Evolving optimal humanoid robot walking patterns using genetic algorithms

Anders Johansson

May 25, 2006

Sammanfattning

Syftet med detta projekt är att utveckla ett sätt att med genetiska algoritmer automatiskt generera knärörelser som gör att människoroboten WABIAN-2 går så människolikt som möjligt. Bakomliggande teori inom genetiska algoritmer, robotik i allmänhet och WABIAN-2 i synnerhet presenteras.

Arbetet är uppdelat i fyra steg: Först läggs grunden för den genetiska algoritmen genom val av datarepresentation och implementation. Sedan definieras målet genom konstruktion av en fitnessfunktion som anger optimeringskriteriet, dvs. vad som anses människolikt. Varje funktion testas genom skapande av knärörelser som är så bra som möjligt enligt funktionens kriterium, och den funktion som skapat mest människolika mönster används. Det visar sig att en kombination av moment och rörelsemängdsmoment ger bäst resultat. Efter detta finjusteras den genetiska algoritmen genom att testa en mängd olika genetiska operatorer och parametrar tills en bra inställning hittats. Slutligen tillämpas metoden genom att använda ett mönster, som inspirerats av tidigare projektresultat, på roboten. Mönstret visar sig ge en jämnare och energisnålare gångstil.

Abstract

The purpose of this project is to develop a method of generating knee patterns making the humanoid robot WABIAN-2 walk as human-like as possible, using genetic algorithms. Relevant theory from the fields of genetic algorithms and Robotics, with special emphasis on WABIAN-2, is introduced.

The project is divided into four parts: First, the base of the genetic algorithm is built by choosing data representation and implementation. Then a fitness function specifying the optimization criterion, i.e. what is considered human-like, is constructed. Each function is tested by creating as good knee patterns as possible, according to the function criterion, and the function resulting in the most human-like patterns is used. The tests reveal that a combination of torque and angular momentum gives the best results. After this, the genetic algorithm is fine-tuned by testing different genetic operators and parameters until a good setup has been found. Finally, a walking pattern inspired by earlier project results is used on the robot. The pattern turns out to give a smoother and more energy efficient walking style.

I would like to thank Professor Atsuo Takanishi for making this project possible by accepting me to his laboratory at Waseda University in Tokyo, Japan. I would also like to thank Sweden-Japan Foundation for their generous donation making it possible for me to live in Japan, Stiftelsen Telefondirektören H. T. Cedergrens Uppfostringsfond for continuously supporting my university studies, and ABB Sweden for showing their interest in this project through their financial support.

Table of Contents

1 Background

The Japanese are very interested in robots. Nowhere else can you find as many robot companies and expositions, and beside their original business area, large Japanese companies such as Sony, Toyota, and Honda also make robots. One reason for the interest of these mechanical devices might be the comic book culture of Japan, where robots have figured as heroes and villains for decades. But serious robotics research is also performed.

Waseda University in Tokyo, Japan, has performed robotics research since 1964. In 2000, a group of professors formed the Humanoid Robotics Institute (HRI), with the goal of researching humanoid robotics, the interaction between humans and robots, and everything related to the field. Takanishi Laboratory, led by professor and HRI board member Atsuo Takanishi, has developed a robot capable of speaking like a human by means of an anatomically correct vocal tract and lungs, a walking wheelchair capable of climbing and descending stairs, and many other robots. But his most interesting robot project might be his humanoid robot.

2 Introduction

Robots can be very useful. They can perform tasks that are dangerous, difficult, or boring to humans, often faster and with greater precision. Many industrial robots exist today, building cars, computers, airplanes, and many other things. However, they generally need very special tools and well-defined non-changing work environments. They would hardly be useful in a home or office environment.

A humanoid robot, by contrast, has the basic anatomy of a human. Thus, it can work in environments designed for humans, using existing tools. A humanoid robot would be vastly more practical in a human society than the industrial counterpart. Theoretically, it should be capable of actions such as vacuum cleaning, lawn mowing or grocery shopping. But this introduces great engineering and controlling problems. Even one of the most basic human motions, walking, is extremely difficult to achieve. After decades of research, there are a number of walking robots, like Honda's Asimo and Sony's QRIO, but their walking is a far cry from human-like. A mathematical singularity present in all two-legged systems (also humans) prevent them from fully stretching their legs, thus impacting the range of possible motions. While humans can circumvent the problem, these robots are doomed to walk with bent knees, which uses more energy, puts more strain on the joints and reduces their maximum speed.

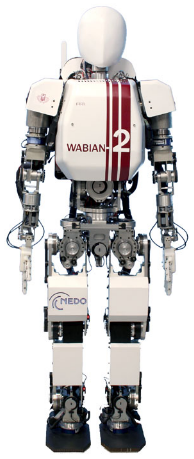


Figure 1 – The humanoid robot WABIAN-2

Takanishi's humanoid robot WABIAN-2 (WAseda BIpedal humANoid-2, see Figure 1), is a humanoid robot capable of knee-stretched walking. The long-term project goal is to explore the possibility of using a biped humanoid robot as a human motion simulator, specifically to help in developing a new walking assistance machine for elderly or handicapped people.

There are several advantages to using a robot instead of human test subjects. The testing task might be dangerous to people; it is also difficult to collect quantitative data such as joint torques or energy consumption from a human, but it is easy from a robot; and it is difficult for healthy people to simulate being handicapped well enough to develop walking assistance machines from their movements, while it is easy to just turn off or even remove a limb from a robot.

However, using a robot also presents a serious problem: It has to be programmed, and it is very difficult to program it to walk just like a human.

3 Problem specification

One way of circumventing the problem of manually programming a robot is to not program it at all, but instead automatically generate the desired movement. This was the purpose of this project, whose original assignment statement was "to develop a GA (genetic algorithm) software to compute the optimal lower-limb walking pattern in knee-extended bipedal walking under external force disturbances". Optimality in this case is the somewhat subjective notion of human-likeness, and the foot and waist motions are given as inputs, leaving the knee angles as the only free parameters of the walking pattern. The external force disturbances are any external forces such as the forces exerted on the robot by a walking assistance machine, but initially external forces were not considered for simplicity, and later for lack of time. Also, development of a full, stand-alone software was not necessary, because software for programming the robot (the pattern generator) was available, already implementing a kinematic model of the robot.

Thus, more specifically, the assignment was to extend an existing robot programming program by adding the capability of automatically generating human-like knee angles using genetic algorithms.

