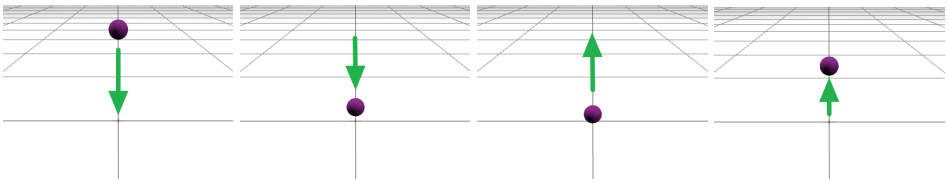
(2) an object is accelerated if the net force is unequal 0, and (3) a force from object A exists in opposite magnitude on object B if they touch each other. ODE uses a first order semi-implicit integrator to calculate each physics step [73], and body constraint forces (forces due to a joint) are calculated with an implicit[1] integrator. External forces, e.g. in order to accelerate an object, use an explicit[2] integrator. Inaccuracy in implicit integrators can result in a reduction of energy, while inaccuracy in explicit integrators increases the system's energy, which can lead to a so called "explosion" [73]. In order to create realistic simulations a user has to select an appropriate simulation time step and make sure that objects don't accelerate too fast. Der and Martius [15] tested ODE's usability in terms of stability and accuracy with a 16-link snake-like animat placed in a vacuum environment (and no gravity). They show that the simulation remains stable and accurate over hours of simulation time.

A rigid body has the following states: position, velocity, orientation, and angular velocity. Furthermore it has a mass, a position of the center of the mass, and an inertia matrix, which do not change over time. The shape of the body (e.g. sphere, or box) is defined as a geometric object, and is used for collision detection.

A joint defines a relationship between two bodies in ODE in order to connect two bodies. ODE allows the creation of five different joint-types: (1) ball and socket joint, (2) hinge joint, (3) slider joint, (4) universal joint, (5) hinge-2 joint. Figure 6.1 shows the most common joints (1-4). Joints either be passive or actuated with an angular motor. The angular motor allows the control of two bodies' relative velocities. For example, an ODE hinge joint applies a torque to each body to accelerate them to the desired angular velocity within one simulation time step. Angular motors are parametrized with a maximum force property when they are created. This means that the applied force/torque can be capped by the maximum force property, which effectively limits the maximum linear/angular velocities.

[1] ODE's implicit integration methods solve an equation by using the current state and a number of subsequent states of a system.

[2] Explicit methods calculate a system's state using only the current state of a system.



(a) $t = 0$ (b) $t = 0.75$ (c) $t = 1$ (d) $t = 1.5$

Figure 6.2: Visualization of four time steps ($t = 0$, $t = 0.75$, $t = 1$, $t = 1.5$) from Algorithm 6.1. The ball is colorized in purple and the ground plane is grid shaded. (a) depicts the initial position ($X = 0$,$Y = 1$,$Z = 0$) of the ball.

It is also necessary to declare which objects will be considered by ODE's collision algorithm. This is done with the so called *near_callback* function, which is called for all pair-wise combinations of bodies, and handles collisions by following user-defined scheme. Specifically, a user can define a bounce factor and the friction coefficient between two objects.

Russell Smith [73] provides this structure for a typical ODE simulation:

1. Create the simulation "world."
2. Create and initialize the placement of all bodies.
3. Create joints and their attachment to bodies.
4. Define joint parameters.
5. Define a collision "world" and collision objects.
6. Create joint groups.
7. While in simulation:
   (a) Apply forces to bodies and parametrize joints if necessary.
   (b) Call collision detection handler call and create collision points.
   (c) Execute ODE simulation step.
   (d) Remove all collision points after simulation step.
8. Destroy all simulation objects.

This structure has been used to create a simple ODE test program outlined in Algorithm 6.1. It defines a ball object in a terrestrial environment (gravity in the vertical direction) that falls from a height of 1 *unit* and bounces on the ground plane. No aerodynamic drag is applied. The downward acceleration (due to gravity) is transformed to upward motion when the ball hits the ground plane (*bounce* = 1). Therefore, the ball bounces without loss of momentum. A spherical body is defined to collide with the ground. A collision point is created at the lowest Y point of the sphere and the corresponding point on the plane in the collision handler. ODE then handles the collision between the sphere and the plane in the simulation step (*world.step*()). This procedure is shown in Figure 6.2.

Algorithm 6.1: ODE example implementation of a bouncing ball

```
1:  BouncingBall
2:      ▷ Create terrestrial environment
3:      world ← ode.World()
4:      world.setGravity((0, −9.81, 0))                    ▷ (Xaccel,Yaccel,Zaccel)
5:      ▷ Create ground plane
6:      space ← ode.Space()
7:      ground ← ode.GeomPlane(space, (0., 1., 0.), 0)     ▷ (s,(X,Y,Z),dist)
8:      contact_group ← ode.JointGroup()
9:      ▷ Create ground plane
10:     sphere ← ode.Body(world)
11:     sphere.setPosition((0., 1, 0.))                    ▷ (Xpos,Ypos,Zpos)
12:     mass ← ode.Mass()
13:     mass.setSphere(100, 0.1)                           ▷ (density,radius)
14:     ▷ Do simulation
15:     for t = 0; t < SIMUTIME; t += STEP do
16:         ▷ contact_group is set in near_callback handler
17:         space.collide((world, contact_group), near_callback)
18:         world.step(t)
19:         contact_group.empty()
20:     end for
21:     Destroy()
22: end
```

6.2 Aquatic environment

ODE does not provide an out-of-the-box method to run simulations with aerodynamic or hydrodynamic forces. A hydrodynamic model in conjunction with a buoyancy force is utilized to create an aquatic environment in ODE. Hydrodynamic simulations in this thesis utilize the hydrodynamic model implementation presented in [47] and [46] (described by Algorithm 6.2). Specifically, a counteracting force is applied to each face of a moving object in the simulation. The force is calculated in direct relation to an object's velocity. Therefore, this model is applicable for slow moving objects only, e.g., robotic fish from [54] and [47]. Buoyancy is defined to eliminate gravity in all ODE simulations. Hence, acceleration in the Y direction is parametrized with $-9.81\ (m/s^2) + 9.81\ (m/s^2) = 0\ (m/s^2)$. An object keeps the same vertical position if no forces in the Y-direction are applied.

Algorithm 6.2: Adapted hydrodynamic model implementation from [58].

```
1: for all body do
2:     lin_vel ← getLinearVelocity(body)
3:     body_rot ← getBodyRotation(body)
4:     for all face do
5:         area ← face_area
6:         norm ← (face_normal ∗ body_rot)
7:         force ← norm ∗ lin_vel ∗ area ∗ drag_coeff
8:         if force > 0 then
9:             addForce(force)
10:        end if
11:    end for
12: end for
```

6.3 Visualization

A qualitative validation of simulations is vital for interpreting the recorded simulation data. For instance, one might want to compare two simulations by just measuring an object's total distance traveled. A simulation "explosion," as described in Section 6.1, would clearly disrupt the result of an object's distance traveled. The "explosion" would be observable in the visualization and actions can be taken to avoid or correct the problem. Moore et al. [48] provide a WebGL[3] browser application named *WebGl-Based Visualizer* for visualizing the outcome of a simulation. It augments a simulation from an adjustable point of view in a web browser rather just observing a generated video. Zooming, rotating, play forward/backward, colorizing objects, setting time and playback speed are the main capabilities of this application. A demonstration of the application can be found at http://jaredmmoore.com/WebGL_Visualizer/visualizer.html. Figure 6.3 shows the visualization of a hexapod[4] simulation.

[3] WebGL is a JavaScript library that allows the rendering of graphics within a web browser.

[4] A hexapod is a six legged robot.

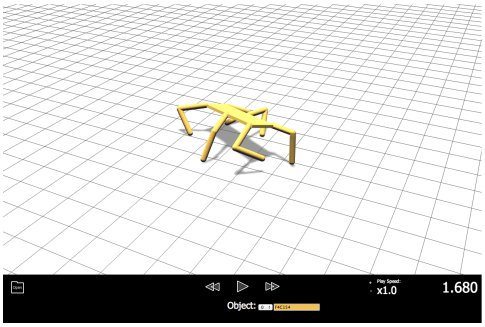


Figure 6.3: Visualization of a hexapod with the *WebGl-Based Visualizer*. Reproduced with permission from [48].

Chapter 7

Experiments

Experiments conducted employ GAs to optimize parameters associated with human designed control strategies. The overall task of object transportation consists of the following three consecutive subtasks: Experiments are divided as follows in order to the fulfill the goal of this thesis:

1. Approach the target object from the robot's initial position.
2. Grasp a target object.
3. Approach the final destination with the captured target object.

The robot has full information relevant to the task in all experiments. Specifically, it has the global positions of the target, its links and the destination region. Furthermore, the robot uses the current angular position of its joints. The utilization of simulated sensors would not change the overall behavior of the robot. However, they would increase the complexity of the experiments and are therefore omitted.

Experiments are presented in chronological order. First, basic locomotion in the aquatic environment is tested. The experiment in Section 7.1 asserts the feasibility of employing GAs with a robot in the computer simulation environment described in Chapter 6. Furthermore, parameter ranges for sinusoidal locomotion (from [56]) are determined in this experiment. Locomotion with an evolved ANN and morphology-related parameter (input frequency for the ANN, a more detailed explanation is provided in the experiment description) is analyzed in Section 7.2.

Second, more complex movements are evaluated and environmental parameters are adjusted. The experiment in Section 7.3 tests a turning gait with the generation of forward momentum, and reveals an effective algorithm to approach a target position. Environmental parameters are adjusted in Section 7.4 in order to provide a realistic aquatic environment. Section 7.5 tests a simple stopping motion, by reverse actuating the robot's joints. Turning behavior tested in Section 7.6 provides the opportunity for subsequent experiments to employ turning without forward momentum.

Third, DEAP library is introduced for evolutionary runs in conjunction with a object grasping algorithm. The evolutionary library DEAP is utilized for evolving forward locomotion parameters in Section 7.7. This experiment asserts the usability of DEAP in conjunction with the employed aquatic simulations. Therefore, it is used in all subsequent experiments. Forward locomotion with two different robotic "heads" that may hold an object is evolved in Sections 7.8. The proposed object grasping algorithm from Section 3.4 is experimentally tested in Section 7.9. A composition of object grasping and evolved forward locomotion is presented in Section 7.10.

Finally, subtasks from previous experiments are composed in order to fulfill the goal of this thesis. The experiment from Section 7.11 combines forward locomotion, approaching a target and grasping in one task. Additionally, a more accurate target approach algorithm compared to Section 7.3 is presented. Section 7.12 shows a composition of all sub tasks in order to approach, grasp and deliver a target object to a destination region.

For the sake of completeness, the experiments can also be divided into experiments/treatment according to the robot's task:

- **Forward Locomotion**: Sections 7.1, 7.2 and 7.7,
- **Turning/Stopping**: Sections 7.3, 7.4, 7.5 and 7.6,
- **Grasping**: Section 7.9,
- **Payload Transportation**: Sections 7.8 and 7.10.

The explored behaviors are combined in Section 7.11 and 7.12 in order to fulfill the goal of this thesis by (1) approaching, (2) grasping, and (3) delivering an object.

Statistics are calculated according to established guidelines in the evolutionary computation community described by Wineberg [64]. Evolutionary fitness and evolved controller parameters are analyzed by assessing performance over multiple generations, with multiple replicates. A replicate is the repetition of the same encoded experiment with a unique seed value for the random number generator. Multiple replicates are used to avoid wrong conclusions from a lucky coincidence of coherent parameter values, which therefore cannot be generalized to compare an algorithm with others. In this thesis, the performance over generations is visualized with the mean over all replicates (per generation) and its confidence interval (CI) as well as the minimum-maximum range between the best and worst performing individual per generation. The mean in conjunction with its 95% CI is used to visually compare different algorithms or experimental setups. An intuitive interpretation of the CI is given by Cumming and Finch [12]:

> *The CI is a range of plausible values for the mean. Values outside the CI are relatively implausible.*

If the CIs of two data sets do not overlap, then it would be a rare occurrence that the means of the two data sets have identical values [64]. The data set

with the higher mean can be said to be perform better with a confidence level of 95% if the CIs of the data sets do not overlap [64]. The minimum-maximum range is calculated in order to show the worst/best performing individual of a replicate per generation. It is used to identify unexpected behavior of individuals, as described in Chapter 7. Independence tests are performed with the non parametric[1] Wilcoxon Rank-Sum Tests (WSR). Two samples are tested if they belong to the same population (null hypothesis) or not (alternative hypothesis) with a significance level of $\alpha = 5\%$. If the resulting $p - value$ of the WSR is smaller than or equal to the significance level, then the null hypothesis has to be rejected. A rejection of the null hypothesis means that in this case that two data sets do not belong to the same population. Therefore, statements like $dataset_1$ performs better than $dataset_2$ can be made.

7.1 Evolved Watersnake

This section describes the first evolutionary experiment performed with the tools described in Chapter 2. Therefore, the purpose of this experiment is to ascertain the feasibility of simulations from a software engineering perspective. In order to automate deployment and retrieval of results from the HPC, tools and scripts have been developed in addition to the simulation and evolution software. Initially, a hand-coded (without the usage of a library) conventional GA is used to maximize the robot's linear velocity.

Purpose
This experiment shows the feasibility of conducting evolutionary runs with an automated toolchain. Evolutionary experiments are prototyped on a local PC and then deployed on the HPC. Result data is retrieved after the evolutionary run ends and then used to analyze the experiment. Forward-swimming locomotion parameters for an eight-link robot are evolved and limits of parameter ranges are determined in order to define the borders of the solution space in subsequent experiments.

Implementation
A simulation is performed by creating an articulated robot in an aquatic environment with a time step of 0.005 seconds and a total simulation time of 20 seconds. The robot comprises eight links and seven joints, Each joint is characterized as triplet of parameters $(\alpha, f, -\beta)$ (see Equation 3.4), where $\omega = 2 * \pi * f$ is the angular velocity of a joint. It is important to note that β is implemented with a negative sign, which dictates the direction of travel (this will be elaborated in a subsequent section). The robot's link density

[1] Non parametric means that the statistic does not assume an underlying distribution of the data, e.g. Gaussian distribution.

$\rho = \frac{m}{V}$ is defined to be 0.1 ($mass\_units/length\_units^3$) which leads to a total mass m of $(8 * 0.1 * (1.5 * 0.5 * 0.5)) = 0.3$ ($mass\_units$). Links are placed consecutively after each other in positive X direction. The front face (outer position of the first element) is set to $X = 0$ and the rear face (outer position of last element) to $X = 12$. Furthermore, the Y position is lifted by 0.5 and the Z position is shifted by 0.25. The maximum joint force is set to 10 ($force\_units$).

This GA implementation is parametrized with a population size of 120 individuals and conducts evolution over 1000 generations. Each individual's genome consists of 21 alleles, where each allele is defined as a robot's joint parameter. Each joint parameter triplet (α, f, β) is placed consecutively in the form: α_0, f_0, β_0, α_1, f_1, β_1, ..., α_6, f_6, β_6. Amplitude parameters (α) range from 0.001 to 2, frequency parameters (f) from 0.001 to 4 and phase shift from 0 to $2 * \pi$ (β). The genome's alleles are initialized according to a uniform distribution within the corresponding parameter range. For each individual, an ODE simulation is conducted with the given parameters. Specifically, the first joint actuates according to Equation 3.4 with the first three alleles in the genome and so on. Fitness is evaluated with:

$$fitness(d) = sign(endposition_X) * \sqrt{d_X^2 + d_Y^2 + d_Z^2}, \tag{7.1}$$

where d is the distance between starting position and end position in the coordinate system and X, Y, Z indicate the distance in the respective dimension. This fitness function rewards for the total Euclidean distance traveled in positive X direction and penalizes movement in negative X direction. For each new child, two parents are chosen from the population with tournament selection (with replacement) and a tournament size of $k = 2$ for each new child. One-point crossover is performed with 100% crossover probability. The crossover point is determined according to a uniform distribution in the 21-parameter genome. This means, that the crossover point may be positioned within a triplet that characterizes the joint. The crossover operation concatenates all first parent's parameters before/equal the crossover point and all second parent's parameters after the crossover point. It returns an intermediate individual which is then processed in the mutation operation. The mutation probability is set to 1% per allele. Mutation is performed according to a uniform distribution within the previously mentioned joint parameter range.

Results

The fitness progress in Figure 7.1 shows that the mean of the best individuals from each of the 18 replicates converges to 53.34 *units*, whereas the mean of the average individuals converges to 50.18 *units*. The 95% CIs overlap each other, but the best individuals perform significantly better than the average individuals ($p < 0.05$ with WRS). Maximum values (upper dashed

line) indicate the best fitness over replicates and minimum values (lower dashed line) represent the worst fitness respectively. Therefore, the best performing parameter configuration can be found at the highest maximum fitness value from the best performing individuals in this case 68.07. This evolutionary algorithm finds the best joint parameters in the last generation with a fitness of 68.07. Furthermore the minimum-maximum range is used to check plausibility of the evolutionary run. It indicates the performance range of the replicates. The actual maximum values of the best performing individuals are greater than the average individuals, which is required for the evolutionary run to be plausible.

Mean, CI and minimum-maximum range performance values show a fast increase in fitness within the first 50 generations. Hence, initial randomized parameters perform worse than the adjusted parameters after generation 50. The rise of all performance values assert the effectiveness of the GA. In contrast, a poorly configured GA may show that fitness values stay at the same level or vary widely. This would happen, for example, if the crossover or mutation operators do not lead to an effective variation of the individuals' genomes. A small but steady fitness increase after generation 50 shows that the GA cannot rapidly narrow the solution space. This may be caused by the evolution of 21 different parameters. The GA constantly improves the configuration of the 21 parameters over 1000 generations. This indicates that the GA quickly finds "good" solutions, but has difficulties in finding the best solution.

Figure 7.2 shows the parameter distribution of the best individuals per replicate in generation 999. Parameters α, f and β show no tendency to reach their upper or lower limits given the parameter ranges defined in the GA implementation. This indicates that the chosen parameter range has reasonable borders for the GA's solution space. Figure 7.2(a) depicts that amplitude values at the front of the robot (joints 4, 5 and 6) are greater than those at the end (joints 3, 2, 1 and 0). This means that the front of the robot exhibits more motion than at the end. In other words, a semi-rigid tail is evolved. Frequency values in Figure 7.2(b) are mainly distributed around 2.4 (Hz). Values at the front of the robot show a narrower distribution to 2.4 (Hz) than those at the end of the robot. This can be explained by the fact that amplitude values at the end of the robot have a smaller amplitude. Hence, the impact of the frequency to the locomotion behavior at the end of the robot is smaller. Figure 7.2(c) depicts the distribution of phase shift β values around π. Phase shift values (multiplied by joint index) determine the spatial offset of a sine wave (see Section 3.2.3) and correspond to the direction of the robot's movement. In contrast to the animat in Chapter 3, β is implemented with a negative sign, so values around π result in movement in forward movement in positive X direction.

This effect can be observed in Figure 7.3, which shows the movement of the evolved robot at different time steps. The actuation of joints with

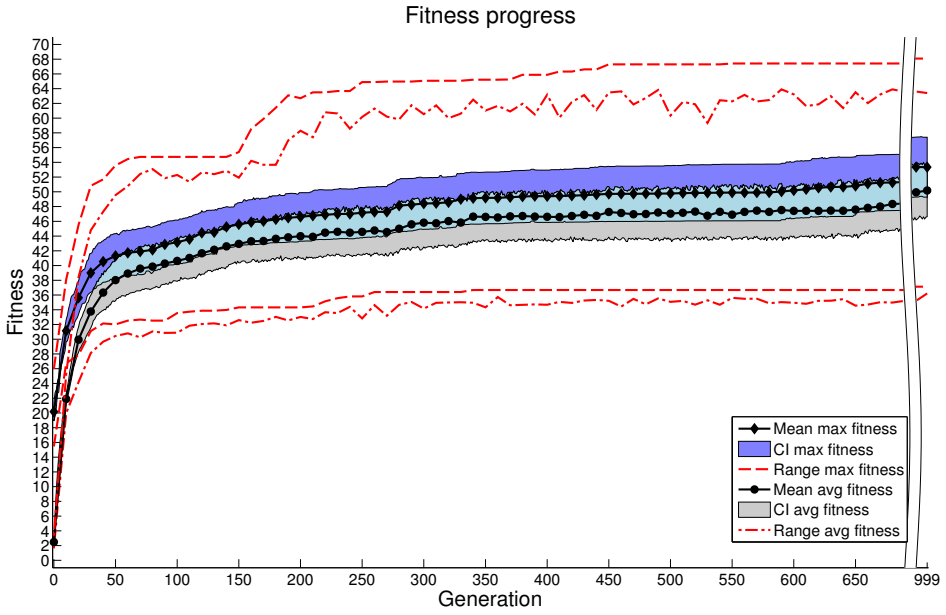


Figure 7.1: Evolutionary fitness progression for the forward locomotion experiment with a robot that consists of seven joints. 21 joint parameters are evolved as described in Section 7.1. The mean of the best individual per generation (*Mean max fitness*) and the mean of the average individual per generation (*Mean avg fitness*) over 18 replicates are significantly different ($p < 0.05$ with WRS). Shaded areas represent the 95% confidence interval. Red dashed lines describe the best and the worst performing individual per generation over all 18 replicates. The area between generations 699 and 951 has been omitted, because it contains no relevant information (although all performance indicators are constantly increasing).

evolved parameters from the best individual of generation 999 in replicate 6 (fitness 66.00), shows a movement in positive X direction. One can see in Figure 7.3(b)-(f) that the joints at the end of the robot (left part of the robot) behave similar to a fish tail.

Conclusion

This experiment has been conducted with an automated toolchain. Other experiments described in this thesis rely on scripts developed for this experiment. Specifically, an experiment revision gets checked out from a repository on a local PC zipped and transferred to the HPC. A script on the HPC automatically runs the experiment and prepares the resulting data for collection from a specified directory. Afterwards, the results are gathered with

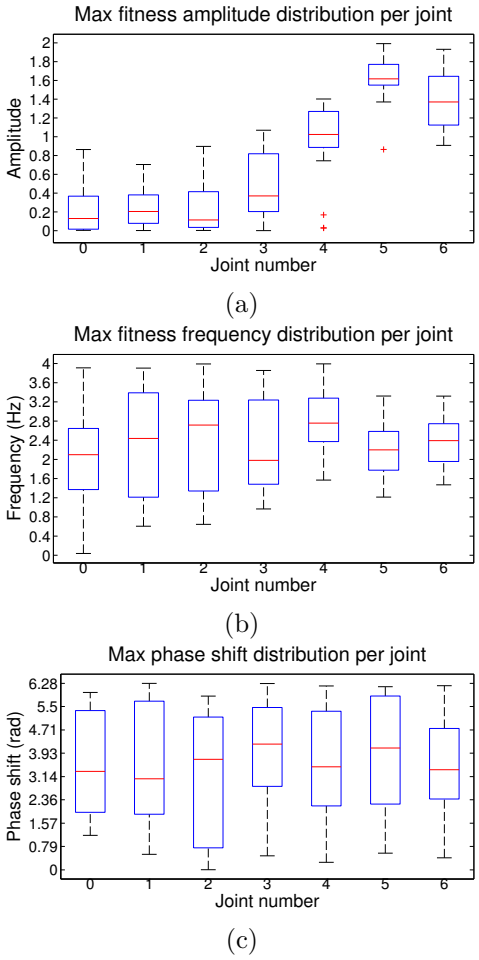
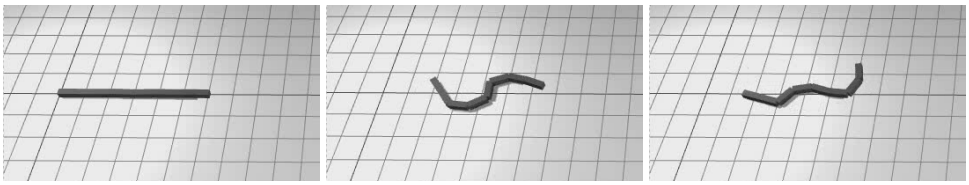


Figure 7.2: Parameter distribution of all best individuals per replicate in generation 999. 21 parameters are evolved for a seven joint robot. Seven amplitude (α) (a), seven frequency (f) (b) and seven phase shift (β) (c) parameters are shown within its according parameter range defined in Section 7.1. Joint number 0 is situated at the back of the robot and 7 at the front.

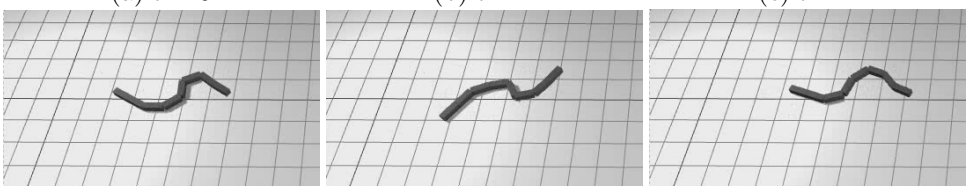
an SFTP[2] client. Visualizations have been generated with the visualization tool described in Section 6.3.

The experiment's fitness progress shows a quick increase at the beginning followed by a slow and steady convergence until over the remaining generations. The quick fitness increase indicates a dependency between the parameters. The relatively slow and steady convergence suggests that the solution space is too big for this GA implementation. Hence, a reduction of

[2]The secure file transfer protocol (SFTP) allows the transmission of data between computers.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$



(d) $t = 3$ (e) $t = 4$ (f) $t = 5$

Figure 7.3: Example of an evolved eight link forward swimming robot with the GA implementation described in Section 7.1. (a) shows the initial position of the robot. Then (b)-(f) depict the temporal progress of the robot at different time steps. A sinusoidal wave propagates through the robots body which causes movement to the right. The actuation of the outer left joint causes its adjacent links to act as tail. A video can be seen at https://youtu.be/OFid7-1-3u8.

21 parameters for evolution will be considered in subsequent experiments. The distribution of evolved parameters shows a reasonable choice for the solution space borders. Fundamentally, this experiment demonstrates that evolution can successfully generate parameters for forward movement.

7.2 ANN Watersnake

This experiment utilizes an ANN for the joint actuation of a robot. ANNs are evolved with a modified version of the NEAT algorithm (see Section 5.2) which includes variation operators for morphology-related parameters. Analysis of this experiment leads to the conclusion that ANNs do not provide advantage when compared with the direct joint actuation encoding from Section 7.1.

Purpose
This experiment shows the feasibility of actuating a robot with an evolved ANN in conjunction with an evolved oscillating input signal (frequency f for the ANN input signal).

Implementation
A simulation similar to Section that in 7.1 is performed by creating an articulated robot in an aquatic environment with a time step of 0.005 seconds

and a total simulation time of 20 seconds. In contrast to the simulation defined in Section 7.1 joint actuation values are calculated and set every fourth time step in order to increase simulation speed. Therefore, actuation values are updated every 0.02 seconds. The robot consists of eight links and seven joints where each joint is actuated according to the output signal of an ANN. Its density is set to 6.25 ($mass\_units/length\_units^3$) which results in a total mass of 5 ($mass\_units$). A robot's link dimensions are exceptionally set to $1.25 \times 0.5 \times 0.5$ $units$ (standard dimensions for this thesis are defined in Chapter 3). Links are placed consecutively after each other in negative X direction. The front face (outer position of the first element) is set to $X = 4.375$ and the rear face (outer position of the last element) to $X = -4.375$. Therefore, the COM of the robot is situated at $X = 0$. Furthermore, the robot's Y position is lifted by 1 $units$ whereas the Z remains at 0 $units$. The maximum joint force is set to 100 ($force\_units$).

This experiment utilizes an ANN controller in order to actuate the robot's joints. Therefore, the ANN has seven output signals (one for each joint). Input values consist of the robot's seven current joint angles states, a sinusoidal wave form, and a constant bias of 1 as input signals. The sine wave is defined as:

$$input(f, t) \; = \; sin(2 * \pi * f \; * \; t), \tag{7.2}$$

where t is the elapsed simulation time and f is a fixed frequency defined at the beginning of a simulation.

In this experiment, frequency (f) is seen as a morphology parameter that characterizes the robot and the ANN as brain that controls the actual joint actuation. Lessin et al. [34] outline the importance of evolving the morphology of a robot and its brain in parallel. They show that the evolution of a robot's morphology together with its brain produces systems that significantly outperform robots whose morphology and brain are evolved separately. To this end, the ANN and the frequency are evolved in parallel for this experiment. The genome is configured to consist of two parts: `ANN` and f. One part represents the ANN and the other part is a scalar frequency (f) value. This experiment utilizes two different variation methods in parallel during an evolution step in parallel. First, NEAT is used with the parameters outlined in Table 7.1 for the variation of the ANN part in the genome. Second, custom crossover and mutation operators are used for the variation of the frequency part in the genome. Specifically, the crossover operator uses one point crossover with a 100% probability. It is implemented to return either the scalar value of the first or the second individual according to a uniform distribution. Mutation of an individual is performed with 10% probability, sampling values from between 0.01 and 4 using a uniform distribution.

The evolutionary run is parameterized with a population size of 120 individuals and conducts evolution over 1000 generations. NEAT is initialized

Table 7.1: NEAT paramters for the experiment described in Section 7.2.

Parameter	*Value*	*Parameter*	*Value*
NN input values	9	NN output values	7
CompatTreshold	50	CompatTresholdModifier	0.3
YoungAgeTreshold	15	SpeciesMaxStagnation	1000
OldAgeTreshold	35	MinSpecies	1
MaxSpecies	25	RouletteWheelSelection	False
RecurrentProb	0.25	OverallMutationRate	0.33
MutateWeightsProb	0.90	WeightMutationMaxPower	10
WeightReplacementMaxPower	50	MutateWeightsSevereProb	0.5
WeightMutationRate	0.75	MaxWeight	20
MutateAddNeuronProb	0.4	MutateAddLinkProb	0.4
MutateRemLinkProb	0.05	CrossoverRate	0.4
ActivationFunction	SIGMOID	seed	0
RandomizedWeights	true	randomrange	1.0

with the parameters in Table 7.1 to process 120 individuals and zero hidden nodes at the beginning. The parameter f is initialized with a sample from a uniform distribution in the range between 0.01 and 4. Then, an ODE simulation is executed with a robot which is parameterized with a genome. Each individual's fitness is evaluated with Equation 7.1 from Section 7.1 and is then passed to NEAT, which internally performs an evolutionary step. NEAT triggers signals that indicate which individuals have been modified during an evolutionary step. All individuals that have been modified in NEAT are used for the variation of frequency (f) with the crossover and mutation operators previously mentioned. Afterwards, the genome is evaluated again and the next evolution step is performed.

Results

The fitness progress in Figure 7.4 shows that the mean of the best individuals of 18 replicates converges to a fitness of 10.09 *units*, whereas the mean of the average individuals converges to 6.03 *units*. Best individuals perform significantly better than average individuals ($p < 0.05$ with WRS). A direct comparison between the fitness values from this experiment with the experiment described in Section, the 7.1 is not applicable, due to the differences in the simulation setup (e.g. masses and sizes of links). In contrast to the experiment in Section 7.1 95% CIs do not overlap each other and exhibit smaller areas. Furthermore, minimum-maximum ranges do not overlap each

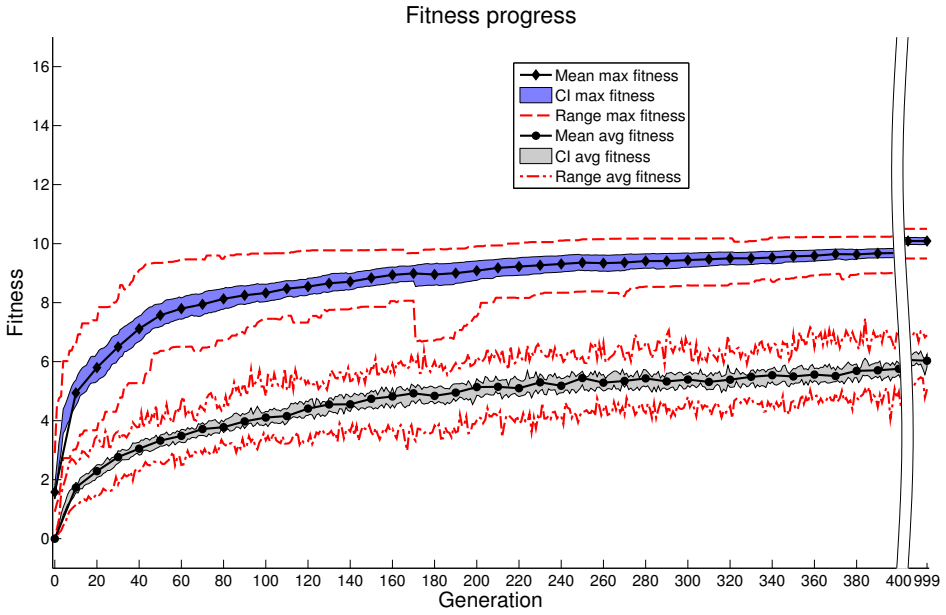


Figure 7.4: Evolutionary fitness progression for forward locomotion experiment with a robot that consists of seven joints. Joints are actuated by an ANN. The mean of the best individual per generation (*Mean max fitness*) and the mean of the average individual per generation (*Mean avg fitness*) over 18 replicates are significantly different ($p < 0.05$ with WRS). Shaded areas represent the 95% confidence interval. Red dashed lines describe the best and the worst performing individual per generation over all 18 replicates. The area between generations 400 and 951 has been omitted, because it contains no relevant information.

other after generation 10 and are situated narrower to its corresponding mean fitness values.

Figure 7.5 illustrates the ANN's input sine frequency progress for best performing individuals. It shows a convergence to 0.63 (Hz). A wide minimum-maximum range and CI between generation 30 and 247 indicate a shallow search process in the solution space. Different frequencies (f) are tested and values between 0.74 (Hz) and 0.5 (Hz) are found to provide the best performance. No significant performance increase can be seen in Figure 7.4 at generation 247. This means that few ANNs used frequencies differing from 0.63 (Hz).

The ANN's output angle values have been recorded during the simulation. This enables the analysis of the ANN's temporal output as shown in Figure 7.6. Principal joint angle values are extracted via selecting the dominant frequency from a Fast Fourier Transformation[3] (FFT) generated

[3]A Fast Fourier Transformation is used to extract frequency information from a time

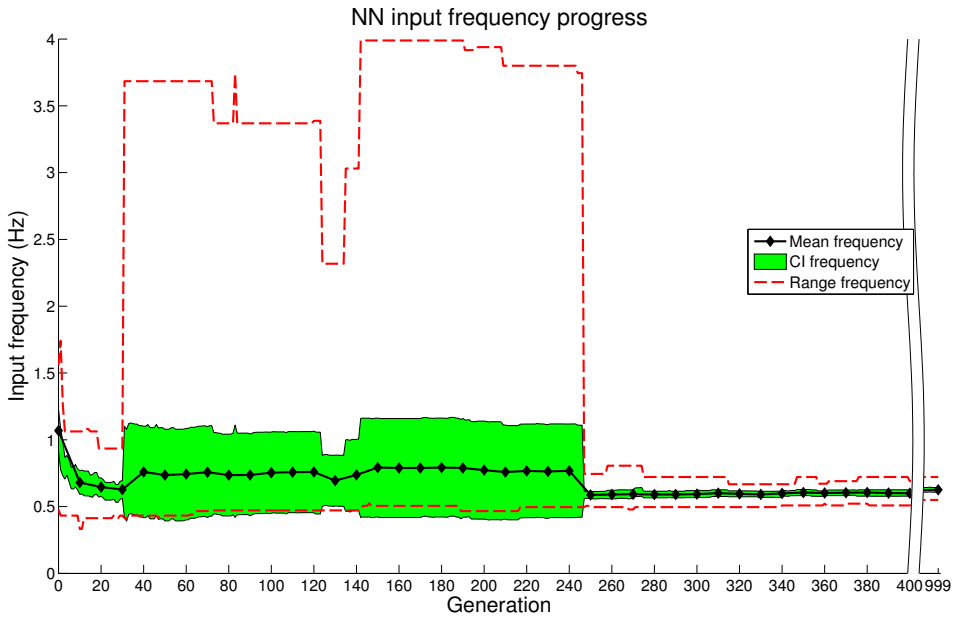


Figure 7.5: Best individual's ANN input frequency progress for the sine function formulated in Equation 7.2. Its mean values over 18 replicates are shown accordingly with its 95% confidence interval and its minimum-maximum range. The area between generations 400 and 951 has been omitted, because it contains no relevant information.

power spectrum (`sample time = 0.02` seconds). The principal frequency is then used with Equation 7.2 in order to print the output of a reference sine function. This reference sine function enables the is used for a qualitative evaluation if this one principal frequency approximates the course of the output joint angle. The best individual from replicate 0 at generation 999 has been chosen for this evaluation, because other best individuals exhibit similar behaviors. Principal frequencies calculated for joint 0 to 6 all result in 0.59 (Hz). A frequency of 0.59 (Hz) is used to depict green reference signals in Figure 7.6. The green reference signal approximates the actual joint angles in blue without a phase shift. This leads to the conclusion that the evolved output signals follow a sinusoidal form with a principal frequency of 0.59 (Hz). The evolved principal ANN output frequency is slightly less than the evolved input frequency of 0.63 (Hz).

Figure 7.7 shows the movement of an ANN actuated robot at different time steps. In contrast to the experiment from Section 7.1, no rigid tail-like behavior at the last segments can be observed.

data set.