4 Theory

Building and controlling a robot is an interdisciplinary project, requiring knowledge from fields such as mechanical engineering, electronics, computer science, control theory and others. Theory relevant to this project is human walking, robotics, and genetic algorithms.

4.1 Notes on conventions

Some notes has to be made about the symbols and expressions used in this report. First of all, there is bound to be confusion about the knee angles. By convention from the robot design, the knee angles are measured by negative angles from $0°$ (straight leg) to a maximum knee bend of $-160°$, while the more intuitive measure of knee bending amount uses positive angles. Therefore, when this report talks about increasing the knee bend or the knee angle, the real angle value is actually decreased. The text in this report uses the intuitive measure while graphs show the design measure for correctness. Perhaps it is easiest to think of absolute values of all angles. Also, sometimes the curious expression "stretched knee" is used, meaning stretched leg. This is also by convention, and to remind the reader that the knee angle is responsible.

Another word commonly and knowingly misused is "waist" instead of pelvis. This is originally a translation mistake made by the Japanese designers. The waist link in the robot connects the hips to each other and to the upper body. Following this convention, expressions such as waist position and waist movement are used in this report.

Boldface is used to indicate vectors and matrices, such as $\mathbf{v}$ or $\mathbf{R}$. All matrices, but also some vectors ($\mathbf{F}$ and $\mathbf{N}$ for force and moment), are written with upper case letters; the type should be clear from context. Because we will be dealing with many different coordinate systems, a leading superscript is used to specify the frame of reference of the quantity, and indices are following subscripts. Examples are ${}^{3}\mathbf{r}_{4}$ (the position of joint 4 expressed in the coordinate system of joint 3) or ${}^{c_7}\mathbf{I}_{5}$ (the moment of inertia matrix of link 5 expressed in the coordinates of the center of mass of link 7). The index c_i stands for the center of mass of link i. Rotation matrices $\mathbf{R}$ and frame operators $\mathbf{T}$ have their indices on the left to save space, for example ${}^{3}_{4}\mathbf{R}$ (the rotation of the coordinate system of joint 4 as seen from the coordinate system of joint 3) instead of ${}^{3}\mathbf{R}_{4}$.

Symbols are used with their normal meaning, for example position $\mathbf{r}$, velocity $\mathbf{v}$, and acceleration $\mathbf{a}$. The one-dimensional joint angular velocities and accelerations are written $\dot{\theta}$ and $\ddot{\theta}$ while the three dimensional link angular velocities and accelerations are $\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$.

On a final note, a GA consists of functional blocks called *operators* by convention, such as the selection operator and the mutation operator. Although called operators also in this report, their implementation will be as normal functions. In C++ operators are just functions with special calling, so the GA operators could have been implemented as actual operators, for example `operator+` for crossover or `operator!` for mutation, but this would only be unintuitive and create confusion.

4.2 Human locomotion

It is difficult to define what "normal" human walking is. Every person has his or her own walking style, but there are some common characteristics. Ayyappa [1] describes normal walking from the perspective of prosthetics and orthotics research, but the ideas are general.

The gait cycle is the period of time between two identical events in the walking cycle. Stance is defined as the time when the foot is in contact with the ground, and swing is when the foot is in the air. Single support is when one foot is in contact with the ground, and double support is when both feet are in contact with the ground. Using these terms, the gait cycle for one leg can be divided into four phases: Initial double support, single support, terminal double support, and swing. During normal walking, double support constitutes about 25% of the gait cycle. With increased velocity, the amount of double support decreases; running constitutes forward motion with no period of double support. The forward distance between a floor-contact point of a foot and the same floor-contact point of the other foot is called the step length; the distance between two successive floor-contact points for the same foot is the stride length. Thus the stride length is about twice the step length.

The waist and ankles are also largely involved in the smooth continu-

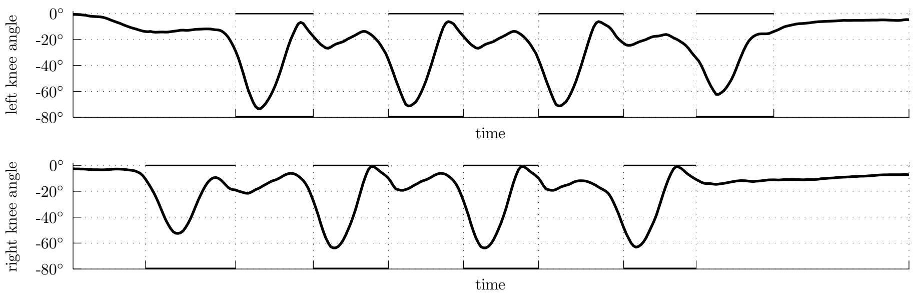


Figure 2 – Human knee trajectories for starting, walking eight steps and stopping. The marked regions are the swing phases; double support is not indicated.

ous motion of walking, keeping balance and minimizing the movement of the center of gravity and therefore the energy expenditure. Lateral pelvic displacement (sideways waist motion) improves the position of the center of gravity above the supporting limb, and lateral pelvic tilt (waist roll motion) prevents an excessive rise in center of gravity during midstance by dropping on the unsupported side. Pelvic transverse rotation (waist yaw) moves the waist forward on the swing side and backward on the stance side, effectively extending the step length. Ankle pitch motion also smooths the pathway of the center of gravity and extends the step length by stretching the foot to allow for early heel support during initial stance and the other way for extra toe support during final stance.

The bending of the knees during walking is closely related to the ankle motion, with the purpose of smoothing the movement of the center of gravity but also to absorb some of the shock forces from the foot hitting the ground. Stretching the leg allows it to function as an inverse pendulum, moving the body forward. Thus the knee bends during loading response, extends during midstance and terminal stance, and bends to its maximum angle during initial swing to clear the foot from the ground. This characteristic knee pattern is called *double knee action* because of the two bends during each gait cycle. A typical knee trajectory for walking eight steps (including start, walking and stop) are shown in Figure 2. This data was collected using Motion Capture on a Japanese male lab member.

4.3 Robotics

Link orientation is very important in robotics[1]. Therefore, rotations are a central part of robotics theory. Any rotation can be described by an axis-of-rotation vector and an angle. The axis vector is constrained by the requirement that it is of unit length, meaning that in $\mathbb{R}^3$ a rotation is defined by only three parameters. Equivalently, the three parameters can be the angles in a composition of rotations about three predefined axes (Euler Angles).