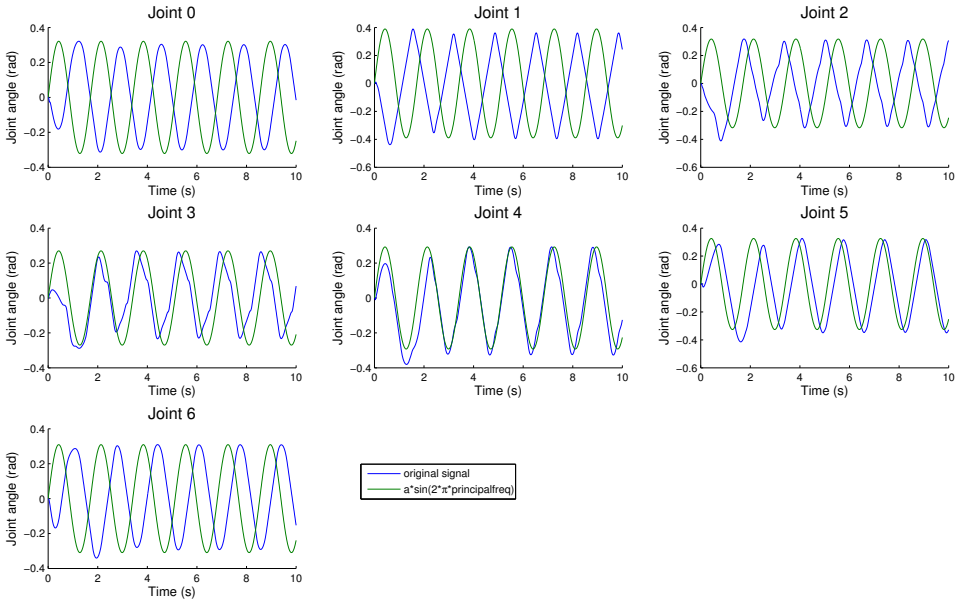


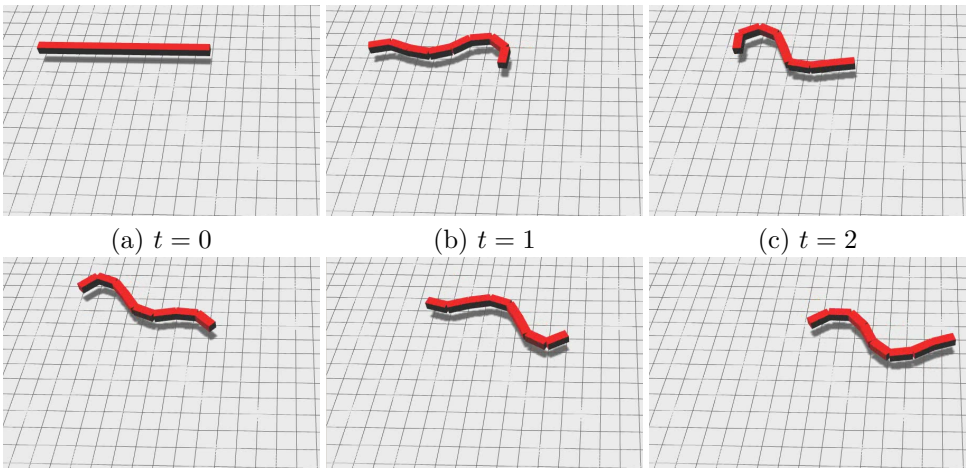
Figure 7.6: ANN output angle course (blue) per joint for the first 10 seconds of the simulation. The green angle course illustrates the sine function from Equation 7.2 where $f = 0.59$ (Hz) is the principal frequency per joint. This figure depicts the best individual from replicate 0 at generation 999 with fitness 10.01.

Conclusion
Results show the feasibility of evolving an ANN for robot joint-actuation. However, configuration of the NEAT algorithm requires more experimentation (i.e., more implementation time) than the direct encoding from Section 7.1. A wrong configuration of NEAT may result in sub-optimal output. Furthermore, due to the nature of ANNs, one cannot predict the output of the simulation, in contrast to the direct actuation encoding. Qualitative assessment of the visualizations show an effective forward swimming behavior similar to the experiment described in Section 7.1.

7.3 Turning

This experiment investigates the basic turning motion discussed in Section 3.2.3. In contrast to previous experiments, all joints use the same α, f, β and γ parameter values for actuation.

Purpose
This experiment shows the feasibility of steering the robot to a specific target position by adjusting the γ (bias) parameter from Equation 3.4. Specifically,



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

(d) $t = 3$ (e) $t = 4$ (f) $t = 5$

Figure 7.7: Example of an evolved eight link forward swimming robot actuated with an ANN as described in Section 7.2. (a) shows the initial position of the robot. (b)-(f) depict the temporal progress of the robot at different time steps. A sinusoidal wave propagates through the robots body which causes movement to the right. Furthermore, a slight movement downwards is observable. A video can be seen at https://youtu.be/Le1pLSut-UY.

it shall be validated if the robot can reach targets in all quadrants in an Euclidean coordinate system from the same initial position.

Implementation

In this experiment, a similar simulation setup to Section 7.2 is used. In contrast, the simulation time is set to 30 seconds and a target object is placed in the simulation environment. Specifically, the object is a sphere with radius 1 ($length\_unit$) at a designated position X, Z and a mass of 0.42 ($mass\_units$) Furthermore, the robot's joints actuate with the same parameters α, f, β from Equation 3.4, where $\omega = 2 * \pi * f$.

The steering algorithm is implemented by adjusting the γ parameter value from Equation 3.4. As the robot has no distance sensors, full information about the target's as well as the robot's position in the global coordinate system is provided to the algorithm. In order to steer the robot to the designated target position, the algorithm utilizes the state machine depicted in Figure 7.8. State transitions between the states: (1) *Move forward*, (2) *Turn left*, (3) *Turn right*, and (3) *Stop*, are initiated by either crossing a distance or an angle threshold. An Euclidean distance between the robot's "head link" ($link_0$) and the target position less or equal 2 ($length\_units$) changes the state to *Stop*. Otherwise (Euclidean distance is greater than 2 ($length\_units$)), the angle between the robot's direction of motion and the

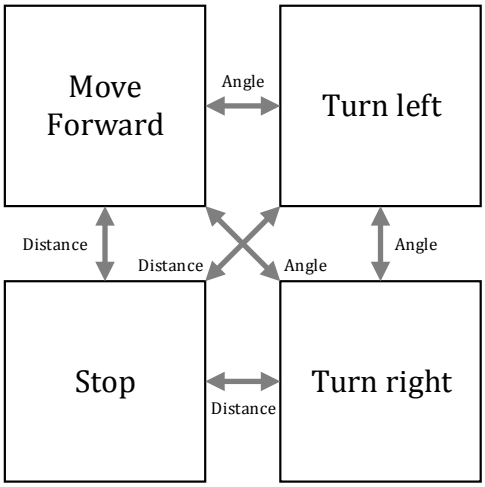


Figure 7.8: State machine utilized by the steering algorithm implemented in Section 7.3. The robot changes its state if a distance (distance between the robot and the target) or angle (angle between the robot's direction of motion and the target position) threshold is reached.

shortest path between its COM and the target is calculated. Specifically, two two-dimensional vectors a and b are defined. Vector a specifies a line between the robot's COM and its "head link" which determines the direction of motion. The latter vector b defines a line between the COM and the target position. Due to the fact that $arcsin(x)$ returns values in $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ and $arccos(x)$ $[0, +\pi]$, neither the cross product formula $a \times b = |a||b|sin\theta$ [68] nor the dot product formula $a \bullet b = |a||b|cos\theta$ [67] can be used to calculate an angle from 0° to 360° directly. Therefore,

$$angle(x) \;=\; arctan2\,\frac{sin(x)}{cos(x)} \tag{7.3}$$

is used which returns angles in $(-\pi, +\pi]$, where $cos(x)$ is the dot product between both vectors and $sin(x)$ the cross product. In this implementation, the cross product is determined by calculating the determinant of the two vectors as described in [77]. Afterwards the angle is smoothed over time with a moving average filter[4] (window size 300). The smoothed angle is used to choose between the three remaining states:

- $angle < 10°$: *Turn left*,
- $angle > 10°$: *Turn right*,
- *else*: *Move forward*.

[4] A moving average filter is a low pass filter used to remove high frequencies in a data set. It calculates average over the last *n..window size* data points.

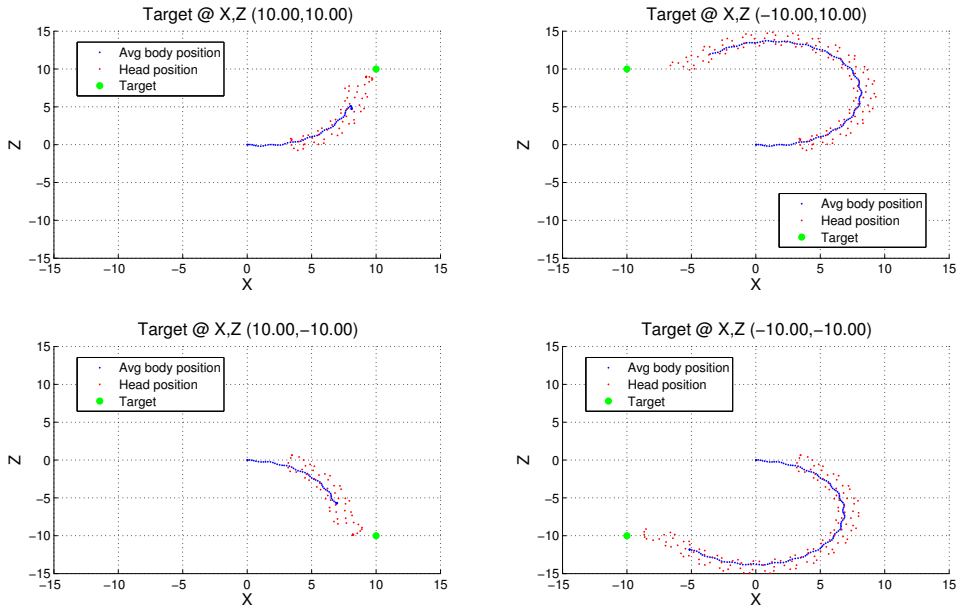


Figure 7.9: Robot approaching a target object at four different locations. Top left shows a target positioned in the first, top right a target in the second, bottom left a target in the fourth and bottom right a target in the third quadrant.

Each state defines its own hand-chosen α, f, β parameter values which are used by all joints to actuate. Following values have been used in this experiment:

- *Move forward*: $\alpha = 0.6,\ f = 1,\ \beta = \frac{2\pi}{7},\ \gamma = 0,$
- *Turn left*: $\alpha = 0.6,\ f = 1,\ \beta = \frac{2\pi}{7},\ \gamma = +0.05,$
- *Turn right*: $\alpha = 0.6,\ f = 1,\ \beta = \frac{2\pi}{7},\ \gamma = -0.05,$
- *Stop*: $\alpha = 0,\ f = 0,\ \beta = \frac{2\pi}{7},\ \gamma = 0.$

Parameters from *Stop* state halt the robot immediately without any drifting. Therefore, the robot performs locomotion only until the the target is reached.

Results

Four simulations have been started with target object locations $(X,\ Z)$: (1) $(10, 10)$, (2) $(-10, 10)$, (3) $(10, -10)$, (4) $(-10, 10)$, in order to validate if it the algorithm is capable of reaching targets in all four quadrants of the Cartesian coordinate system. As in Figure 7.9, the robot's COM, "head" link and the object's positions have been recorded during the simulations. The robot's COM exhibits a steady movement (blue), while the head position varies in a sinusoidal form. Furthermore, the figure shows that the target is reached in all four quadrants from the same initial position. In each sub-

figure, the robot's initial direction of motion points to the right (positive X). It changes to the left (negative X) during simulation time for targets positioned in the second and third quadrant.

Actuation of joints with the same α, f, β parameters results in consistent snake-like motion as detailed in Section 3.2.3, see Figure 7.10. A target object in the form of a ball is successfully approached in all four quadrants.

Conclusion
The results show that the adjustment of the γ parameter leads to turning motion. Targets in all quadrants have been reached. Furthermore, the snake-like motion of the robot is smooth and can be employed in subsequent experiments.

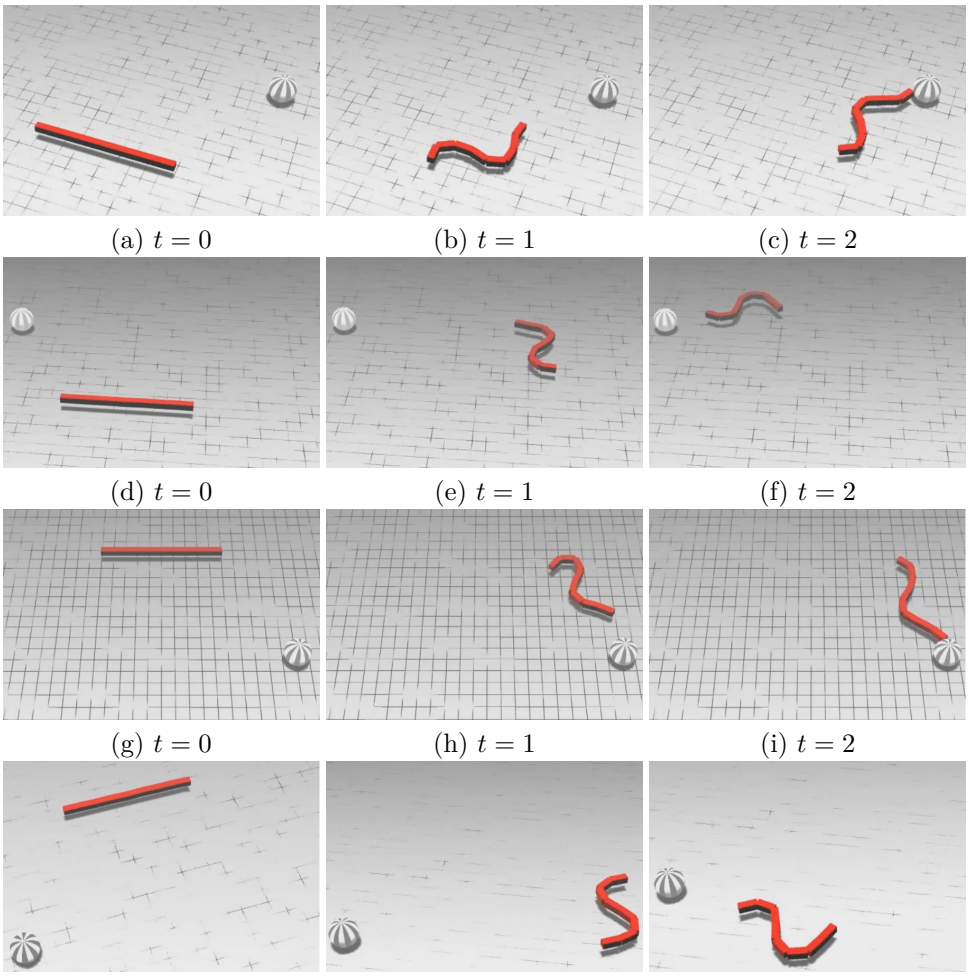
7.4 Drifting

The previous experiments did not consider the effect of a "realistic" momentum while the robot was moving in the aquatic environment. Straightening the robot's joints, i.e., setting all its joint angles to zero, led to an abrupt stop of the robot. Therefore, parameters of the robot or the aquatic environment need to be refined in order to provide more realistic motions, where momentum affects robot movement.

Purpose
This experiment produces drifting behaviors after the robot stops its actuation. Momentum increases as the robot accelerates. It is decreased by drag forces or when the robot reverses its direction of movement. The goal of this experiment is to change simulation properties in a manner that the robot shows a "realistic" drifting behavior when it stops. Additionally, a simple optimization for the robot's frequency parameter shall be conducted after the adjustment of the simulation properties.

Implementation
Behavior before and after changes on simulation properties are compared. To this end, two simulations are executed: One with simulation properties used in all previous experiments so far and the other with proposed changes.

A simulation similar to Section 7.1 is performed, but joints are actuated with the parameters α, f, β from Equation 3.4, where $\omega = 2 * \pi * f$. The robot starts to move from its initial position with the parameter values $\alpha = 1,\ f = 0.6,\ \beta = \frac{2\pi}{7}$. Hence, it moves in positive X direction until $t = 10$ (*seconds*) simulation time. After 10 seconds, the robot applies the parameters $\alpha = 0,\ f = 0.6,\ \beta = \frac{2\pi}{7}$. This means, that the robot sets each joint to 0°. No joint actuation occurs from this time on. Drag forces stop the robot by reducing its forward velocity.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

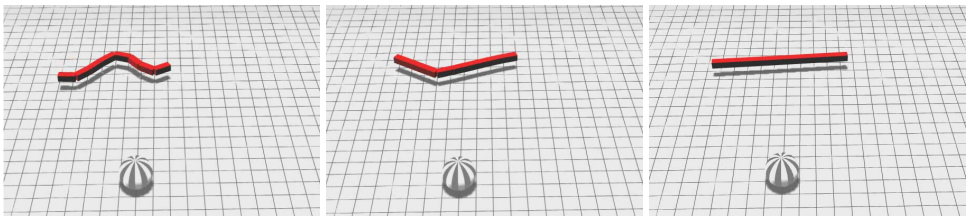
(d) $t = 0$ (e) $t = 1$ (f) $t = 2$

(g) $t = 0$ (h) $t = 1$ (i) $t = 2$

(j) $t = 0$ (k) $t = 1$ (l) $t = 2$

Figure 7.10: Example of an eight link robot that approaches four different target locations: (1) first quadrant at $X = 10$, $Z = 10$ in (a)-(c), (2) second quadrant at $X = -10$, $Z = 10$ in (d)-(f), (3) third quadrant at $X = 10$, $Z = -10$ in (g)-(i) and (4) fourth quadrant at $X = -10$, $Z = -10$ in (j)-(l). Each approach to the target is depicted with the robot's the temporal progress between its initial position and the end of simulation time. The robot steers to the target location and stops immediately when it is reached. A video can be seen at https://youtu.be/tdCnS7xTcu8.

Three changes are applied in order to alter the robot's drifting behavior. First, the parameterization of the hydrodynamics algorithm (Algorithm 6.2) has been changed such that forces are only applied to planes that are not "hidden" behind the robot's body (e.g., the face at the end in X direction of the robot does not cause friction as long as the robot moves in positive X direction). Second, the maximum joint forces have been increased



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.11: Visualization of a eight link robot that is put in stop mode after moving 10 seconds forward in positive X direction. Maximum joint forces are set to 100 ($force\_units$). (a) shows the robot while actuating its joints, (b) the beginning of the stopping behavior and (c) the stopped robot. A video can be seen at https://youtu.be/UNVnnD9f_PI.

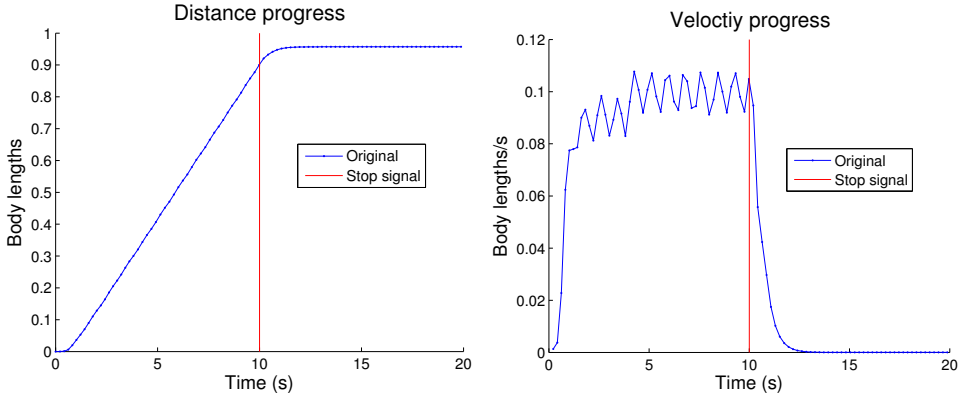
to 1000 ($force\_units$). This allows higher angular velocity for each joint (i.e., a higher momentum of the robot, because calculated joint angles may not be fully applied without this change if the required force is greater than 100 ($force\_units$)). Third, the robot's joint actuation frequency (f) parameter is set to 2 (Hz).

Frequency (f) optimization is implemented via a parameter sweep comparing the robot's distance traveled with different parameter values. Simulations are started with the parameter values: (1) 0.2 (Hz), (2) 0.4 (Hz), (3) 0.6 (Hz), (4) 1.0 (Hz), (5) 2.0 (Hz), (6) 3.0 (Hz), (7) 4.0 (Hz).

Results

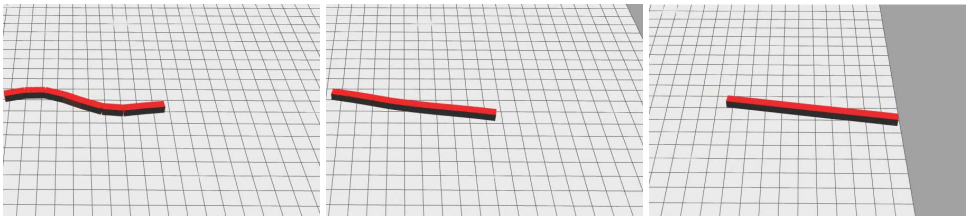
The robot's COM positions have been recorded during all simulations. A sphere has been placed at position $X = 10,\ Z = -10$ in order to enable a visual comparison of distances traveled.