However, rotating an object by an angle θ around some axis $\mathbf{k}$ is also equivalent to rotating the coordinate system by $-\theta$ around $\mathbf{k}$. The rotated base vectors can be written as column vectors in a 3×3 rotation matrix $\mathbf{R}$, which acts as a matrix operator. Matrix multiplication of such a matrix with a vector will map the vector onto the inversely rotated space, equivalent to performing the desired rotation by θ around $\mathbf{k}$. Because rotation matrices consist of vectors that by definition are orthogonal and of unit length, the matrix is orthonormal, which means it is invertible and that $\mathbf{R}^{-1} = \mathbf{R}^T$. Generally, rotations do not commute, which is to be expected since they are described by non-commuting matrix operators.

A very common type of rotation is around a base vector of a coordinate system. To refer to such rotations, special names adapted from aeronautics are used. A rotation around the longitudinal (front-back) axis is called a *roll*, a rotation around the lateral (right-left) axis is called a *pitch* and a rotation around the vertical (up-down) axis is called a *yaw*.

A robot can be constructed in almost any way imaginable. Many robots consist of a chain of rigid links i, connected by revolute joints j (rotational joints with one degree of freedom θ_j). Several joints can be connected by links with zero length along the axis of rotation, effectively creating a several-degrees-of-freedom rotational joint. One end of the link chain is firmly attached to the ground while the position and orientation of the other end can be controlled by varying the joint angles. The joint controlling a specific link is called its actuating joint.

To define and relate the positions and orientations of the links of a robot, each link i has its own coordinate system $\{j\}$ located at the position of its actuating joint j. This coordinate system can be described as a rotation ${}^{j_0}_{j}\mathbf{R}$ and translation ${}^{j_0}\mathbf{r}_j$ of any other coordinate system $\{j_0\}$. This combination of a position vector ${}^{j_0}\mathbf{r}_j$ and a rotation matrix ${}^{j_0}_{j}\mathbf{R}$ is called a *frame* and is written ${}^{j_0}_{j}\mathbf{T}$. A frame completely describes the position and orientation of a

[1]Consider, for example, a robot trying to fix a screw with a screwdriver. Even with the tip of the screwdriver touching the dent in the screw, the robot might fail if the screwdriver is not oriented parallel to the screw axis!

link in relation to those of another link (or to the universe frame, {U}).

Frames are used as operators with the properties

$$\begin{aligned}
{}^{\mathrm{B}}_{\mathrm{A}}\mathbf{T}\,{}^{\mathrm{A}}\mathbf{v} &= {}^{\mathrm{B}}_{\mathrm{A}}\mathbf{R}\,{}^{\mathrm{A}}\mathbf{v} + {}^{\mathrm{B}}\mathbf{r}_{\mathrm{A}} = {}^{\mathrm{B}}\mathbf{v},\\
{}^{\mathrm{A}}_{\mathrm{B}}\mathbf{T}\,{}^{\mathrm{B}}_{\mathrm{C}}\mathbf{T} &= \{{}^{\mathrm{A}}_{\mathrm{B}}\mathbf{R}\,{}^{\mathrm{B}}_{\mathrm{C}}\mathbf{R},\ {}^{\mathrm{A}}\mathbf{r}_{\mathrm{B}} + {}^{\mathrm{A}}_{\mathrm{B}}\mathbf{R}\,{}^{\mathrm{B}}\mathbf{r}_{\mathrm{C}}\} = \{{}^{\mathrm{A}}_{\mathrm{C}}\mathbf{R},\ {}^{\mathrm{A}}\mathbf{r}_{\mathrm{C}}\} = {}^{\mathrm{A}}_{\mathrm{C}}\mathbf{T},\\
{}^{\mathrm{A}}_{\mathrm{B}}\mathbf{T}^{-1} &= \{{}^{\mathrm{A}}_{\mathrm{B}}\mathbf{R}^{T},\ -{}^{\mathrm{A}}_{\mathrm{B}}\mathbf{R}^{T}\,{}^{\mathrm{A}}\mathbf{r}_{\mathrm{B}}\} = \{{}^{\mathrm{B}}_{\mathrm{A}}\mathbf{R},\ {}^{\mathrm{B}}\mathbf{r}_{\mathrm{A}}\} = {}^{\mathrm{B}}_{\mathrm{A}}\mathbf{T}.
\end{aligned}$$

These relations can be derived and compactly expressed by writing frames as 4×4 matrices and vectors as 4×1 vectors in homogeneous coordinates,

$${}^{\mathrm{B}}_{\mathrm{A}}\mathbf{T} = \begin{bmatrix} {}^{\mathrm{B}}_{\mathrm{A}}\mathbf{R} & {}^{\mathrm{B}}\mathbf{r}_{\mathrm{A}} \\ 0 & 1 \end{bmatrix}, \quad {}^{\mathrm{A}}\mathbf{u} = \begin{bmatrix} {}^{\mathrm{A}}\mathbf{v} \\ 1 \end{bmatrix},$$

however the explicit relations are computationally more efficient. Strictly speaking, a frame could be completely defined using only six parameters: three for the rotation, three for the position. Therefore, the minimum number of degrees of freedom required for a robot to be able to achieve any (reachable) position and orientation in $\mathbb{R}^3$ is 6.

Many robots with one-dimensional revolute joints use electrical motors to actuate the joints by applying torques, which means that the robot is controlled by modifying the joint angles θ_j spanning the *joint space*. However, the desired motion is usually defined by ${}^{0}_{n}\mathbf{T}$, the position and orientation of the last link as seen from the base, in Cartesian space. It is relatively easy to construct all the frames from the joint angles, a process called *direct kinematics*. The reverse process, calculating the angles needed to position the end link correctly, is called *inverse kinematics*. It is a nonlinear problem, and is much more difficult to solve.

The workspace is defined as the space of all frames reachable by the end effector. When the robot reaches the edge of the workspace, or when two or more axes of rotation line up, the number of degrees of freedom is effectively reduced. This is called a *singularity* of the mechanism. Near a singularity, inverse kinematics cannot be solved and the joint velocities go to infinity.

To compute the joint forces and torques for a robotic link system, the *Iterative Newton-Euler Dynamic formulation* (INED) can be used [4]. The calculation is split in two parts, first *outward* starting at the base and propagating velocities and accelerations to the free end, then *inward* from the end link propagating the joint forces and torques back to the base. For each outward iteration step we have to calculate the quantities

$$\begin{aligned}
{}^{j}\boldsymbol{\omega}_j &= {}^{j}_{j-1}\mathbf{R}\,{}^{j-1}\boldsymbol{\omega}_{j-1} + \dot{\theta}_j\,{}^{j}\hat{\mathbf{z}}_j \\
{}^{j}\boldsymbol{\alpha}_j &= {}^{j}_{j-1}\mathbf{R}\,{}^{j-1}\boldsymbol{\alpha}_{j-1} + {}^{j}_{j-1}\mathbf{R}\,{}^{j-1}\boldsymbol{\omega}_{j-1} \times \dot{\theta}_j\,{}^{j}\hat{\mathbf{z}}_j + \ddot{\theta}_j\,{}^{j}\hat{\mathbf{z}}_j \\
{}^{j}\mathbf{a}_j &= {}^{j}_{j-1}\mathbf{R}\left({}^{j-1}\boldsymbol{\alpha}_{j-1} \times {}^{j-1}\mathbf{r}_j + {}^{j-1}\boldsymbol{\omega}_{j-1} \times ({}^{j-1}\boldsymbol{\omega}_{j-1} \times {}^{j-1}\mathbf{r}_j) + {}^{j-1}\mathbf{a}_{j-1}\right) \\
{}^{j}\mathbf{a}_{c_i} &= {}^{j}\boldsymbol{\alpha}_j \times {}^{j}\mathbf{r}_{c_i} + {}^{j}\boldsymbol{\omega}_j \times ({}^{j}\boldsymbol{\omega}_j \times {}^{j}\mathbf{r}_{c_i}) + {}^{j}\mathbf{a}_j \\
{}^{j}\mathbf{F}_i &= m_i\,{}^{j}\mathbf{a}_{c_i} \\
{}^{j}\mathbf{N}_i &= {}^{c_i}\mathbf{I}_i\,{}^{j}\boldsymbol{\alpha}_j + {}^{j}\boldsymbol{\omega}_j \times {}^{c_i}\mathbf{I}_i\,{}^{j}\boldsymbol{\omega}_j,
\end{aligned}$$

and the equations for each inward iteration step are

$$\begin{aligned}
{}^{j}\mathbf{f}_j &= {}^{j}\mathbf{F}_i + {}^{j}_{j+1}\mathbf{R}\,{}^{j+1}\mathbf{f}_{j+1} \\
{}^{j}\mathbf{n}_j &= {}^{j}\mathbf{N}_i + {}^{j}_{j+1}\mathbf{R}\,{}^{j+1}\mathbf{n}_{j+1} + {}^{j}\mathbf{r}_{c_i} \times {}^{j}\mathbf{F}_i + {}^{j}\mathbf{r}_{j+1} \times {}^{j}_{j+1}\mathbf{R}\,{}^{j+1}\mathbf{f}_{j+1} \\
\tau_j &= {}^{j}\mathbf{n}_j^T\,{}^{j}\hat{\mathbf{z}}_j.
\end{aligned}$$

$\boldsymbol{\omega}$ and $\boldsymbol{\alpha}$ are the total angular velocity and acceleration vectors, $\dot{\theta}$ and $\ddot{\theta}$ are the scalar joint angular velocity and accelerations, $\hat{\mathbf{z}}$ is the axis of rotation, $\mathbf{r}$ is the position vector, $\mathbf{a}$ is the linear acceleration, m is the link mass, $\mathbf{F}$ and $\mathbf{N}$ are the force and torques on the link due to its motion only, $\mathbf{f}$ and $\mathbf{n}$ are the final joint force and torque vectors, and τ is the scalar torque that actually does work. $\mathbf{I}$ is the moment of inertia tensor and $\mathbf{R}$ is a rotation matrix.

But humanoid robots are more complicated than conventional robots in that they do not have a fixed base frame: It alternates between the left and right foot during walking, and when the robot is standing it instead has *two* base frames. Also, they do not consist of a single link chain. At the waist, the chain splits in two, one down the swinging leg and one up the upper body. For these reasons, the numbering of links and joints is complicated.

Therefore, in the above Iterative Newton-Euler equations, $j-1$ conceptually means "the preceding joint", and $j+1$ means "the succeeding joint", as seen from the base. This is important because the base, where iterations start, can be the right *or* the left foot, and the directions of "outward" and "inward" iterations depend on which one. The directions change for each step of the robot, because it is then supported by the other foot. The link index i is the link actuated by joint j, which also depends on what foot is currently supporting the robot.

4.4 Genetic algorithms

A genetic algorithm (GA) is a method of solving an optimization problem by searching the space of possible solutions $\mathbb{S}$ for solutions that minimize or maximize a given objective function $f : \mathbb{S} \rightarrow \mathbb{R}$. For many real-world problems, the search space is incredibly large and an exhaustive search is impossible. A common approach is to traverse the search space more or less randomly, following some heuristic to try to get to good solutions quickly. The heuristics guiding a GA are inspired by Darwin's biological theory of evolution.

In nature, individuals with beneficial genetic traits (highly fit individuals) are more likely to survive and reproduce, thus increasing the frequency of those traits in the next generation. To simulate this process, a GA keeps a population of virtual individuals, each of which has a potential solution encoded in its genes. The solutions are evaluated by the objective fitness function, assigning a fitness value to each individual. Based on the fitnesses, some individuals are selected for mating and allowed to reproduce. The offspring form a new population and the process is repeated, producing better and better solutions with each generation.

A gene is the smallest unit of information encoded in the solution. The genes are arranged in one or more chromosomes in the individuals, and its total collection of genes is its genome. The position of a gene is called its locus, and is usually important for the decoding of the gene (for example, if the genes encode a number by its digits, rearranging the genes changes the number). Locus-independent encodings are called "messy" encodings because the genes can be arranged in any way possible.

Genetic algorithms were invented by John H. Holland and his colleagues at the University of Michigan in the 1960s. They encoded data as binary strings in the genes, and invented several genetic operators to work on them: The roulette wheel selection operator chose the individuals allowed to mate, and reproduction was performed by a one point crossover operator, combining the genes of two individuals, and a random mutation operator, introducing random changes to the offspring. An inversion operator reordered the genes to try to form short-order schemata. This GA setup is called the Simple Genetic Algorithm, or SGA.