A visualization of the simulation without property changes in Figure 7.11 shows a robot that moves near to the ball during the first 10 seconds of simulation time. Then it starts to apply the stopping behavior. The robot straightens out its joints and immediately stops. This behavior can also be seen in Figure 7.12(a). The figure shows that the robot experiences a short delay in distance at the beginning. Afterwards, the distance continuously increases until the stop behavior is applied. It reaches a distance of 0.9 ($body\_lenghts$). After the stopping behavior is applied, the robot increases its traveled distance for 2 seconds until it reaches a value of 0.96 ($body\_lenghts$). No change in distance can be seen after 12 seconds simulation time. The velocity progress in Figure 7.12(b) show a fast speed increase at the beginning of the simulation. After 1.42 $seconds$, an apparent variation of speed between 0.08 and 0.1 $body\_lenghts/seconds$ is observable. This suggests that maximum joint forces may be low for this robot. After 10 seconds, a steady decline in velocity can be seen.



(a) (b)

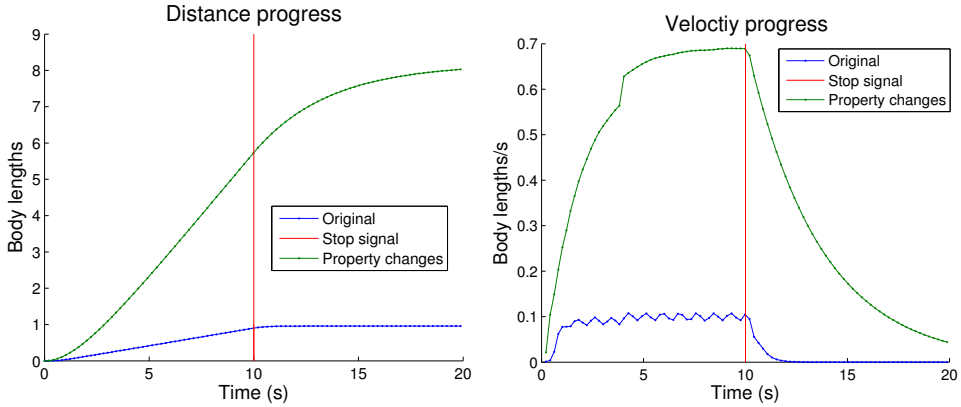
Figure 7.12: Distance progress (a) and velocity progress (b) of a eight link robot which is put into a stop mode after 10 seconds simulation time (red line). Distance is measured in *body lengths* and velocity in *body lengths/seconds*.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.13: Visualization of a eight link robot that is put in stop mode after moving 10 seconds forward in positive X direction. Only planes that are not hidden behind the robot's body are used for the hydrodynamics calculation. Maximum joint forces are set to 1000 ($force\_units$). (a) shows the robot while actuating its joints, (b) the beginning of the stopping behavior and (c) the stopped robot. The solid gray region represents an area beyond the measurement grid in the visualization. There is no difference in physics between the grid and the solid gray area. A video can be seen at https://youtu.be/nhTjkA9_CUs.

A visualization of the simulation with the mentioned property changes can be seen in Figure 7.13. The robot outreaches the sphere at $X = 10$, $Z = -10$ by far. In fact, it almost reaches visualization's measurement grid border at 75 ($length\_units$) while actively actuating, see Figure 7.13(b). Then the robot applies a stopping behavior and starts to drift. It crosses the border of the measurement grid as visualized in Figure 7.13(c). Compared to Figure 7.11, a drifting behavior is clearly observable. The different drifting behav-

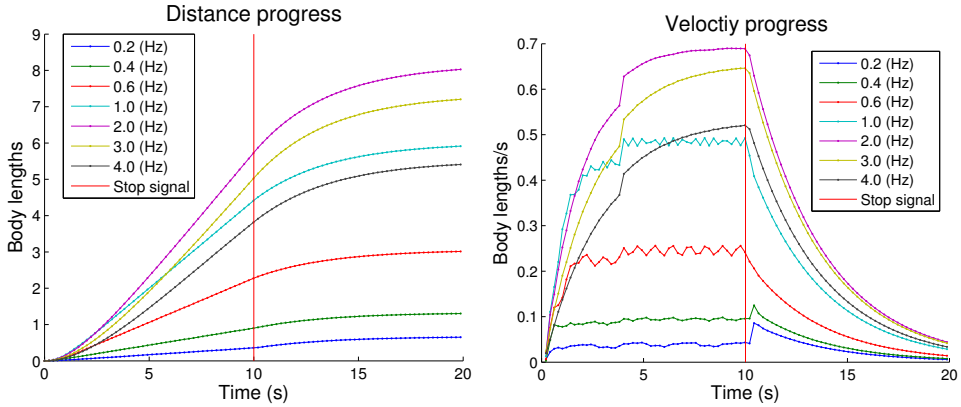


(a) (b)

Figure 7.14: Distance progress (a) and velocity progress (b) comparison of a eight link robot which is put into stop mode after 10 seconds simulation time (red line). The blue line indicates values from the original simulation without property changes and the green line values after property changes in the simulation.

ior between the two simulations can also be seen in Figure 7.14. Specifically, Figure 7.14(a) depicts the distance progress of a simulation without (blue) and with property changes (green). The simulation with property changes reaches a distance of 5.72 (*body_lengths*) after 10 seconds of simulation time. Therefore, it exhibits a distance more than 5 times greater than the simulation without changes. Also, drifting is more prominent after 10 seconds of simulation time. The robot shows a drifting behavior until the end of simulation time and it reaches a total distance of 8 (*body_lengths*). Velocity reaches a value of 0.69 (*body_lengths/seconds*) in Figure 7.14(b) and is therefore more than 6 times greater than in the simulation without changes. Furthermore, the velocity progress does not exhibit an apparent variation anymore. The green line indicates that the robot has still velocity, and therefore forward momentum, at the end of the simulation.

Seven simulations have been executed with different frequency (f) parameter values in order to conduct a simple optimization. The result is depicted in Figure 7.15. A frequency of 2 (Hz) causes the greatest distance traveled (5.72 (*body_lengths*) after 10 seconds) as well as the greatest velocity (0.69 (*body_lengths/seconds*) after 10 seconds). Frequencies of 0.2 (Hz) and 0.4 (Hz) show a spike at 10 seconds of simulation time, see Figure 7.15(b). After 10 seconds all joints are immediately straightened out to 0°, which causes forward velocity. Both frequencies have a low total forward velocity at this time, which causes the robot to propel for a short time. Other frequencies have higher velocities at this time in the simulation. Hydrody-



(a) (b)

Figure 7.15: Distance (a) and velocity (b) progress comparison of a eight link robot which is put into stop mode after 10 seconds simulation time (red line). Dotted lines represent different frequency (f) value settings.

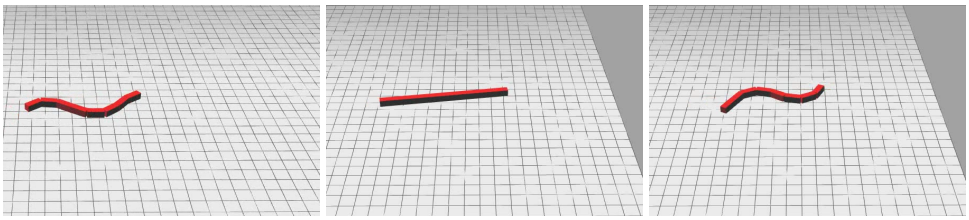
namic drag forces counteract this effect in this case.

Conclusion
The changes of the simulation properties: (1) increase of maximum joint forces, (2) call of the hydrodynamics algorithm with only "active" faces for drag forces, and (3) adjustment of the actuation frequency, cause an increase of the robot's velocity which leads to a higher momentum. The robot exhibits a drifting behavior when it is put in a stop mode. This behavior can be expected in an aquatic environment. A simple frequency parameter optimization shows that the highest velocity is reached with 2 (Hz).

7.5 Stopping

Stopping a robot can be achieved by providing a counteracting force to its momentum. Momentum is generated by increasing a robot's velocity, which is achieved by actuating its joints. If the joint actuation is then halted ("stop mode" in Section 7.4), drag forces counteract the momentum. Therefore the momentum reduces until the robot has come to a standstill. In order to reduce the amount of time required to come to complete stop, the robot's joints can be actuated in a "reverse" manner. This means that the robot propels itself in the opposite direction of its momentum.

Purpose
This experiment shows that a reverse actuation of a robot's joints lead to a counteracting force of its momentum (i.e., faster stopping).



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.16: Visualization of a eight link robot that applies reverse joint actuation while drifting in one direction. (a) shows the robot while actuating its joints, (b) the drifting robot, and (c) the robot applying joint actuation in reverse order. The robot ultimately comes to a standstill after actuating its joints in reverse order. A video can be seen at https://youtu.be/-g_yagHO8us.

Implementation

The simulation consists of an articulated robot in an aquatic environment similar to Section 7.2. Changes from Section 7.4 are applied in order to provide a "realistic" aquatic simulation environment. The robot actuates its joints with the same parameters α, f, β from Equation 3.4, where $\omega = 2{*}\pi{*}f$. Maximum joint forces are set to 10000 ($force\_units$).

The robot starts to move from its initial position with the parameter values $\alpha = 1$, $f = 2$, $\beta = \frac{2\pi}{7}$. It moves in positive X direction until 5 ($seconds$) simulation time. Then it drifts by straightening out its joints between the time stamps 5 and 9 *seconds*. Subsequently, the robot actuates its joints in the opposite direction by reversing its joint order from 9 to 9.25 *seconds*. Afterwards, it straightens out its joints again.

Results

Forward actuation at the beginning of the simulation is similar to the experiment described in Section 7.4. The robot almost reaches the visualization's grid border at 75 ($length\_units$) as shown in Figure 7.16(a). Then it drifts for 4 seconds. The end of the 4 seconds drifting is depicted in Figure 7.16(b). Afterwards, the robot actuates its joints in reverse order (see Figure 7.16(c)) for a short period of time and straightens out its joints again. The robot's forward momentum is reduced. This means that the robot can come to a standstill in less time than the drifting robot from Section 7.4.

Conclusion

The time until the robot comes to a standstill can be reduced by actuating its joints in reverse order as long as moment in the opposite direction exists. In this experiment, the time span of reverse actuating has been adjusted by executing the simulation several times. For each simulation the time span has been reduced until the robot showed a stand still behavior in the

visualization.

7.6 In situ Turning

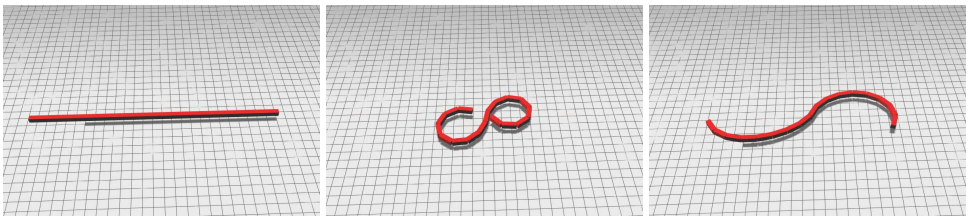
The turning motion described in Section 7.3 adjusts the γ parameter value of Equation 3.4. Scuh a turning motion generates forward velocity in the robot's direction of movement. In contrast, the experiment described here investigates a turning motion without the generation of forward (or reverse) velocity. Thereby expanding the number of experimentally tested behaviors..

Purpose
This experiment shows the feasibility of turning a robot without generating forward locomotion.

Implementation
The simulation consists of an actuating robot in an aquatic environment with a time step of 0.005 seconds and a total simulation time of 10 seconds. Joint actuation values are calculated and set every fourth time step in order to increase simulation speed. Therefore, actuation values are updated every 0.02 seconds (similar to Section 7.2). The robot is defined to consist of 17 links and 16 joints. The robot's total mass is set to 318.75 ($mass\_units$). A robot's link dimensions are set to $1.5 \times 0.5 \times 0.5$ $units$. Links are placed consecutively after each other in negative X direction. The front face (outer position of the first element) is set to $X = 0$ and the rear face (outer position of the last element) to $X = -25.5$. Therefore, the COM of the robot is situated at $X = -12.75$. Furthermore, the robot's Y position is lifted by 1 $units$ whereas Z remains at 0 $units$. Maximum joint forces are set to 100000 ($force\_units$).

Joint actuation angles are calculated according to Equation 3.4 as described in Section 3.3. A so-called spinning gait is generated by computing the parameter values β and γ with Equations 3.6 and 3.7. This spins the robot on its COM around the Y axis. A visualization of a robot in a vacuum environment is shown in Figure 3.10. Initial parameter values are set to: $\alpha = 0.2$, $f = 1$, $\beta = \frac{2\pi}{16}$ $\gamma = 0.3$.

Results
After a simulation starts, the robot turns around the Y axis (vertical) in counter-clockwise direction as shown in Figure 7.17. It can be seen that two half sine waves travel from the middle of the robot (COM) to the outside links. Each in opposite direction. Both half sine waves force the robot to spin in place by stretching (Figure 7.17(c)) and contracting (Figure 7.17(b)) its links in the appropriate manner, while the robot's COM stays at the same location. Therefore, no forward velocity is generated.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.17: Example of an 17 link robot applying the spinning gait described in Section 7.3. (a) shows the initial position of the robot. (b) depicts the robot with contracted joints and (c) with straightened joints at subsequent time stamps. This causes the robot to spin around the Y axis. A video can be seen at https://youtu.be/Q6puZfnDEhY.

Conclusion
The results show that the robot can be turned without the generation of forward motion. Two sine waves travel in opposite direction through the robot's body, each propelling the robot in the same angular counter clockwise direction. The direction can be modified by changing the sign of the γ parameter. In situ turning can be used for hand-coded robot control in subsequent experiments that require a turning behavior without forward velocity, e.g. for steering the robot's direction to a position in which it can touch a target.

7.7 Watersnake Evolved with DEAP

DEAP, a Python library, provides standardized functionality for evolutionary algorithms [20]. Use of the library can reduce code complexity and help to improve source code quality. As an initial test, a forward swimming robot is evolved with DEAP.

Purpose
This experiment shows the feasibility of using DEAP for optimizing the robot's sinusoidal locomotion parameters. Furthermore, the usability of DEAP compared to that of a hand-coded implementation are discussed quantitatively. Evolved parameters will be used for simple forward swimming tasks in subsequent experiments.

Implementation
The simulation setup for this experiment is similar to that in Section 7.6. This implementation utilizes the DEAP library in order to codify a GA for sinusoidal parameter optimization. Selection, crossover and mutation operators are called from DEAP's *tool* module, which provides a set of standard

functions for selecting, moving and modifying individuals and is designed to be used in conjunction with DEAP's *toolbox* module. The *toolbox* module provides functionality to build the basic structure of evolutionary algorithms (in this case a GA). For instance, it allows the registration of variation operators and genome creators.

The GA is parameterized with a population size of 120 individuals and conducts evolution over 1000 generations. Each individual's genome consists of 3 alleles, where each allele is defined as a robot's joint parameter in the form: $\alpha,\ f,\ \beta$. The amplitude parameter (α) ranges from 0.1 to 1.5, the frequency parameter (f) from 0.1 to 3 (Hz) and phase shift (β) from 0 to $2 * \pi$ (rad). The genome's alleles are initialized according to an uniform distribution within the corresponding parameter range. For each individual, an ODE simulation is conducted with the given parameters. Specifically, all joints actuate according to Equation 3.4. Only the joint index i is responsible for different joint angles for different joints. Fitness is evaluated with:

$$fitness(d) \ = \ d_X \tag{7.4}$$

where d is the distance between starting position and end position in the coordinate system. Therefore, the fitness function only rewards traveling in positive X direction and penalizes movement in negative X direction. Movement in Z and Y is omitted with this fitness function. From the population, 120 parents are chosen with tournament selection (without replacement) and a tournament size of $k = 3$. Afterwards, a simulated binary crossover (see Section 5.1) is performed with 90% crossover probability. The *simulated binary crossover* implementation takes an upper bound, a lower bound, and a crowding degree argument. A "high" crowding degree produces children resembling to their parents, while a "small" crowding degree produces children more different from their parents. Upper and lower bounds are set to the according limits of the actuation parameter ($\alpha,\ f,\ \omega$) ranges and the crowding degree to a value of 20. This procedure returns an intermediate individual which is then processed in the mutation operation. The mutation probability of an intermediate individual is set to $33.\overline{3}\%$. Mutation is performed as *polynomial mutation* with upper and lower bounds set to the actuation parameter limits and a crowding degree of 20.

Results

The fitness progress in Figure 7.18 shows that the mean of the best individuals of 18 replicates converges to a fitness of 97.20 *units*, whereas the mean of the average individuals converges to 80.92 *units*. Both 95% CIs are close to their corresponding mean values. The CI of the average individual data set is small over all generations. This effect is even more prominent in the best individuals data set, where the CI is barely visible in the plot after generation 10. This result indicates a low variance for each data set over

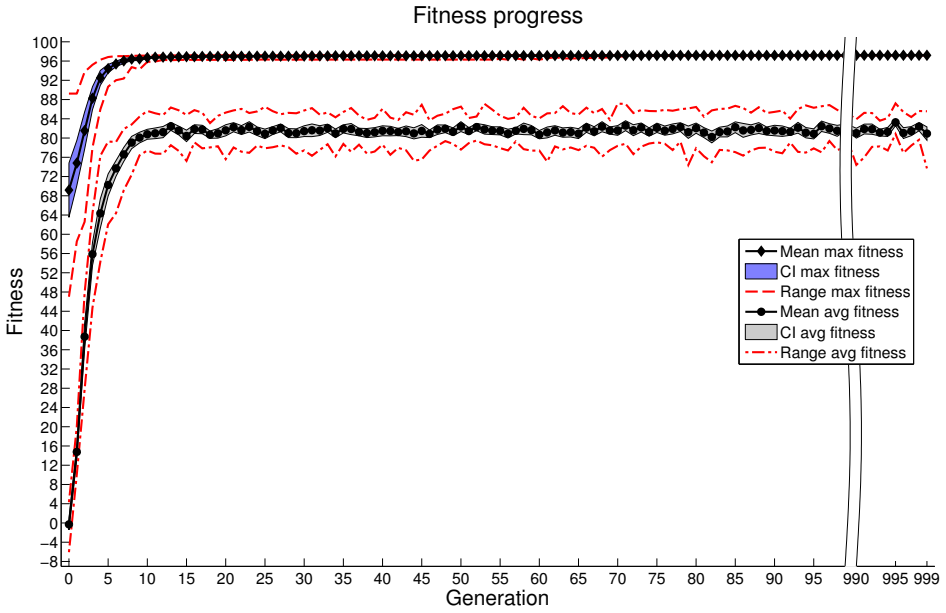


Figure 7.18: Evolutionary fitness progression for the forward locomotion experiment implemented with DEAP that consists of a 16 joint robot. 3 joint parameters are evolved as described in Section 7.7. The mean of the best individual per generation (*Mean max fitness*) and the mean of the average individual per generation (*Mean avg fitness*) over 18 replicates are significantly different ($p < 0.05$ with WRS). Shaded areas represent the 95% confidence interval. Red dashed lines describe the best and the worst performing individual per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.

all replicates. Evolutionary runs exhibit the same fitness progress independent from its random number seed configuration. Maximum values indicate the best fitness over replicates and minimum values represent the worst fitness respectively. The best performing individual is the maximum value in the data set of the best performing individuals. In this case, the maximum values quickly converge to 97.20 *units*. They overlap with the mean after generation 10. In other words, each replicate generates a very well performing individual. The individuals with lowest fitness from the best individuals data set are greater than the highest values from the average individuals data set at every generation. Therefore, best individuals perform significantly better than the average individuals. For the sake of completeness, a statistical significance test results in $p < 0.05$ with WRS.