Goldberg [5] describes the theoretical basis for genetic algorithms by the building block hypothesis. Introducing schemata as partially defined binary strings representing the subset of all strings that share the same characteristics, calculations about the evolution of the genes under the GA operators were made. The idea is that partial solutions should form short schemata and combining them should make a better solution. Statistically, if the solution can be built by short-order (i.e. specialized) high fitness schemata, these building blocks will form better and better solutions each generation. Long, general schemata have a large probability of being destroyed by crossover and mutation operations and thus do not contribute much. The undefined parts of the schemata allow for an implicit parallelism. Of course, if the solution to the problem can not be constructed by partial solutions or if there are no meaningful short-order schemata, the GA would perform no better than a random search through the search space. To accelerate the potential creation of short-order schemata, the inversion operator, shuffeling the genes, is used.

Binary strings are not always a good choice for data representation. Many problems are real-valued, suggesting a real-valued representation, but some purists point out that larger alphabets require a larger population size to have individuals representing all strings and that binary representation should always be used. One disadvantage of using binary strings is that a single binary mutation might change the value by a very large amount, where a real-valued representation can control the mutation more.

The *selection operator* chooses an individual for mating. The choice of selection operator controls the selection pressure: too low pressure means slow convergence, too high pressure may lead to premature convergence (i.e. loosing diversity and getting stuck in a local optimum). Holland originally used fitness-proportionate selection, commonly implemented as "roulette wheel selection": A virtual roulette wheel is spun, each individual representing a slice equal in size to the proportion of its fitness to the total fitness of all individuals. The probability for individual k to be selected is $P(k) = f_k / \sum_i f_i$. But roulette wheel selection, or fitness-proportionate selection in general, has several large disadvantages. Firstly, it selects individuals that maximize the fitness, while many problems need minimization. This is not trivially circumvented, because the two common fixes $F' = -F$ violates the requirement of only positive fitnesses, and $F' = \frac{1}{F}$ changes the fitness distribution, more strongly favoring very low F. More importantly, any early super-fit individuals will get selected most or all of the time, eliminating diversity, leading to premature convergence in a local maximum. Also, toward the end when most individuals are similar, or if there is a fitness offset reducing the relative difference between fit and unfit individuals, the selection pressure is lowered making convergence slow.

An alternative selection operator without these shortcomings is tournament selection, in which k randomly sampled individuals compete to be selected. The best of them is selected with probability p, the second best with probability $p(1-p)$, the third $p(1-p)^2$ etc. In the unlikely event of probability $(1-p)^k$ that they are all rejected, a new random sample of k is chosen to compete for the selection. The parameters k and p become parameters of the GA. Other selection methods include sexual selection, in which individuals are pairwise selected because they are fundamentally different, leading to cooperative specialization (e.g. males hunt, females care for children); or random selection which randomly selects an individual regardless of the fitness distribution.

Often, to stabilize the progress of the GA, one or more of the best individuals of each generation are inserted unchanged into the new generation, avoiding any potentially damaging crossover, or mutation operators. This makes sure that the best individual of the next generation is at least as good as the previous generation. Thus, once a certain best fitness is achieved, it cannot be lost and the fitness can only improve further or remain unchanged. These individuals are called *elite individuals*. However, in this project, the elite individuals will be subject to the survival operator, to give even better children a greater survival chance.

The *crossover operator* is responsible for combining the chromosomes of two (or more) individuals to produce offspring. The most common example is one point crossover, in which the genome of each parent is cut at a random position, before which the child get the genetic material from one parent and after which it gets it from the other. Often, a second child is also created, getting the complementary genes, so that no genetic material is lost between the populations. Another common method is uniform crossover, in which the child gets each gene from one parent or the other with equal probabilities, thus copying random pieces of the chromosomes of both. The complementary bits are again given to the second child. Other crossover operators are arithmetic crossover, in which the genes are not distributed but arithmetically combined, for example through averaging the parents' genes. n-point crossover is similar to one point, but n cuts are made in the genomes and the child gets pieces alternately from the parents. Multisexual crossover is also possible, where more than two parents share their genes to create offspring.

If the individuals have two chromosomes having the same basic structure, it might be beneficial to be able to share genetic material between the them. This can be done by a swap operation before crossover. If enabled, there is a 50% chance that the first chromosome of one parent is crossed over with the second chromosome of the other parent, and vice versa. In the other 50%, or if swap is not enabled, a first chromosome is always crossed over with a first, and a second with a second. This argument can also be generalized to more than two chromosomes.

The *mutation operator* randomly changes or introduces new data in the chromosomes, partly to introduce new good changes and partly to make sure no good changes are lost for good, if accidentally removed during crossover or the individual is not selected. Mutation operates on each single gene with a certain probability p_{mut}. The modification can typically be random, changing the gene to a completely new setting, or creeping, applying a small change to the gene. A mutation operator specifically invented for the problem in this report is “block mutation”, where the same creeping mutation is applied to a random range of genes. In addition to these modification algorithms, genes can also be randomly added to or removed from the genome, by repeatedly deleting random genes with probability p_{mut}, and then repeatedly insert new genes with probability p_{mut}.

The *survival operator* determines who gets to form the next generation. This is usually considered part of the selection operator, but survival has a completely different function and is in this report separated as an operator of its own. The most commonly used form of survival is Children, in which all individuals in a generation die, and all their children form the next. Another survival operator is Invasion, in which only children survive but are then invaded by new, random individuals, killing and replacing the lowest-fitness individuals. The parameter p determines how many of the original individuals survive the attach, typically set at $p = 50\%$. This should add diversity for small populations. The $(\mu + \lambda)$-survival operator lets the old generation and the children together compete for space in the next population, allowing fit parents to survive and reproduce again.

To add further diversity to small population sizes, the differentiation operator can be used as a last step of the survival operator. It kills all individuals having the same fitnesses to within 10^{10}, assumed to be clones, except one. The holes in the population are filled with new, random individuals just like with invasion survival.

The *inversion operator* reverses a random portion of the genes, in the hope of randomly bringing related genes close so they can form building blocks. It has been discussed whether this operator contributes to the performance of the GA or not, and it seems any effects would be visible only in long GA runs. Two positions are randomly chosen and the genes between them are reversed by repeatedly swapping genes and moving inward. If the gene has length n, a linear inversion operator would reverse position k with probability $P(k) = \frac{(1+k)(n-k)}{n^2}$, thus reversing the midsection more often than the ends. Instead, a uniform distribution was achieved using a circular inversion operator (i.e. connecting the ends of the genome, to allow wrapping around the edges). In this case, each position is reversed with equal probability $P(k) = \frac{n(n+1)}{2n^2}$.