Mean, CI and minimum-maximum range performance values show a fast increase of fitness within the first 10 generations. The rise of all performance values shows an effective parameter search process in the solution space.

A relatively low visible variance of the mean in the best individuals data set indicates that the GA cannot find better performing individuals in the solution space. The average individuals data set shows an expected variance over all generations. This means that search in the solution space is ongoing until the end of the evolutionary run.

Figure 7.19 shows the progress of the parameters α, f and β of the best individuals. Amplitude (α) converges to 0.46, frequency (f) to 2.00 and phase shift to 0.68. The minimum-maximum range of all three parameters narrow to its average value until they are almost invisible during the first 100 generations. This effect is even more prominent for the CI. Therefore, well performing parameters are found very fast in early generations. No significant variance is observable after the first 100 generations.

Figure 7.20 shows the movement of an evolved robot at different time steps. The actuation of joints with evolved parameters from the best individual of generation 999, shows a movement in positive X direction.

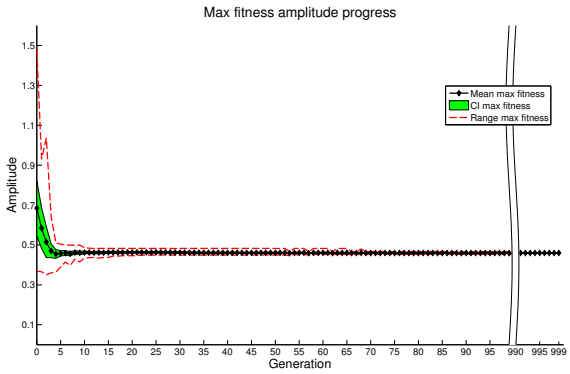
Conclusion
A direct comparison to the forward swimming experiment from Section 7.1 is not applicable due to the differences in the experiment setup, although both experiments evolve forward moving robots. Despite this fact, one can see that fitness values increase much faster in this experiment.

The usage of DEAP reduces development time and effort by providing ready to use functionality for evolutionary algorithms. It can be configured to utilize a ODE simulation for the evaluation of an individual. Due to provided standardized functionality it improves the quality of the evolutionary implementation. For these reasons, subsequent experiments will use DEAP for the implementation of evolutionary runs.

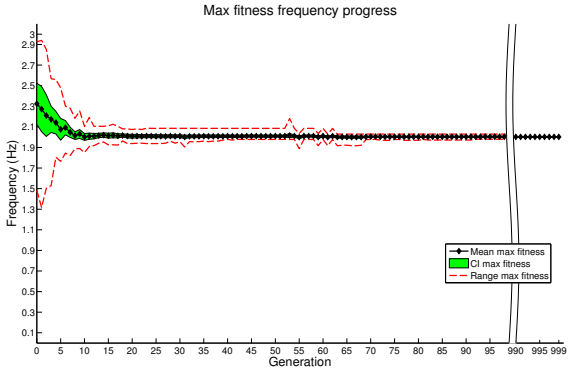
7.8 Evolved Watersnake Head

Previous experiments investigated a robotic behavior where all joints have been used for actuation, namely forward moving, stopping and turning. In contrast, this experiment uses a certain number of joints to enclose a planar area with a geometric form which is assumed to be able to hold an object in subsequent experiments. Specifically, the robot folds its leading joints at the beginning of the simulation. A GA is utilized to evolve actuation parameters after the robot has finished with folding its so called “head.”

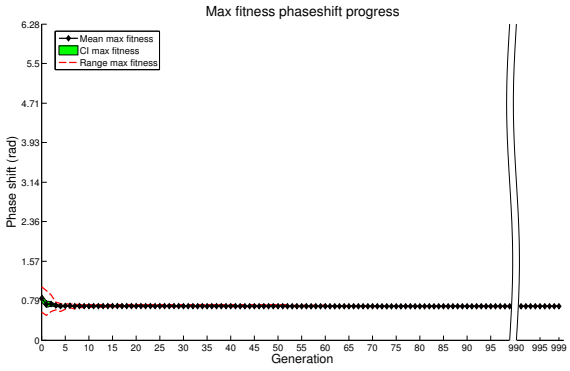
Purpose
This experiment shows how the form of a robot’s head impacts its forward moving behavior. Therefore, its fitness progress is compared between two different “head” forms. The comparison implies analysis of the evolved parameter values. Additionally, this experiment shows, whether the robot can



(a)

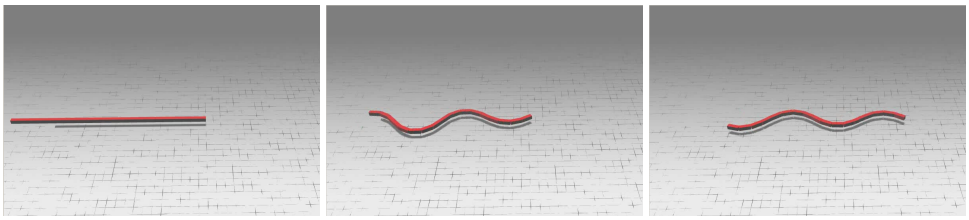


(b)



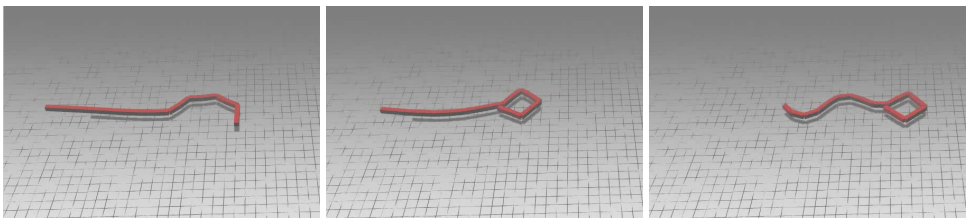
(c)

Figure 7.19: Progress of amplitude (α) (a) frequency (f) (b) and phase shift (β) (c) parameters in the best individuals data set from the evolutionary run described in Section 7.7. Shaded areas represent the 95% confidence interval. Red dashed lines describe the minimum and maximum values per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.20: Example of an evolved 17 link forward swimming robot with the GA implementation described in Section 7.7. (a) shows the initial position of the robot. Then (b)-(c) depict the temporal progress of the robot at different time steps. A sinusoidal wave propagates through the robots body which causes movement to the right. A video can be seen at https://youtu.be/A3CZqODXDCA.



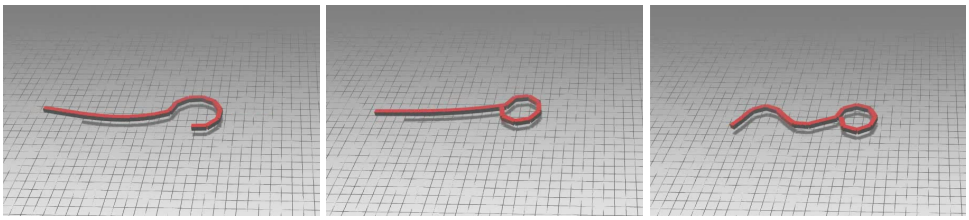
(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.21: Example of an evolved 17 link forward swimming robot and a square head with the GA implementation described in Section 7.8. (a) shows the robot while folding its head. (b) depicts the initial position after before the robot begins to actuate. (c) illustrates the actuating robot at an certain time step. A sinusoidal wave propagates through the robots tail which causes movement to the right. A video can be seen at https://youtu.be/EOFdDCacgso.

keep the form of the head during the whole simulation.

Implementation

This experiment uses the same initial simulation setup as in Section 7.6. At the beginning of the simulation, all robot joints are straightened out to 0°. Within the first three seconds of simulation time, the robot forms a planar "head" by setting the angles of the first eight joints accordingly. This implementation folds two different heads in the form of a square, and a circle as shown in Figure 7.21(b) and Figure 7.22(b) respectively. Each head form is investigated in a separate simulation. First, the square head actuates joints $1, 3, 5$ to $-90°$ while keeping joints $0, 2, 4, 6$ at 0°. Joint 7 is set $+45°$ to orient the head in to positive X direction. Second, the circle head actuates joints 0 to 6 uniformly with $-\frac{360°}{7} = -51.43°$. It also utilizes



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.22: Example of an evolved 17 link forward swimming robot and a circular head with the GA implementation described in Section 7.8. . (a) shows the robot while folding its head. (b) depicts the initial position after before the robot begins to actuate. (c) illustrates the actuating robot at an certain time step. A sinusoidal wave propagates through the robots tail which causes movement to the right. A video can be seen at https://youtu.be/rD7Mek5fkdA.

joint 7 to orient the head in positive X direction by setting its angle to $+45°$. After three seconds, joints 8 to 16 actuate with the same parameters α, f and β according to Equation 3.4, where $\omega = 2 * \pi * f$. This experiment utilizes the GA implementation from Section 7.7 to evolve the parameters α, f and β after the head has been formed.

Results

The mean of the best individuals of 18 replicates converge to a fitness of 24.07 *units*, whereas the mean of the average individuals converges to 21.48 *units* for the square headed robot, see Figure 7.23. According to Figure 7.24, the mean of the best individuals converge to 23.18 *units* and the mean of the average individuals to 20.49 *units* with a circular folded head. CIs of best individuals and average individuals data sets are close to their mean values in either figure. This indicates a low variance for each data set over all replicates. Evolutionary runs exhibit the same fitness progress independent of its random number seed configuration. Maximum values indicate the best fitness over replicates and minimum values represent the worst fitness respectively. No visible difference between the overall best individual and the mean of the best individuals is observable in both figures after generation 10. Hence, each replicate generates a very well performing individual. The overall best individual from the square folded robot attains a fitness of 24.07 *units* and circular folded 23.18 *units*. Individuals with lowest fitness from the best individuals data set are always greater than the highest values from the average individuals data set in for both simulation configurations. Best individuals are therefore performing significantly better than the average individuals (for the sake of completeness: $p < 0.05$ with WRS).

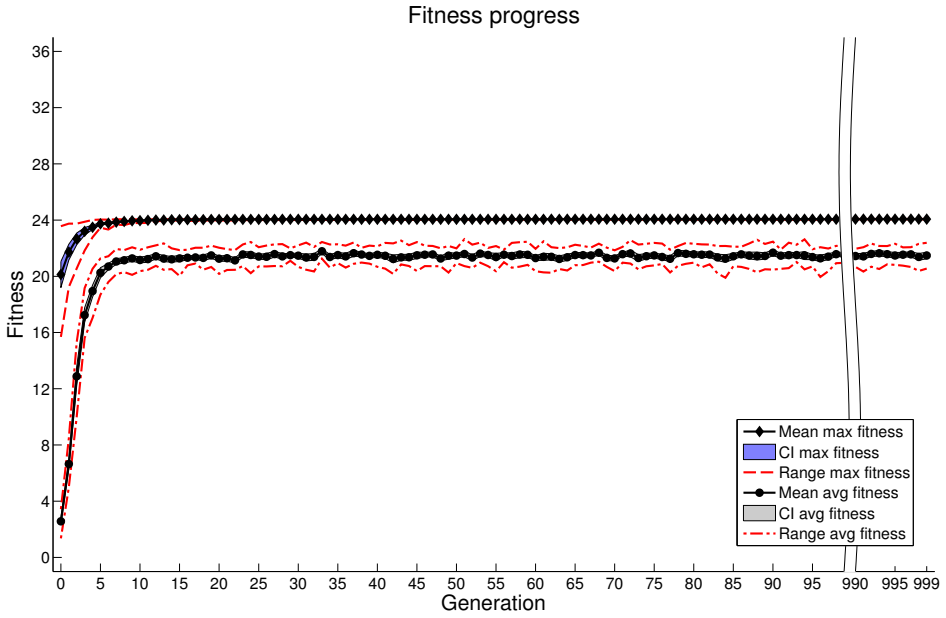


Figure 7.23: Evolutionary fitness progression for forward locomotion implemented with DEAP that consists of a 16 joint robot using its first nine joints to fold a square head. 3 joint parameters are evolved as described in Section 7.7. The mean of the best individual per generation (*Mean max fitness*) and the mean of the average individual per generation (*Mean avg fitness*) over 18 replicates are significantly different ($p < 0.05$ with WRS). Shaded areas represent the 95% confidence interval. Red dashed lines describe the best and the worst performing individual per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.

Similar to the results of forward swimming robot in Section 7.7, mean, CI and minimum-maximum range performance values show a fast increase of fitness within the first 10 generations. This indicates an effective GA search process in the solution space at early generations. A relatively low visible variance in the best individuals data sets indicates that the GA is not able to find better performing individuals in the solution space after a early convergence. The prominent minimum-maximum range for the average performing data sets suggests an active search in the solution space until the end of the evolutionary run.

Figure 7.25 shows the progress of the parameters α, f and β of the best individuals for a square folded head and Figure 7.26 for a circular folded head. Amplitude (α) converges to 0.73, frequency (f) to 1.72 (Hz) and phase shift (β) to 0.92 (rad) for the square headed robot. Evolved parameter values for the circular headed robot are very similar ($\alpha = 0.75$, $f = 1.79$ (Hz),

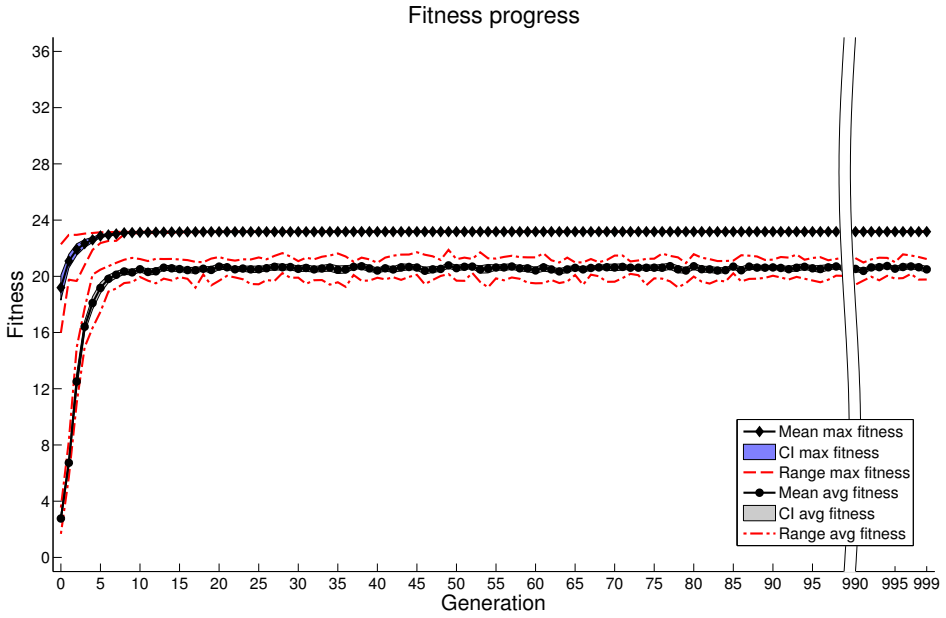


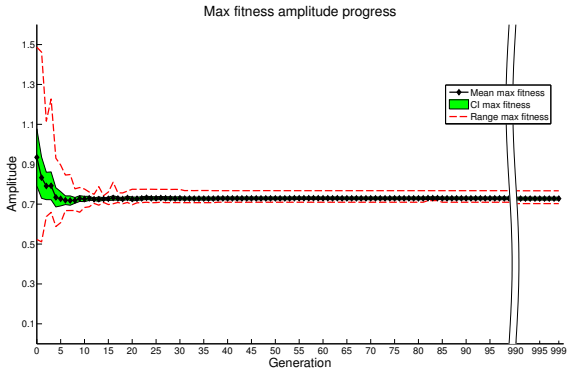
Figure 7.24: Evolutionary fitness progression for forward locomotion implemented with DEAP that consists of a 16 joint robot using its first nine joints to fold a circle head. 3 joint parameters are evolved as described in Section 7.7. The mean of the best individual per generation (*Mean max fitness*) and the mean of the average individual per generation (*Mean avg fitness*) over 18 replicates are significantly different ($p < 0.05$ with WRS). Shaded areas represent the 95% confidence interval. Red dashed lines describe the best and the worst performing individual per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.

$\beta = 0.93$ (rad)). Both simulation configurations exhibit progress at the beginning, but after generation 10, no visible difference between the mean and the CI is observable for all three parameters. Minimum-maximum ranges for parameters f (b) and β (c) are completely hidden after 10 generations. In contrast, parameter α (a) shows a steady visible difference through the last generation.

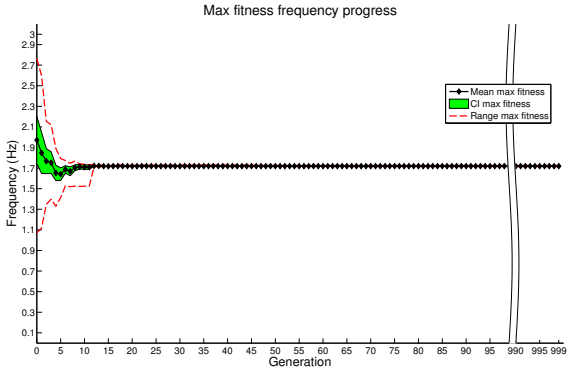
Figure 7.21 depicts the movement of a robot with a square head and Figure 7.22 a robot with a circular head respectively. Both simulation configurations exhibit a prominent sinusoidal actuation for all non folded joints. The folded head keeps its form during the whole simulation.

Conclusion

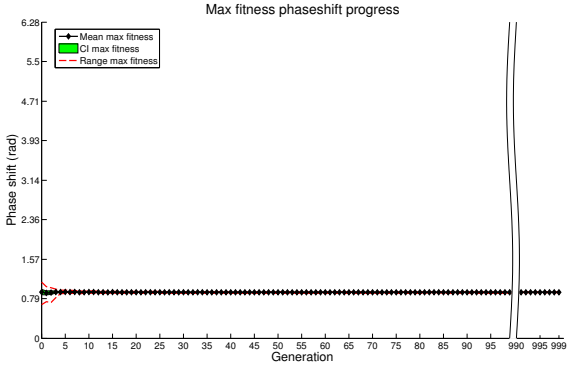
Fitness values of both simulation configurations are very similar. They cannot be directly compared to the forward swimming robot experiment from



(a)

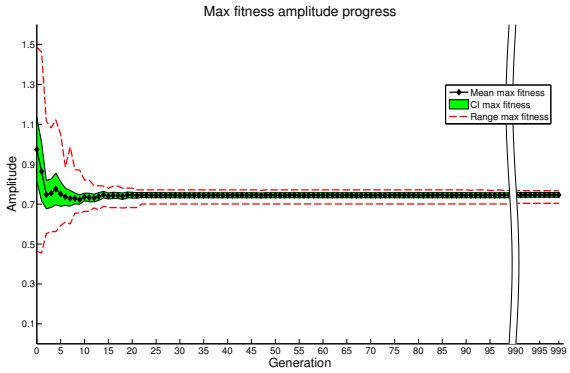


(b)

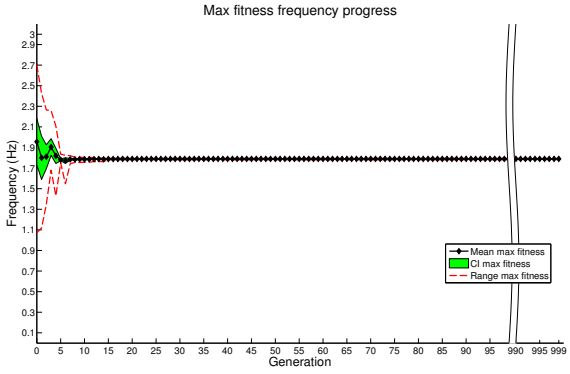


(c)

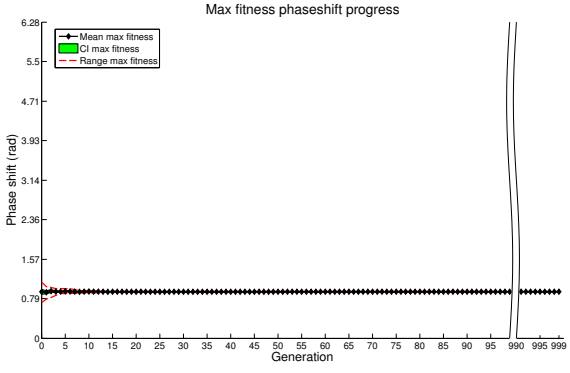
Figure 7.25: Progress of amplitude (α) (a) frequency (f) (b) and phase shift (β) (c) parameters in the best individuals data set from the evolutionary run described in Section 7.8 for a robot with a square head. Shaded areas represent the 95% confidence interval. Red dashed lines describe the minimum and maximum values per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.



(a)



(b)



(c)

Figure 7.26: Progress of amplitude (α) (a) frequency (f) (b) and phase shift (β) (c) parameters in the best individuals data set from the evolutionary run described in Section 7.8 for a robot with a circle head. Shaded areas represent the 95% confidence interval. Red dashed lines describe the minimum and maximum values per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.

Section 7.7, due to different simulation setups. Additionally, evolved parameters show similar values for both types of snake heads. Therefore, it can be concluded that the form of the robot's head does not significantly change the forward moving behavior.

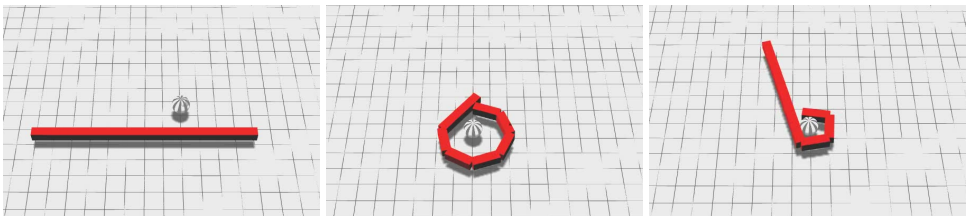
7.9 Grasping

The previous experiment demonstrates forward locomotion with a robot that uses parts of its body to surround an object. This next experiment investigates the formation of the surrounding shape in order to "capture" an actual object. Specifically, the proposed algorithm from Section 3.4 is utilized to grasp a sphere.

Purpose
This experiment evaluates the effectiveness of the proposed grasping algorithm from Section 3.4.

Implementation
A simulation consists of an articulated robot and a stationary target in an aquatic environment with a time step of 0.005 seconds and a total simulation time of 10 seconds. Joint actuation values are calculated and set every fourth time step in order to increase simulation speed. Therefore, actuation values are updated every 0.02 seconds. The robot is defined to consist of 8 links and 7 joints, and the target object is a sphere with radius 1 $length\_unit$ at the initial position $X = -4$, $Z = -2$. The robot's total mass is set to 150 ($mass\_units$) and the target object's to 0.42 ($mass\_units$). The robot's link dimensions are set to $1.5 \times 0.5 \times 0.5$ $units$. Links are placed consecutively in negative X direction. The front face (outer position of the first element) is set to $X = 0$ and the rear face (outer position of the last element) to $X = -25.5$. Therefore, the COM of the robot is situated at $X = -12.75$. Furthermore, the robot's Y position is lifted by 1 $units$ whereas Z remains at 0 $units$. Maximum joint forces are set to 10000 ($force\_units$).

The robot keeps its joints straightened out with 0° during the first second of the simulation. Afterwards, the robot employs Algorithm 3.1. The robot forms a circle around the object (see Figure 3.13(a) and (b)) and tightens it every second. In this experiment, the circle is tightened four times. Hence, four links surround the target object. It is important to note that this implementation works only if the object is smaller than the resulting area formed by the folded part of the robot. The robot does not actuate its joints after the grasping algorithm has been executed. Therefore, it keeps its position for the remaining time of the simulation.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.27: Example of an eight link robot that employs the grasping algorithm from Section 3.4 in order to grasp a spherical target object. (a) depicts the initial position of the robot and the target object. (b) shows the robot while enclosing the object with a full circle. (c) illustrates the tightened circle with the target object in it. A video can be seen at https://youtu.be/Jy3CFm1ecOc.

Results
Figure 7.27 shows that the robot successfully surrounds and grasps the target object. It starts to form a circle around the target object and tightens the circle once every second, until the target is completely surrounded by four links. One can see that the robot changed its orientation (compare Figure 7.27(a) and (c)).

Conclusion
Results show the effectiveness of the proposed engineered grasping algorithm. In this experiment, the target object has been placed at a location which allows the robot to grasp it from its initial position. Therefore, subsequent experiments can utilize this algorithm for grasping tasks, once the robot has reached a location near the target.

7.10 Evolved Watersnake with Grasped Object

This experiment combines locomotion and grasping behaviors from previous sections. A target object is placed at a specific location, so that the robot is able to grasp it from its initial position. Sinusoidal locomotion parameters will be evolved after a target object has been grasped.

Purpose
This experiment shall show if the implemented GA can successfully find parameters for forward swimming with a grasped object.

Implementation
This experiment uses the same initial simulation setup as in Section 7.6.

Additionally, a sphere is placed into the environment at the position $X = -12$, $Z = -3$ as target object. Its mass is set to 0.42 ($mass\_units$).

The robot surrounds the target object at the beginning of the simulation by forming a circle around its COM. Afterwards, it iteratively tightens the circle 13 times until the target object touches four links. Each iteration step is executed with a delay of 0.12 seconds in the simulation. Therefore, the grasping process is finished after 1.68 seconds. After 3 seconds of simulation time, the robot starts to actuate its remaining joints according to Equation 3.4 with α, f and β, where $\omega = 2 * \pi * f$. In this case, joints $0, 1, 2$ are set to $+90°$ (output of the grasping algorithm Algorithm 3.1) and joint 3 is used to turn the head into the robot's principal direction of motion by setting it $-45°$. Parameters α, f and β are evolved with the GA implementation from Section 7.7.

Results

Figure 7.28 shows that the mean of the best individuals of 18 replicates converges to a fitness of 24.92 $units$, whereas the mean of the average individuals converges to 13.51 $units$. The 95% CI of the best individuals data set narrows to its mean value during the first 20 generations until it is not visible in the figure. Additionally, the minimum-maximum range exhibits a similar tendency, but it can be visually distinguished from the mean value until the last generation of the evolutionary run. Therefore, the best individuals over all replicates exhibit the same fitness progress independent of the random number seed configuration. CI and minimum-maximum range from the average individuals data set show an inverted behavior. They are very close to the mean at the beginning and expand until generation 5. Afterwards they keep the same wide range until the end of the evolutionary run. This means, that the GA's search in the solution space is ongoing until the end. The worst performing individuals from the average individuals data set exhibit a visually distinct progress. They are considerably lower than the mean value and show a notable variance. This hypothesis may indicate that the solution space is difficult to search. For example, it might contain many local minima to be explored by the GA. In other words, the adjustment of the three parameters α, f and β is fragile, in that a small modification of one parameter may reduce the fitness tremendously. This is supported by the relatively wide CIs and minimum-maximum ranges in Figure 7.29 (best individuals data set) compared to previous experiments. Although, the mean values of amplitude (α), frequency (f) and phase shift (β) converge to 0.62, 1.78 and 0.73 one can see a prominent minimum-maximum range. These results indicate that the GA cannot narrow down the parameter values and repeatedly tries values in this range. The analysis of the dependency between α-f-β in generation 999 from the best individuals data set (depicted in Figure 7.30) shows one outlier, namely replicate 5. Although the other

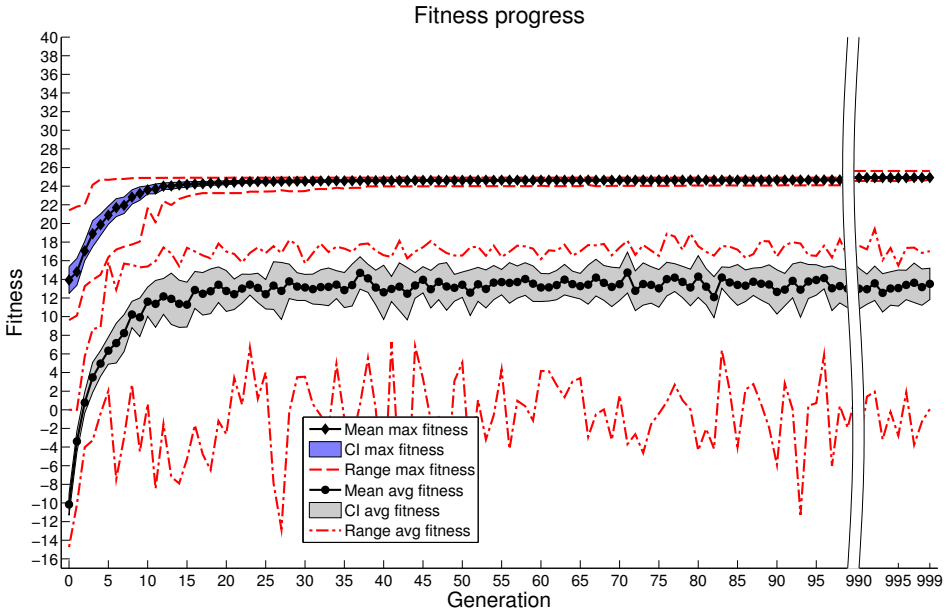
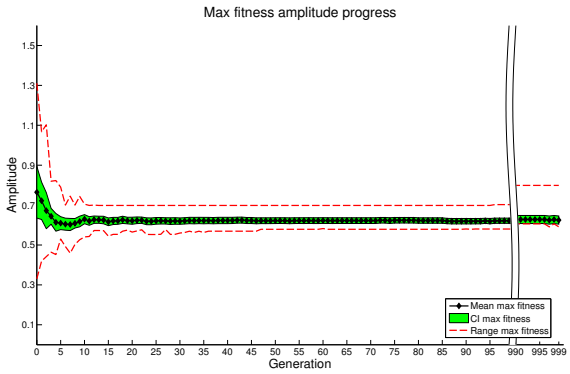


Figure 7.28: Evolutionary fitness progression for the forward locomotion experiment with a grasped target object that consists of a 16 joint robot. The parameters α, f and β are evolved as described in Section 7.10. The mean of the best individual per generation (*Mean max fitness*) and the mean of the average individual per generation (*Mean avg fitness*) over 18 replicates are significantly different ($p < 0.05$ with WRS). Shaded areas represent the 95% confidence interval. Red dashed lines describe the best and the worst performing individual per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.

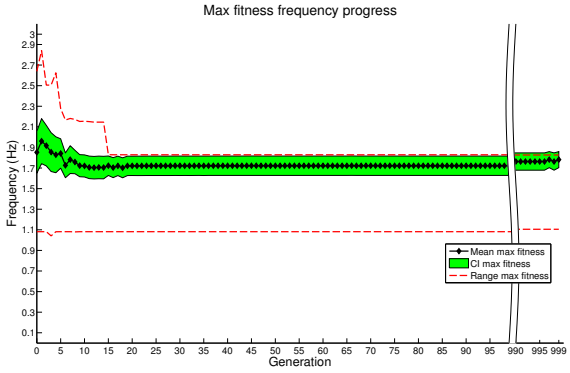
replicates converged to relatively similar values ($\alpha \approx 0.61$, $f \approx 1.82$ (Hz) and $\beta \approx 0.75$ (rad)), replicate 5 evolved to $\alpha = 0.80$, $f = 1.11$ (Hz) and $\beta = 0.35$ (rad). Both parameter settings result in approximately the same fitness value. Otherwise, the minimum-maximum range of the best individuals data set at generation 999 in Figure 7.28 would show a discrepancy. See figure 7.31 for a robot that is grasping and swimming.

Conclusion

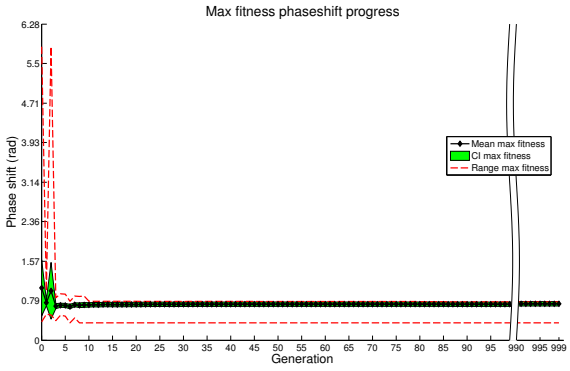
Results show that the GA can successfully evolve parameters for a forward swimming robot with a grasped object. Evolution finds quickly well performing parameters, although the solution space seems to be more complex compared to previous experiments.



(a)



(b)



(c)

Figure 7.29: Progress of amplitude (α) (a) frequency (f) (b) and phase shift (β) (c) parameters in the best individuals data set from the evolutionary run described in Section 7.10. Shaded areas represent the 95% confidence interval. Red dashed lines describe the minimum and maximum values per generation over all 18 replicates. The area between generations 100 and 990 has been omitted, because it contains no relevant information.

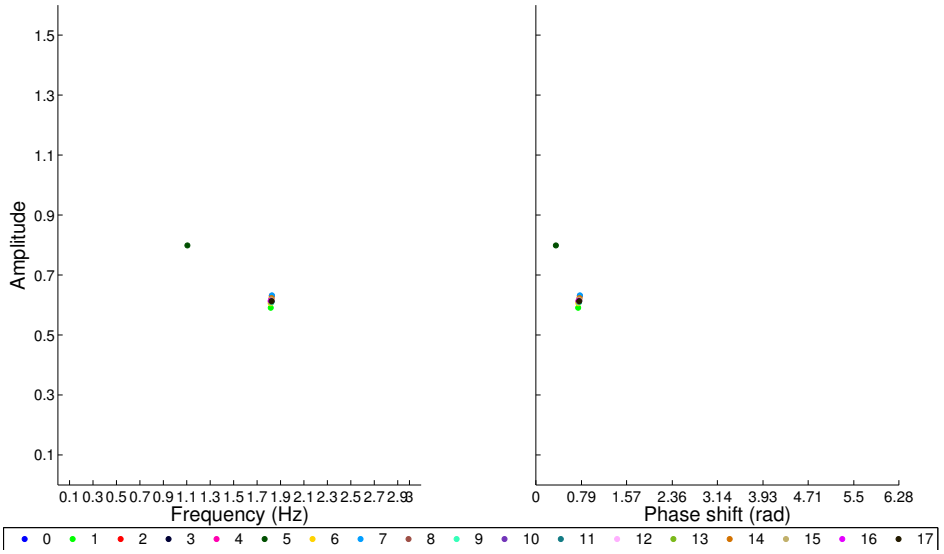


Figure 7.30: Parameter dependency of the best individuals for each of the 18 replicates in the last generation from the evolutionary run described in Section 7.10. The dependency amplitude (α)-frequency (f) is depicted on the left and amplitude (α)-phase shift (β) on the right.

(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

Figure 7.31: Example of an evolved 17 link robot that grasps a target object and swims forward with the object. Locomotion parameters are evolved with an GA. The implementation is described in Section 7.10. (a) shows the robot while forming a circle around the object. (b) shows the robot while tightening the circle. (c) illustrates the robot actuation its joints sinusoidally with a fully tightened circle and the target object in it. A video can be seen at https://youtu.be/kLRpDrxdSA4.

7.11 Capture Target

The previous target approach experiment (described in Section 7.3) does not include a mechanism to grasp a target object. In this next experiment, the robot approaches and grasps or *captures*, a target.

Purpose
This experiment shows the feasibility of capturing a target object in all four quadrants of the Cartesian coordinate system. Additionally, a continuous steering algorithm shall be implemented in order to replace the state machine from Section 7.3.

Implementation
The experiment uses the same simulation setup as the experiment described in Section 7.10. In order to evaluate the functionality in every quadrant, the target object (a sphere) is placed at following four locations (X, Z): (1) $(30, 30)$, (2) $(-30, 30)$, (3) $(30, -30)$, and (4) $(-30, -30)$. Each location is tested with a separate simulation run.

Locomotion is performed with $\alpha = 0.7$, $f = 0.9$, $\beta = 0.8$ according to Equation 3.4, where $\omega = 2{*}\pi{*}f$. The steering algorithm from the experiment described in Section 7.3 has been used to approach the target. In contrast, this implementation utilizes a continuous direction adjustment instead of a state machine. Therefore, γ is used to steer the robot into a desired direction. It is calculated with:

$$\gamma(angle) \;=\; \frac{-0.2}{\pi} \;*\; angle, \tag{7.5}$$

where *angle* is calculated with Equation 7.3. In order to smooth the angle between the robot's direction motion and the shortest path to the target, an average filter with window size 30 is utilized. The distance between the robot and the target object (called *comdist*) is calculated as the Euclidean distance between the robot's COM and the target position. If the robot reaches a region near to the target object ($comdist \;<\; arclength$, where $arclength = 25.5\ units$) one following two paths is executed:

- $comdist < \frac{arclength}{\pi}$: The robot can reach the target if it forms a circle ($\frac{circumference}{\pi}$). Therefore, the grasping algorithm as experimentally tested in Section 7.10 is executed. The direction of the grasping (to the left, or to the right) has been determined in the other execution path.
- else: The robot cannot reach the target if it would form a circle. Hence, it steers further to the target object. Additionally, the direction of the grasping is determined in this execution path. If the (unfiltered) θ value is greater than or equal to 0 then the robot grasps to its left, otherwise it grasps to its right.

The robot keeps its joint angles constant after the grasping algorithm has been performed until the end of the simulation. Therefore, the robot does not actively move with the grasped target. That last subtask will be addressed in Section 7.12.

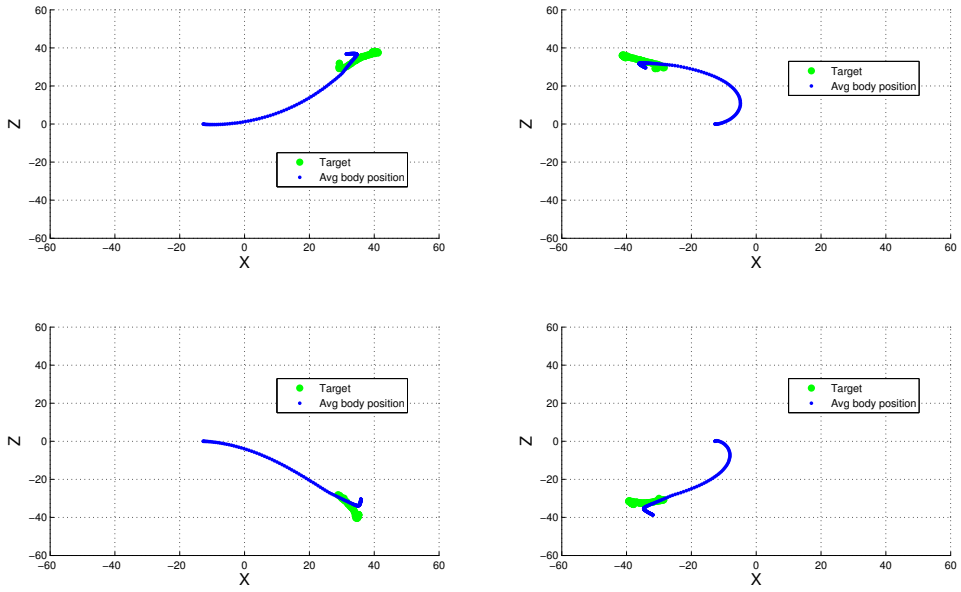


Figure 7.32: Robot approaching and capturing a target object at four different locations. Top left shows a target positioned in the first, top right a target in the second, bottom left a target in the fourth and bottom right a target in the third quadrant.