Usually, crossover and mutation are performed with certain probabilities, to allow some individuals to enter the new generation unchanged. Typically, the crossover probability p_{cro} is rather high, around 60–70%, because crossover is needed to advance the evolution. The mutation probability p_{mut} is usually low, around 1% or less. A higher mutation probability introduces more changes, which might be beneficial early in a GA run, but might also destroy already good individuals. A too high mutation probability makes the GA work more like a random walk, randomly traversing the search space without direction.

Several research teams have used this method to generate optimal gait patterns for robots before. For example, Capi et al used a genetic algorithm to generate a gait pattern for a five-link biped robot and trained a Radial Basis Function Neural Network with the results [3], and Bessonet et al used another optimization technique but used a similar evaluation function, targeted at minimizing the joint torques for a seven-link planar biped [2]. Using a GA to solve this kind of problem might work well, because GAs are especially well suited to solve difficult problems without prior knowledge of the shape of the solutions. However, there are important design choices affecting the effectiveness or the method, such as fitness function design, data representation, and the GA parameters.

5 WABIAN-2 and Pattern Generation

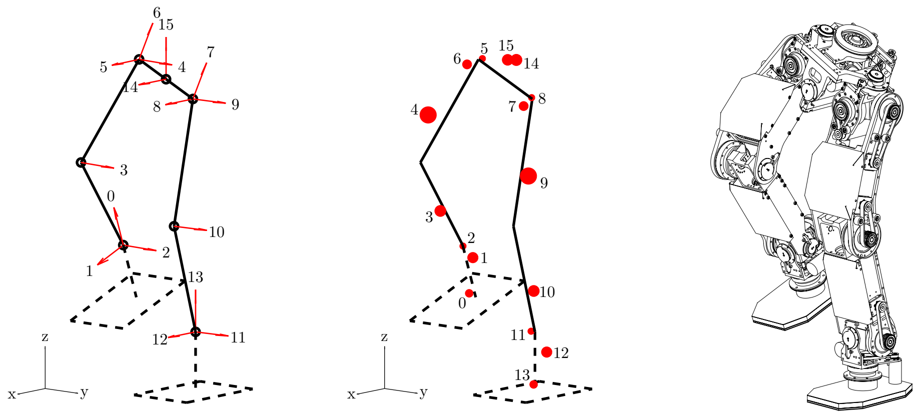
5.1 Robot design

WABIAN-2 is a humanoid robot with 41 degrees of freedom (DOF): 7 in each leg, 2 in the waist, 2 in the trunk, 3 in the neck, 7 in each arm and 3 in each hand. "He" stands 1.53 m tall and weighs 64.5 kg with batteries.

Figure 3 shows the joint and link design of the lower limbs of the robot. Only links 3, 4, 9, 10 and 14 have non-zero lengths. Note that link 14 (waist) rigidly connects joints 6 and 7 (hips); joints 14 and 15 only control the orientation of the upper body.

The world frame for the robot is a right-handed coordinate system with the positive x axis pointing in the robot's forward direction, the y axis pointing to its left and the z axis pointing up. A rotation around the x axis is a roll, around the y axis is a pitch and around the z axis is a yaw.

To be able to place the foot in any (reachable) position and orientation, a minimum of six degrees of freedom is required. Therefore, a common setup for a robot leg has three degrees of freedom in the hip (roll, pitch, yaw), one in the knee (pitch), and two in the foot (roll and pitch). But when the knee is stretched, one degree of freedom is lost because the foot can no longer be moved radially from the hip. Division by zero makes other joints' angular velocities go to infinity in an attempt to compensate; a singularity has occurred. To avoid this singularity, most robots walk with bent knees. WABIAN-2 instead uses predetermined knee trajectories, so that the legs can be considered rigid structures at any point in time. This effectively reduces the lower limbs by 2 degrees of freedom and avoids the singularity because



(a) Axes of rotation **(b)** Mass points **(c)** Actual model

Figure 3 – Design of the lower limbs of the robot. Figure 3a shows the axis of rotation for each joint, and Figure 3b shows the center of mass position for each link. Figure 3c shows the actual model.

inverse kinematics is not used for the knee joints. To mimic human anatomy, WABIAN-2 also has two extra degrees of freedom in the waist (roll and yaw) and one for each foot (yaw). The waist roll can be used to compensate for the loss of degrees of freedom in the knees, the waist yaw can extend the step length like in humans, and the foot yaw can be used for obstacle avoidance. Thus WABIAN-2 has 16 degrees of freedom in the lower limbs.

However, there is one large difference between a human and WABIAN-2. A human uses the toes, heel and foot pitch to create a smooth walking, while the robot has solid, flat feet that are typically (although not necessarily) parallel to the ground.

5.2 Robot control

WABIAN-2 carries a computer, running a tight loop that controls the robot motors. Time is discretized into regular intervals (usually set at 30ms) called *phases*. There are typically 512 phases in a normal walking pattern, thus lasting 15.36s. The movement of a joint is called its *trajectory*, describing the relevant joint properties at each phase. Several trajectories combined as a functional unit constitute a *pattern*. For example, a left foot trajectory and a right foot trajectory form a gait pattern, describing the steps to be walked by the robot. The combination of a gait pattern, a knee pattern, and a waist pattern (together with upper limb patterns such as hand, trunk and head patterns, which are not considered in this report) completely describes the motion of the robot and is called a *walking pattern*.

Walking patterns for WABIAN-2 are created in the pattern generator (PG) program. First, the gait pattern is defined by specifying key foot positions and orientations, and the trajectories are constructed by interpolation resulting in smooth movements. The waist position and orientations are automatically calculated, but can be modified if specific waist movements are desired. Next, the knee trajectories are set by specifying key left and right knee angles and interpolating. The goal of this project is to replace this step with an automatic GA calculation. After setting the knee pattern, the Zero Momentum Point (ZMP) trajectory, used for balance, is automatically calculated. As an optional final step, the waist and trunk compensatory motions can be calculated. This is necessary for real-world patterns because the movements of the legs during walking exert momentums on the robot, making it twist or even fall over. This can be compensated by exerting an opposite momentum, e.g. by moving the arms. However, patterns intended for computer simulation only can make an approximation by skipping this computationally expensive step.

For each phase in the control loop, the foot positions and the knee angles (along with knowledge of shin and thigh link lengths) are used to calculate the hip positions. The waist link is then positioned between the hips at the correct angle, giving the full posture for the phase, on which inverse kinematics is used to find the required joint angles. Because of this, however, it is possible to create "impossible" patterns, where the distance between the hip joints is larger than the waist link length. Thus it is important to make sure that the foot positions and the knee angles always result in a valid posture.

During double support, the robot supports its weight on both the right and left foot. This makes the mechanical system overdetermined and the finite element method or another similar method has to be used to calculate the forces and torques. This lies outside the boundaries of this project, but double support could not be ignored because the robot might collapse if totally unconstrained. Therefore, an approximation was made by running the Iterative Newton-Euler calculations twice for each double support phase: First as if the robot is actually standing on the right foot, then again as if it is standing on the left. To avoid relying too much on this crude approximation, gait patterns keeping double support a minimum were used in this project.

6 Experiments and Results

The project was divided into four different steps. First, the GA base was built by designing the data structure, representation and interface to the PG program. Then, a fitness function with the desired properties was constructed and implemented. After that, the GA was fine tuned to produce as fast and reliable results as possible. Finally, a GA-constructed knee pattern was tested on the robot, comparing it with an existing, manually constructed knee pattern.

6.1 Building a base

The Pattern Generation program is written in C++ using the programming environment Borland® C++Builder™ 6. Several classes were constructed to keep the different data structures.

6.1.1 The class structure

The class `CGeneration` contains a STL[2] vector of pointers to `CIndividual` objects, thus describing a full population of individuals and providing a virtual environment for them to reproduce and generate increasingly better solutions. The class `CIndividual` represents an individual.

Every individual holds a potential solution; in this case two knee trajectories, one for the left and one for the right knee. Each trajectory defines a knee angle for every phase, but most walking patterns have more than 300 phases, so encoding the angles as an array and optimizing every angle independently would make the problem more than 600-dimensional. Instead, trajectories are encoded in the individuals as a series of *midpoints*, each specifying a phase and an angle. These are then connected by cubic spline interpolation and sampled at every phase to create the full angle sets only when needed. Thus, a trajectory is a chromosome in an individual, and a midpoint is a gene in a chromosome. Using this messy gene encoding also simplifies the implementation of some genetic operators, because the genes are independent of their locus.

Implementation-wise, a trajectory is represented by the class `CTrj`. It contains a STL vector of pointers to `MPnt` midpoint objects, each containing a (phase, angle)-pair for the trajectory. The trajectory class has functions for counting, adding, retrieving and removing midpoints from the vector, as well as setting and getting the `changed` status used to avoid recalculating the fitness if no changes has been made to the trajectory. The interface

[2]Standard Template Library

to the rest of the pattern generator program is though the `updateTrajBase` function, which updates an existing `CTrajectryBase<Vec1d>` (an internal PG data type) with the `CTrj` trajectory data. This is unfortunately needed before every fitness calculation, to create the sampled spline data from the midpoints.

6.1.2 Introducing genetic code

The `CGeneration` class has functions for getting the size of and reserving space in the population, and for adding and retrieving individuals. But there are also genetic functions: `select` chooses an individual for mating through roulette wheel or tournament selection. The `crossover` function passes on to the `CTRj::crossover` function to perform crossover on the individual trajectories. The most important function, `evolve`, is the heart of the GA. It creates a new generation from the current by first cloning the elite individual(s) to a child pool, then repeatedly selecting two parents, performing crossover to create two new child individuals, performing mutation on each child, and adding them to the child pool, until the same number of children and parents exist. Finally, it calls the `survive` function with child pool and the parent generation as arguments, which returns the survivors as a new, hopefully better, generation.

Most of the genetic operator code is in the trajectory class `CTrj`. The `crossover` function takes pointers to two parent and two child `CTrj` trajectories, and performs one point or uniform crossover by adding copies of the parent `MPnt` midpoints to the child trajectories. The `mutate` function performs random, creeping or block mutation on the trajectory. For creeping and block mutation, the midpoints are randomly moved by $|\Delta\theta| \leq 5^\circ$ and $|\Delta p| \leq 2$ phases. The `invert` function simply chooses two points and swaps the pointers for all indices between them, effectively reversing a part of the genome.

Finally, some pass-on functions exist in the class `CIndividual`: The `mutate` and `invert` functions pass on to `CTrj::mutate` and `CTrj::invert` for each trajectory. This is because the functions should be called for the individual, but the actual calculations work on the trajectories.

Controlling the operators and parameters used by the GA is done by global variables, which are changed throughout this report. Some parameters are set in run-time by the user on the GA form of the PG program (not discussed in this report); these are the population size (default 20), the elite size (default 1), crossover probability (default 60%) and mutation probability (default 1%).

6.1.3 The laws of physics

The GA-specific class `GALowerLimb` is a modified version of the PG class `CWABIAN2LowerLimb`, a complete kinematic model of the lower limbs of the robot. The purpose of this modified class is to calculate the fitness for a given walking pattern by simulating the robot motion and recording the appropriate quantities. When a `GALowerLimb` object is created, which is done only once after the foot and waist trajectories have been set, the constructor extracts necessary kinematic data from the provided `CWABIAN2LowerLimb` object, and foot and waist trajectory data from the `CGaitGenerator` object. Then any knee pattern evaluation can be made by updating the `angles` matrix with knee angles and calling the `getFitness` function: For each phase, the posture is updated by using inverse kinematics to find all joint angles, then direct kinematics to find the resulting frames (i.e. all link positions and orientations). Fitness function specific code then gathers data and calculates the fitness.

`GALowerLimb` internally uses the specialized data types `vec`, `mat` and `frm` to make its calculations. `vec` is a 3×1 vector, `mat` is a 3×3 matrix and `frm` is a frame (containing a rotation matrix and a position vector). The PG program has similar data types `Vec3d` and `Mat3x3d`, but they are implemented as special cases of more general array data types and are too general and too slow to be used in the large number of calculations needed for simulating thousands or millions of robot positions during a single GA run.