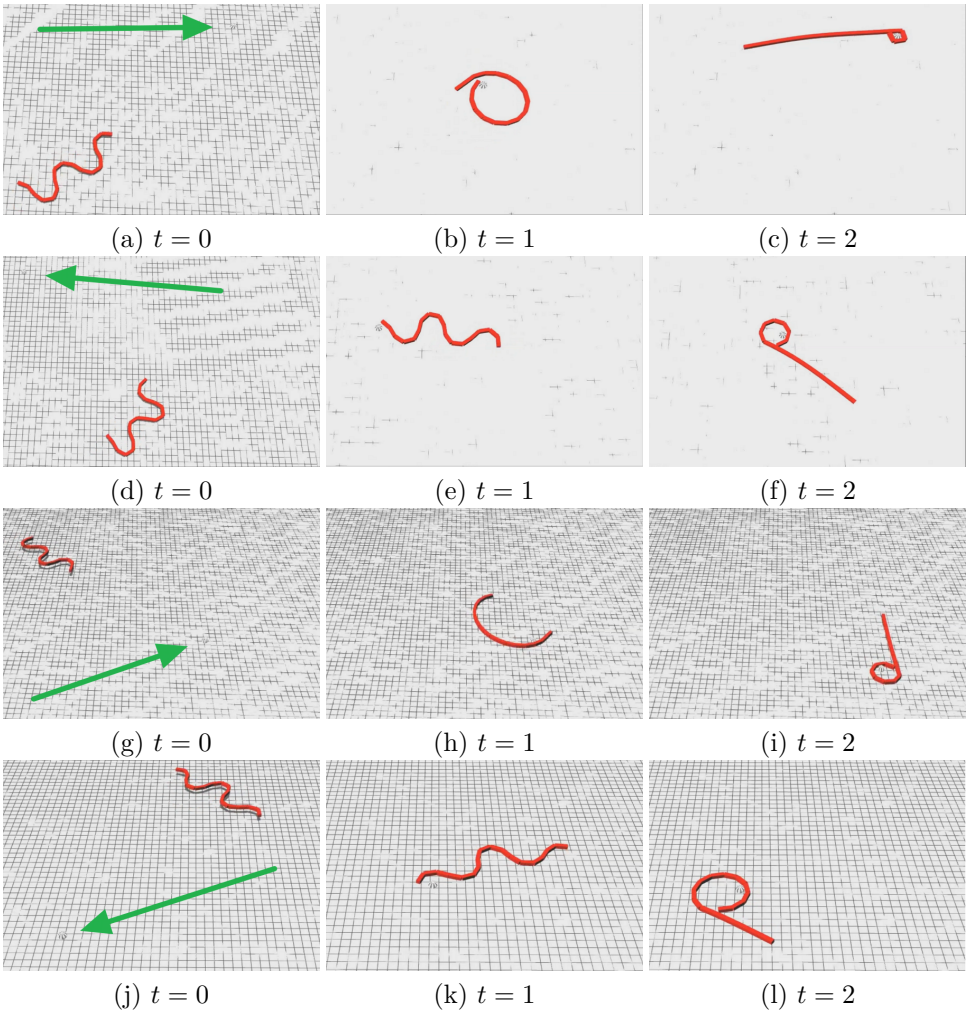
Results

Four different simulations were conducted in order to validate whether the algorithm is capable to capture a target in all four quadrants in the Cartesian coordinate system. The robot's COM and the object's position have been recorded during the simulations. The former is visualized in Figure 7.32. One can see that the robot successfully approaches every target position by examining the robot's movement of the COM (blue line). The robot has to turn its body in the top right and the bottom left figures in order to successfully approach the target. After the robot has reached a certain region, it executes the grasping algorithm which causes the target object (green line) to move slightly.

This behavior can also be seen in the simulation visualization in Figure 7.33. The robot approaches all four targets from the same initial position and successfully grasps them (see (c), (f), (i) and (l)). (d)-(f) and (j)-(l) show the robot's turning behavior at the beginning of the simulation in order to approach the target.

Conclusion

This experiment shows that the proposed continuous steering algorithm successfully enables the robot to approach the target object. Furthermore, it is shown that the target can be successfully grasped following the approach.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

(d) $t = 0$ (e) $t = 1$ (f) $t = 2$

(g) $t = 0$ (h) $t = 1$ (i) $t = 2$

(j) $t = 0$ (k) $t = 1$ (l) $t = 2$

Figure 7.33: Example of an 17 link robot that captures a target at four different target locations: (1) first quadrant at $X = 30$, $Z = 30$ in (a)-(c), (2) second quadrant at $X = -30$, $Z = 30$ in (d)-(f), (3) fourth quadrant at $X = 30$, $Z = -30$ in (g)-(i) and (4) third quadrant at $X = -30$, $Z = -30$ in (j)-(l). Each is depicted temporally starting from a position at the beginning of the simulation until the target is successfully captured. The robot steers to the target location (marked with a green arrow in (a), (d), (g) and (j)) and starts the grasping algorithm. A video can be seen at https://youtu.be/8iONn0tlILk.

Therefore, this implementation can be used for the final experiments.

7.12 Capture and Deliver Target

Previous experiments investigated performance of three individual tasks: (1) forward locomotion in order to reach a target, (2) grasping the target, and (3) forward locomotion with a payload in order to reach a destination. The experiment described in Section 7.11 already combined (1) and (2). In contrast, the experiment described here combines all three individual tasks.

Purpose
This experiment shall combine all individual tasks in order to perform target collection and delivery. In order to improve robustness, if the robot fails to grasp the target on the first attempt, it will try again. As described below, the robot will keep trying until it has captured the object.

Implementation
This experiment uses the same simulation setup as the experiment described in Section 7.11. Furthermore it uses partly the same source code for the joint angle calculation. Due to the fact, that the grasping algorithm failed to grasp the target object at $X = -40$, $Z = 40$ in a preliminary simulation, this implementation uses the four target locations $(X,\ Z)$: $(40, 40)$, $(-40, 40)$, $(40, -40)$, and $(-40, -40)$ in order to capture and consequently handle this case. The ultimate target destination is defined to be circular ($radius = 8\ units$) area around $X = 0$, $Z = 0$. This is the region in which the target object should be situated at the end of the simulation.

As in the experiment described in Section 7.11, locomotion parameters for forward swimming without a payload are set to $\alpha = 0.7$, $f = 0.9$, $\beta = 0.8$ for Equation 3.4, where $\omega = 2 * \pi * f$. This experiment utilizes the implementation for capturing a target object as described in Section 7.11.

After the robot has approached the target and executed the grasping algorithm, a check is performed to verify that the target object has actually been caught by the robot. It might be the case that the target is "kicked" away by the robot during the execution of the grasping algorithm. To this end, the current position of the target is compared with the link positions (links $0-3$, defined by the grasping algorithm) which are involved for holding the object. If one of the target's X/Z coordinates is greater or less than all of the involved link coordinates, then the target object is not captured, because it is positioned outside of the grasping "arm". Otherwise, the object is successfully grasped and therefore captured. In that case the robot uses the same steering procedure as described for approaching the target in order to turn itself into the right direction for returning to the destination region. The robot calculates the angle between its COM and the ultimate destination region at $X = 0$, $Z = 0$ and adjusts the γ variable accordingly. Forward propulsion is generated by actuating joints 3 to 15 sinusoidally with the values $\alpha = 0.61$, $f = 1.82$, $\beta = 0.75$.

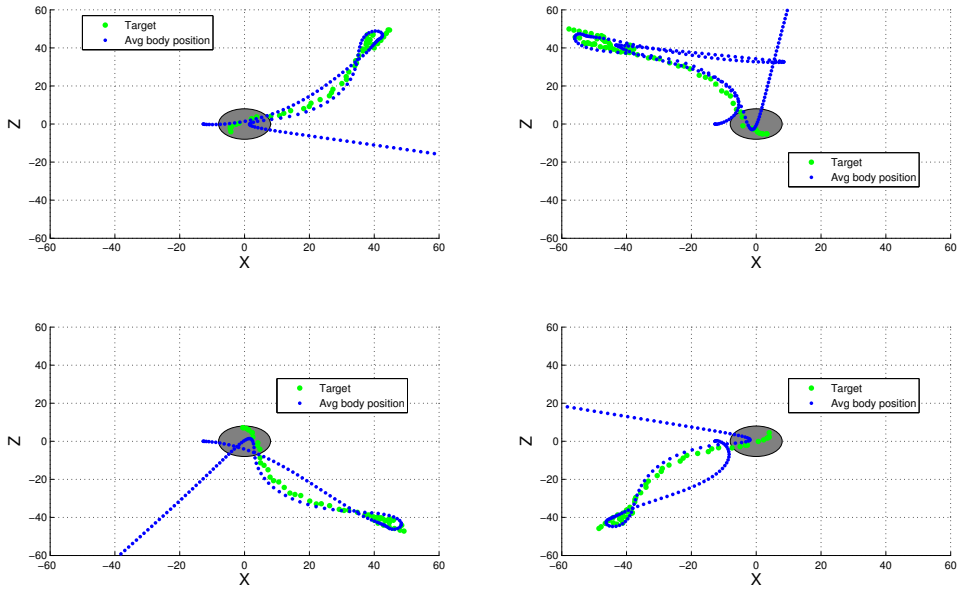


Figure 7.34: Robot capturing and delivering a target object. The target object is placed at four different locations. Top left shows a target positioned in the first, top right a target in the second, bottom left a target in the fourth and bottom right a target in the third quadrant. The destination region is always positioned at $X = 0$, $Z = 0$.

In the case that the object has not been successfully grasped, the robot performs reverse locomotion (by applying a negated β value from the sinusoidal parameters) until it is twice the robot's *arclength* away from the target object (Euclidean distance from the robot's COM to the target object). Then it starts the target capture procedure all over again. Therefore, the robot tries to capture the target until it is actually captured until the end of the simulation, if necessary.

The robot drops the target object if the Euclidean distance between target object destination ($X = 0$, $Z = 0$) is less than 4 *units*. Afterwards, the robot starts to actuate its joints in reverse direction (negated β value) which causes locomotion in the opposite direction. The robot swims in this direction until the simulation ends.

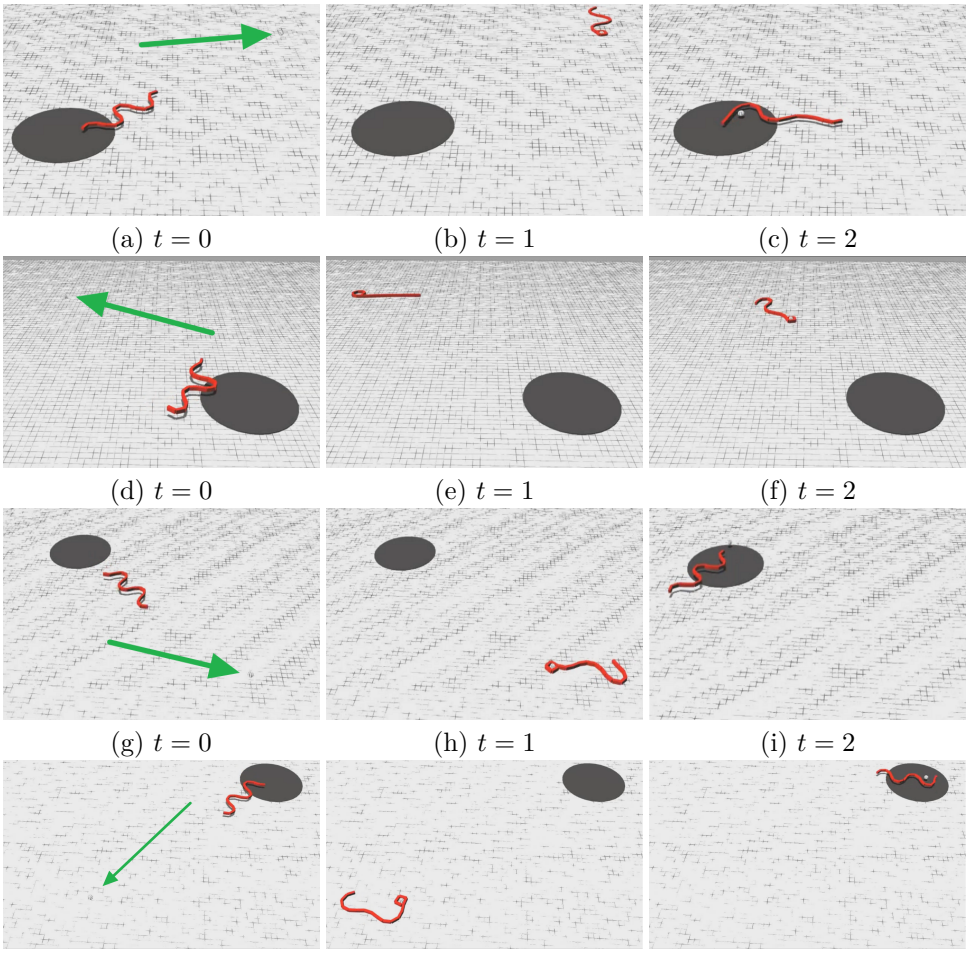
Results

Four different simulations have been started, one for each designated target position in order to validate if the algorithm can gasp and carry a object from each quadrant. The robot's COM and the object's position have been recorded during the simulations. This is visualized in Figure 7.34. The robot (blue) approaches the target (green) in each simulation, grasps it, and starts to approach the destination region (gray), which appears as a blue "sling"

around the target's initial position. The approach to the destination then has a curved form, but all four cases reach the destination region. Afterwards, the robot drops the object and moves away from the destination region. The object stays within the destination region until the end of simulation (green dots remain in the gray area).

A temporal progress for each simulation is visualized in Figure 7.35. Subfigures (a), (d), (g) and (j) show the approaching robot. Subfigures (b), (h) and (k) depict a successful grasping procedure. Subfigure (e) illustrates a failing grasping procedure. The target object is situated outside of grasping links. In this case, the robot swims away from the target object and restarts the target capturing procedure. It is successful after one retry. Consequently, the robot approaches the destination region, where it drops the target and swims away into the opposite direction. Subfigures (c) and (l) depict the moment of dropping the target object. Subfigure (i) shows a time instance in which the robot swims away from the destination region, after it has dropped the target.

Conclusion
Results show that the experiment successfully combines all individual tasks, namely approaching the target, grasping the target, approaching the destination with payload and releasing the target in the destination area. Furthermore, a successful reattempt at capturing the object after an initial failure is demonstrated.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$

(d) $t = 0$ (e) $t = 1$ (f) $t = 2$

(g) $t = 0$ (h) $t = 1$ (i) $t = 2$

(j) $t = 0$ (k) $t = 1$ (l) $t = 2$

Figure 7.35: Example of an 17 link robot that captures a target at four different target locations: (1) first quadrant at $X = 40$, $Z = 40$ in (a)-(c), (2) second quadrant at $X = -40$, $Z = 40$ in (d)-(f), (3) third quadrant at $X = 40$, $Z = -40$ in (g)-(i) and (4) fourth quadrant at $X = -40$, $Z = -40$ in (j)-(l) and delivers it to the destination region (gray area). Each is depicted temporally starting from a position at the beginning of the simulation until the target is successfully delivered to the destination region. The robot steers to the target location (marked with a green arrow in (a), (d), (g) and (j)) and starts the grasping algorithm. Afterwards, it steers to the destination region. The target is released when the destination region has been reached. A video can be seen at https://youtu.be/Sd7RQM8XG_s.

Chapter 8

Conclusion and Future Work

The Experiments in Chapter 7 demonstrate the feasibility of the methodology proposed in this thesis, namely combining human engineering with evolutionary computation in order to address complex problems e.g., capturing and transporting an object in an aquatic environment. Locomotion parameters for the aquatic robot are evolved with a GA. The grasping method and a steering controller are human designed. The literature review in Chapter 3 shows that locomotion of biological creatures can be approximated with sinusoidal wave propagation through a robot's body. Many publications employed this approach to create snake-like robots, but none compose smaller subtasks into a complex task, as demonstrated here.

The robot's design requires sinusoidal actuation of its links/joints in order to generate forward propulsion. Therefore, the number of links used for locomotion must at least be greater or equal than three. Three additional links are necessary to surround a object. Thus, a robot must have at least six links. The experiments in Chapter 7 demonstrate effective forward locomotion (without a grasped object) using eight links. Forward locomotion with a grasped object is implemented with a 17-link robot. These numbers are chosen in order to provide a clearly observable sine wave propagation through the robot's body. A deformation of the robot's shape by its motorized joints is sufficient for the given tasks. More complex shapes could be achieved by bending the robot's body. Risi et al. [53] propose a method that can deform a robot's links in a manner similar to a biological ribosome by using compositional pattern producing networks. This might be useful if the robot should is required to transport much larger objects. Joints might then have three degrees of freedom, enabling the robot to surround a target in three dimensions.

In the experiments described here, GAs are used to optimize sinusoidal locomotion parameters in the conducted experiments. These experiments show the evolution of forward locomotion parameters. While basic swimming locomotion for snake-like robots is well understood, locomotion for an

encumbered robot is not a straightforward extension. Locomotion parameters for a robot grasping an object are difficult to optimize and potentially unintuitive to human engineers.

While GAs excel in optimization tasks, specifying an appropriate representation is difficult in tasks such as grasping and steering. In these cases it is often faster to develop a basic form through an engineered solution. The GAs can be then employed to optimize parameters for the engineered behavior, thereby exploiting the strengths of both engineering and evolution.

Terrestrial undulatory locomotion in robotics is extensively discussed in the literature. However, only a few publications apply the sinusoidal motion pattern to aquatic snake-like robots. This thesis shows the effectiveness of this gait in a computer simulation. The task of capturing and transporting a payload to a destination area can be applied to real world tasks like rescuing an arbitrary shaped object floating in the water. A rescue team on a boat could launch the robot, which would then swim to the object, grasp it, and deliver it to a destination area.

This thesis might also be the starting point for applications of the snake-like robot in field of microrobotics (dimensions less than 1 (mm)) or nanorobotics (dimensions close to 1 (nm)). The scales of nanorobotics and microrobotics would require an aquatic physics simulation that handles physical effects of a low *Reynolds number*[1] as discussed by Purcell [52] for the locomotion of microorganisms. Simulated robots (with external propulsion designed for *in vivo* applications) that perform object transportation in such an environment are investigated in [6], [7] and [8]. These papers employ GAs for the evolving a robot's controller in order to keep the protein level of an human organ high enough. In addition Lenaghan et al. [32] emphasizes the advantage of evolution in biological and therefore nanorobot design. Such a nanorobot could be used in targeted drug delivery for cancer treatment [51], for example. Robots with a snake-like embodiment have already been built for *in vivo* applications in the form of a worm that can stretch and contract its body [9]. If this robot would include motorized joints, then it would be possible to provide active undulatory locomotion. Parameters for the locomotion could be optimized as described in this thesis.

[1]The *Reynolds number* quantifies the relation between momentum and viscosity.

Appendix A

Software Versions

This chapter is used to document software versions that have been used to generate and analyze data on the local machine and the HPC.

A.1 Local Machine

Operating system: *Microsoft Windows 7 Professional 64-bit, 6.1.7601 Service Pack 1 Build 7601*

Visual Studio: *Microsoft Visual Studio Ultimate 2012, Version 11.0.61030.00 Update 4, Microsoft .NET Framework Version 4.5.51209*, with following packages:
Visual Studio 2012 Code Analysis Spell Checker 04940-004-0038003-02121
Microsoft®Visual Studio®2012 Code Analysis Spell Checker
Portions of International CorrectSpell™spelling correction system ©1993 by Lernout & Hauspie Speech Products N.V. All rights reserved.
The American Heritage®Dictionary of the English Language, Third Edition Copyright ©1992 Houghton Mifflin Company.
Electronic version licensed from Lernout & Hauspie Speech Products N.V. All rights reserved.
NuGet Package Manager 2.6.40627.9000
NuGet Package Manager in Visual Studio. For more information about NuGet, visit http://docs.nuget.org/.
Python Tools for Visual Studio 2.1.21008.00
Python Tools for Visual Studio provides IntelliSense, projects, templates, Interactive windows, and other support for Python developers.
Python Tools for Visual Studio - Profiling Support2.1.21008.00
Profiling support for Python projects.

MATLAB: *7.10.0.499 (R2010a)*
Python: *Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit*

(Intel)] with following modules:
angles==1.1
cffi==0.9.0
Cython==0.22
deap==1.0.1
euclid==0.1
matplotlib==1.4.2
MultiNEAT==0.1
numpy==1.7.1
ode==0.12
parse==1.6.6
progressbar==2.3
py2exe==0.6.9
pycparser==2.10
PyEmail==0.0.1
pygame==1.9.1
pymunk==4.0.0
pyparsing==2.0.3
python-dateutil==2.4.0
scipy==0.15.1
six==1.9.0
solidpython==0.1.1

Windows 7, *Visual Studio 2012* and *MATLAB* have been used with licenses from University of Applied Sciences Upper Austria.

A.2 High Performance Computer

Operating system: *#1 SMP Debian 3.2.65-1+deb7u1 x86_64 GNU/Linux*

Python: *Python 2.7.3 (default, Mar 13 2014, 11:03:55) [GCC 4.7.2]* with following modules:
apt-xapian-index==0.45
BeautifulSoup==3.2.1
Brlapi==0.5.7
chardet==2.0.1
cups==1.0
Cython==0.22
deap==1.0.2
defer==1.0.6
feedparser==5.1.2
fpconst==0.7.2
gnome-app-install==0.4.7-nmu1

GnuPGInterface==0.3.2
httplib2==0.7.4
lazr.restfulclient==0.12.0
lazr.uri==1.0.3
louis==2.4.1
Mako==0.7.0
MarkupSafe==0.15
MultiNEAT==0.1
numpy==1.6.2
oauth==1.0.1
Open-Dynamics-Engine==0.1
pexpect==2.4
PIL==1.1.7
progressbar==2.3
pycurl==7.19.0
Pyste==0.9.10
python-apt==0.8.8.2
python-debian==0.1.21
python-debianbts==1.11
pyxdg==0.19
reportbug==6.4.4
reportlab==2.5
simplejson==2.5.2
SOAPpy==0.12.0
unattended-upgrades==0.1
uTidylib==0.2
wadllib==1.3.0
zope.interface==3.6.1

References

Literature

[1] Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. 1st. Chapman & Hall/CRC, 2009 (cit. on p. 30).

[2] Yuhi Awa and Kobayashi Kobayashi. "Kinematics Simulation by Using ODE and MATLAB". In: *Proceedings of the Internation Conference on Instrumentation, Control and Information Technology (SICE)*. Sept. 2007, pp. 3090–3094 (cit. on p. 33).

[3] Antonio Bicchi and Vijay Kumar. "Robotic Grasping and Contact: A review". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. Apr. 2000, 348–353 vol.1 (cit. on p. 21).

[4] Josh Bongard. "The 'What', 'How' and the 'Why' of Evolutionary Robotics". In: *New Horizons in Evolutionary Robotics*. Vol. 341. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2011, pp. 29–35 (cit. on p. 2).

[5] Rodney Brooks. "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network". In: *Neural computation* 1.2 (1989), pp. 253–262 (cit. on p. 2).

[6] Adriano Cavalcanti. "Assembly Automation with Evolutionary Nanorobots and Sensor-Based Control applied to Nanomedicine". In: *Proceedings of the 2nd IEEE Conference on Nanotechnology (NANO)*. Aug. 2002, pp. 161–164 (cit. on p. 88).

[7] Adriano Cavalcanti and Robert Freitas. "Nanorobotics Control Design: A Collective Behavior Approach for Medicine". In: *IEEE Transactions on NanoBioscience* 4.2 (June 2005), pp. 133–140 (cit. on p. 88).

[8] Adriano Cavalcanti, Tad Hogg, and Bijan Shirinzadeh. "Nanorobotics System Simulation in 3D Workspaces with Low Reynolds Number". In: *Proceedings of the International Symposium on Micro-*

NanoMechatronics and Human Science (MHS). Nov. 2006, pp. 1–6 (cit. on p. 88).

[9] Francesco Cepolina and Rinaldo Michelini. "Robots in Medicine: A Survey of In-body Nursing Aids". In: *Proceedings of the 35th International Symposium on Robotics International Symposium on Robotics*. 2004 (cit. on p. 88).

[10] Lara Cowan and Ian Walker. ""Soft" Continuum Robots: The Interaction of Continuous and Discrete Elements". In: *Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems (ALIFE 11)*. MIT Press, Cambridge, MA, 2008, pp. 126–133 (cit. on p. 22).

[11] Alessandro Crespi, Konstantinos Karakasiliotis, Andre Guignard, and Auke Jan Ijspeert. "Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits". In: *Transactions on Robotics* 29.2 (2013), pp. 308–320 (cit. on pp. 8, 12, 18).

[12] Geoff Cumming and Sue Finch. "Inference by Eye: Confidence Intervals and How to Read Pictures of Data". In: *American Psychologist* 60.2 (2005), p. 170 (cit. on p. 40).

[13] Kalyanmoy Deb and Ram Bhushan Agrawal. "Simulated Binary Crossover for Continuous Search Space". In: *Complex systems* 9.2 (1995), pp. 115–148 (cit. on p. 30).

[14] Kalyanmoy Deb and Mayank Goyal. "A Combined Genetic Adaptive Search (GeneAS) for Engineering Design". In: *Computer Science and Informatics* 26 (1996), pp. 30–45 (cit. on p. 31).

[15] Ralf Der and Georg Martius. "Model Learning". In: *The Playful Machine*. Vol. 15. Cognitive Systems Monographs. Springer Berlin Heidelberg, 2012, pp. 183–200 (cit. on p. 34).

[16] Ralf Der and Georg Martius. "The LpzRobots Simulator". In: *The Playful Machine*. Vol. 15. Cognitive Systems Monographs. Springer Berlin Heidelberg, 2012, pp. 293–308 (cit. on p. 33).

[17] Jack Dongarra, Iian Duff, Danny Sorensen, and Hank van der Vorst. *Numerical Linear Algebra on High-Performance Computers*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 1998 (cit. on p. 6).

[18] Evan Drumwright, John Hsu, Nathan Koenig, and Dylan Shell. "Extending Open Dynamics Engine for Robotics Simulation". In: *Simulation, Modeling, and Programming for Autonomous Robots*. Ed. by Noriaki Ando, Stephen Balakirsky, Thomas Hemker, Monica Reggiani, and Oskar von Stryk. Vol. 6472. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 38–50 (cit. on p. 33).

[19] Agoston Eiben and James Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003 (cit. on p. 29).

[20] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13 (July 2012), pp. 2171–2175 (cit. on p. 63).

[21] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989 (cit. on p. 30).

[22] James Gray. "Studies in Animal Locomotion: I. The Movement of Fish with Special Reference to the Eel". In: *Journal of Experimental Biology* 10.1 (1933), pp. 88–104 (cit. on p. 12).

[23] Shahir Hasanzadeh and Ali Tootoonchi. "Ground adaptive and optimized locomotion of snake robot moving with a novel gait". In: *Autonomous Robots* 28.4 (2010), pp. 457–470 (cit. on pp. 1, 8, 15, 18).

[24] Shigeo Hirose. *Biologically Inspired Robots: Snake-Like Locomotors and Manipulators*. Oxford University Press, 1993 (cit. on pp. 11, 12, 14, 22).

[25] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992 (cit. on p. 29).

[26] Scott Hooper. "Central Pattern Generators". In: *eLS*. John Wiley & Sons, Ltd, 2001 (cit. on p. 18).

[27] Brian Hunt, Ronald Lipsman, and Jonathan Rosenberg. *A Guide to MATLAB: For Beginners and Experienced Users*. 3rd ed. Cambridge University Press, 2014 (cit. on p. 6).

[28] Hadi Kalani, Alireza Akbarzadeh, and Javad Safehian. "Traveling Wave Locomotion of Snake Robot along Symmetrical and Unsymmetrical body shapes". In: *Proceedings of the 41st International Symposium on Robotics (ISR) 2010 and the 6th German Conference on Robotics (ROBOTIK)*. June 2010, pp. 1–7 (cit. on p. 18).

[29] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers". In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. Portland, Oregon, USA: ACM, 2010, pp. 119–126 (cit. on p. 33).

[30] C.R. Kothari. *Research Methodology: Methods and Techniques*. New Age International (P) Limited, 2004 (cit. on p. 4).

[31] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Matthias Steinbrecher, and Pascal Held. *Computational Intelligence: A Methodological Introduction*. Springer Publishing Company, Incorporated, 2013 (cit. on p. 31).

[32] Scott Lenaghan, Yongzhong Wang, Ning Xi, Toshio Fukuda, Tzyhjong Tarn, William Hamel, and Mingjun Zhang. "Grand Challenges in Bioengineered Nanorobotics for Cancer Therapy". In: *IEEE Transactions on Biomedical Engineering* 60.3 (2013), pp. 667–673 (cit. on p. 88).

[33] Jadran Lenarčič, Tadej Bajd, and Michael Stanišić. "Robot Grasp". In: *Robot Mechanisms*. Vol. 60. Intelligent Systems, Control and Automation: Science and Engineering. Springer Netherlands, 2013, pp. 291–311 (cit. on p. 22).

[34] Dan Lessin, Don Fussell, and Risto Miikkulainen. "Adapting Morphology to Multiple Tasks in Evolved Virtual Creatures". In: *Proceedings of The Fourteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14)*. 2014 (cit. on pp. 2, 47).

[35] Dan Lessin, Don Fussell, and Risto Miikkulainen. "Open-ended Behavioral Complexity for Evolved Virtual Creatures". In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. New York, NY, USA: ACM, 2013, pp. 335–342 (cit. on p. 2).

[36] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. "Analysis and Synthesis of Snake Robot Locomotion". In: *Snake Robots*. Advances in Industrial Control. Springer London, 2013, pp. 63–87 (cit. on pp. 11–13).

[37] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. "Analysis of Snake Robot Locomotion Based on Averaging Theory". In: *Snake Robots*. Advances in Industrial Control. Springer London, 2013, pp. 131–151 (cit. on pp. 1, 12, 15, 18).

[38] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. *Snake Robots: Modelling, Mechatronics, and Control*. Springer London, 2012 (cit. on pp. 1, 8).

[39] Hod Lipson. "Evolutionary Design and Open-Ended Design Automation". In: *Biomimetics*. CRC Press, 2005, pp. 129–155 (cit. on p. 2).

[40] Hamidreza Marvi and David Hu. "Friction enhancement in concertina locomotion of snakes". In: *Journal of The Royal Society Interface* 9.76 (2012), pp. 3067–3080 (cit. on p. 11).

[41] Kenneth McIsaac and James Ostrowski. "A Framework for Steering Dynamic Robotic Locomotion Systems". In: *The International Journal of Robotics Research* 22.2 (Feb. 2003), pp. 83–97 (cit. on pp. 1, 8, 18, 19).

[42] Kenneth McIsaac and James Ostrowski. “Motion Planning for Anguilliform Locomotion”. In: *IEEE Transactions on Robotics and Automation* 19.4 (Aug. 2003), pp. 637–652 (cit. on pp. 8, 12, 15, 19).

[43] Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 1st ed. O’Reilly Media, 2012 (cit. on p. 6).

[44] Kenneth Mclsaac and James Ostrowski. “A Geometric Approach to Anguilliform Locomotion: Modelling of an Underwater Eel Robot”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 4. May 1999, pp. 2843–2848 (cit. on p. 20).

[45] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998 (cit. on p. 30).

[46] Jared Moore. “Exploring Joint-Level Control in Evolutionary Robotics”. PhD thesis. Michigan State University, 2015 (cit. on p. 36).

[47] Jared Moore, Anthony Clark, and Philip McKinley. “Evolution of Station Keeping As a Response to Flows in an Aquatic Robot”. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. Amsterdam, The Netherlands: ACM, 2013, pp. 239–246 (cit. on p. 36).

[48] Jared Moore, Anthony Clark, and Philip McKinley. “Evolutionary Robotics on the Web with WebGL and Javascript”. In: *Proceedings of the Workshop on Artificial Life and the Web 2014, held in conjunction with the 14th International Conference on the Synthesis and Simulation of Living Systems*. ALIFE 14. New York, New York, 2013 (cit. on pp. 37, 38).

[49] Stefano Nolfi and Dario Floreano. *The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA, USA: MIT Press, 2000 (cit. on p. 2).

[50] Camilla Pandolfi, Tanja Mimmo, and Renato Vidoni. “Climbing Plants, a New Concept for Robotic Grasping”. In: *Biomimetic and Biohybrid Systems*. Ed. by Nathan Lepora, Anna Mura, Holger Krapp, Paul Verschure, and Tony Prescott. Vol. 8064. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 418–420 (cit. on p. 22).

[51] Geeta Patel, Gayatri Patel, Ritesh Patel, Jayvadan Patel, and Madhabhai Patel. “Nanorobot: A versatile tool in nanomedicine”. In: *Journal of drug targeting* 14.2 (2006), pp. 63–67 (cit. on p. 88).

[52] E. M. Purcell. “Life at low Reynolds number”. In: *American Journal of Physics* 45.1 (1977), pp. 3–11 (cit. on p. 88).

[53] Sebastian Risi, Daniel Cellucci, and Hod Lipson. “Ribosomal Robots: Evolved Designs Inspired by Protein Folding”. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. Amsterdam, The Netherlands: ACM, 2013, pp. 263–270 (cit. on p. 87).

[54] Matthew Rose, Anthony Clark, Jared Moore, and McKinley Philip. “Just Keep Swimming: Accounting for Uncertainty in Self-Modeling Aquatic Robots”. In: *Proceedings of the 6th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ELARS)*. Taormina, Italy, 2013 (cit. on pp. 33, 36).

[55] Martino Sabia and Cathy Wang. *Python Tools for Visual Studio*. Community experience distilled. Packt Publishing, 2014 (cit. on p. 6).

[56] Masashi Saito, Masakazu Fukaya, and Tetsuya Iwasaki. “Serpentine Locomotion with Robotic Snakes”. In: *Control Systems Magazine, IEEE* 22.1 (Feb. 2002), pp. 64–81 (cit. on pp. 1, 8, 11, 12, 14, 15, 18, 39).

[57] Gene I Sher. *Handbook of Neuroevolution Through Erlang*. Springer Science & Business Media, 2012 (cit. on p. 31).

[58] Karl Sims. “Evolving Virtual Creatures”. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. New York, NY, USA: ACM, 1994, pp. 15–22 (cit. on pp. 2, 37).

[59] Mike Snell and Lars Powers. *Microsoft Visual Studio 2012 Unleashed*. 2. Sams Publishing, 2012 (cit. on p. 6).

[60] Kenneth Stanley and Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies”. In: *Evolutionary Computation* 10.2 (June 2002), pp. 99–127 (cit. on p. 32).

[61] Aksel Transeth and Kristin Pettersen. “Developments in Snake Robot Modeling and Locomotion”. In: *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision (ICARCV ’06)*. Dec. 2006, pp. 1–8 (cit. on pp. 1, 8, 11, 12).

[62] Jeffrey Trinkle, Jacob Abel, and Richard Paul. “An Investigation of Frictionless Enveloping Grasping in the Plane”. In: *The International Journal of Robotics Research* 7.3 (June 1988), pp. 33–51 (cit. on p. 22).

[63] Darrell Whitley. “A Genetic Algorithm Tutorial”. In: *Statistics and Computing* 4 (1994), pp. 65–85 (cit. on p. 30).

[64] Mark Wineberg. “Statistical Analysis for Evolutionary Computation: An Introduction”. In: *Proceedings of the 2014 Conference Companion on Genetic and Evolutionary Computation Companion (GECCO)*. Vancouver, BC, Canada: ACM, 2014, pp. 345–380 (cit. on pp. 40, 41).

[65] Ke Yang, Xu-yang Wang, Tong Ge, and Chao Wu. "Simulation Platform for the Underwater Snake-Like Robot Swimming Based on Kane's Dynamic Model and Central Pattern Generator". In: *Journal of Shanghai Jiaotong University (Science)* 19.3 (2014), pp. 294–301 (cit. on pp. 1, 12, 15, 18).

[66] Changlong Ye, Shugen Ma, Bin Li, and Yuechao Wang. "Turning and Side Motion of Snake-like Robot". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*. Vol. 5. Apr. 2004, pp. 5075–5080 (cit. on pp. 1, 8, 18, 21).

Online sources

[67] Encyclopedia of Mathematics. *Inner product*. URL: http://www.encyclopediaofmath.org/index.php?title=Inner_product&oldid=29549 (visited on 06/27/2015) (cit. on p. 53).

[68] Encyclopedia of Mathematics. *Vector algebra*. URL: http://www.encyclopediaofmath.org/index.php?title=Vector_algebra&oldid=18802 (visited on 06/27/2015) (cit. on p. 53).

[69] Microsoft. *Python Tools for Visual Studio*. URL: http://microsoft.github.io/PTVS/ (visited on 06/09/2015) (cit. on p. 6).

[70] Microsoft. *Visual Studio*. URL: https://msdn.microsoft.com (visited on 06/09/2015) (cit. on p. 6).

[71] Elizabeth Pennisi. *Sidewinder robots slither like snakes*. 2015. URL: http://www.sciencemag.org (visited on 06/21/2015) (cit. on p. 11).

[72] Python Software Foundation. *About Python*. URL: https://www.python.org/about/ (visited on 06/09/2015) (cit. on p. 6).

[73] Russell Smith. *Open Dynamics Engine User Guide*. URL: http://ode.org/ode-latest-userguide.html (visited on 06/11/2015) (cit. on pp. 33–35).

[74] Hillebrand Steve. *Eastern garter snake slithers through a muddy area*. 2015. URL: http://www.public-domain-image.com (visited on 06/21/2015) (cit. on p. 11).

[75] The MathWorks, Inc. *MATLAB The Language of Technical Computing*. URL: http://www.mathworks.com/help/matlab/ (visited on 06/09/2015) (cit. on p. 6).

[76] Filip Tkaczyk. *Great Basin Gopher Snake using Rectilinear Locomotion*. 2015. URL: http://www.youtube.com (visited on 06/21/2015) (cit. on p. 11).

[77] Eric Weisstein. *"Cross Product." From MathWorld.* URL: http://mathworld.wolfram.com/CrossProduct.html (visited on 06/27/2015) (cit. on p. 53).