6.2 Finding a fitness function

Before any optimization can be done, the optimization criterion has to be mathematically defined to reflect the desired outcome. For genetic algorithms, the criterion is the fitness function, and in this case, we want to achieve "human-like walking". However, human-likeness is not a mathematically well-defined quantity. Instead, several fitness function guesses based on assumptions about human walking were tested. Given four gait patterns, low-fitness knee patterns were generated according to each fitness function, and the function giving the most human-like patterns was chosen. The gait patterns used were

(a) **Right foot lift.** The right foot is lifted 5 cm and then put down again. This trivial pattern is expected to result in some bending of the right knee.

(b) **Both feet lift.** The right foot is lifted 5 cm and put down, followed by a left foot 5 cm lift and put down. This pattern is a simple coordination test. The symmetry of the pattern is expected to result in symmetric knee trajectories.

(c) **Six 20 cm steps.** Starting with the right foot, six 20 cm steps are taken. Note that the stride length for steps 2–5 is 40 cm, making all step lengths 20 cm. This is the gait pattern commonly used in demonstrations of WABIAN-2. Here it tests start, continuous walking and stop motions.

(d) **Six 40 cm steps.** Starting with the right foot, six 40 cm steps are taken. Like the previous pattern, stride lengths 2–5 are 80 cm. This is designed to be a more human-like gait pattern, because 20 cm steps might be too short for requiring any significant bending of the knees.

Each gait pattern starts and ends with 5 still phases to give the model time to prepare for and gracefully end the compensatory motion. Each step lasts for 20 phases, with a single midpoint at phase 10 having height $z = 5$ cm and forward displacement of half the stride length, to allow for a smooth movement. For simplicity, there are no phases of double support.

However, the GA could not be used to test the fitness functions, as its precise implementation depends on the choice of function. Therefore, another method had to be used for creating the knee patterns. Obvious candidates were alternatives to GA like random walk or simulated annealing, but these methods are random and in this case a deterministic, apprehensible method was preferred. The following "midpoint sweeping" algorithm was invented:

Starting with an empty knee pattern, a new midpoint is added to the right trajectory. The midpoint space is probed at regular intervals[3] by systematically moving the midpoint and calculating the resulting fitness value. The midpoint is then placed at the position giving the lowest fitness: the currently best position to add a midpoint. Alternating between the left and right trajectories, the process is repeated by adding midpoints until the addition of a midpoint at any position does not lower the fitness more.

This method will undoubtedly not find the optimal solution, because midpoints are added and then fixed, while their would-be optimal positions might change as more midpoints are added and the shape builds up. Also, adding midpoints alternately to the right and left trajectories might not be the optimal sequence. However, it should give a hint about the shape of solutions considered "good" by the different fitness functions.

[3]Phases $p \in \{2, 3, \ldots, n_p - 3\}$, angles $\theta \in \{0°, -1°, \ldots, -160°\}$, where n_p is the number of phases. The first and last two phases are not considered for continuity reasons.

6.2.1 Angular velocity

Based on the idea that keeping the joint angular velocities low should prevent unnecessary movements, the first fitness function candidate was one of the simplest imaginable, also serving as a test of the procedure. The fitness function was

$$F = \sum_p \sum_j \dot{\theta}_j^2(p),$$

using $\dot{\theta}_j(p) = \frac{\theta_j(p+1) - \theta_j(p)}{\Delta t}$ for simplicity.[4] Angular velocity squared ignores the direction (sign) of the angular velocity, and gives larger angular velocities bigger impact on the fitness. The summations are over all phases p and all joints j.

The resulting knee trajectories and visualizations of the legs during the simulated walking, *stick diagrams*, are shown in Figure 4, where (a) to (d) correspond to the previously defined test gait patterns. For each gait pattern, the top graphs show the left leg and the bottom graphs show the right leg. In the trajectory graphs, circles indicate midpoint locations.

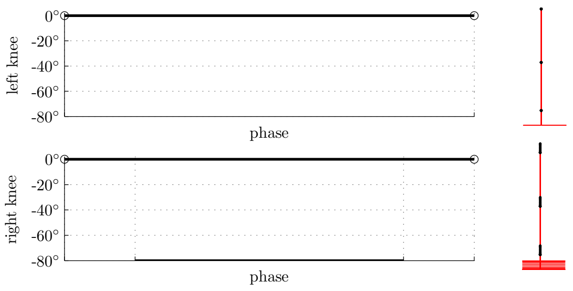
The generated patterns (a) to (c) have all knee angles identical to zero, while the (d) pattern (40 cm steps) have small angular deviations for the right knee but remains very close to zero. All patterns have stretched knees without significant bending, even for the trivial (a) pattern. The stick diagrams show that the hip positions are not smooth, resulting in a jerky walk.

6.2.2 Energy

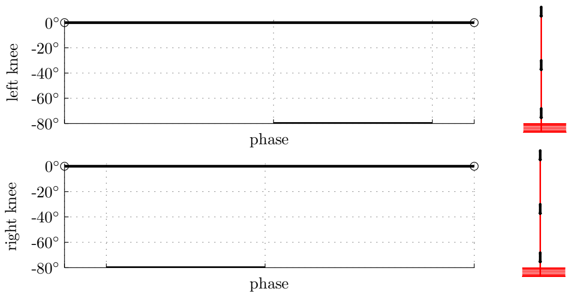
The assumption that humans walk as they do because it is energy efficient suggested that the next fitness function candidate should be a measure of energy. Since the joints are actuated by motors applying a torque proportional to the energy input, the total sum of the joint torques was used as the next fitness function.

Early experiments calculated the dynamic torque at joint j using $\boldsymbol{\tau}_j^{(d)} = \mathbf{I}_j \, \ddot{\theta}_j \, \hat{\mathbf{z}}_j$, where $\mathbf{I}_j$ is the moment of inertia tensor, appropriately translated by the parallel axis theorem, $\ddot{\theta}_j$ is the joint angular acceleration and $\hat{\mathbf{z}}_j$ is the axis of rotation. However, this model only takes motion into account, not "static" torques like those needed to maintain the posture due to gravity. A gravity compensation torque term was calculated by $\boldsymbol{\tau}_j^{(g)} = \mathbf{r}_{c_i} \times m_i(\mathbf{a}_{c_i} - \mathbf{g})$, where $\mathbf{r}_{c_i}$ is the position of the center of mass of i, m_i is the link mass, $\mathbf{a}_{c_i}$ is the acceleration of the center of mass and $\mathbf{g}$ is the gravity acceleration vector.

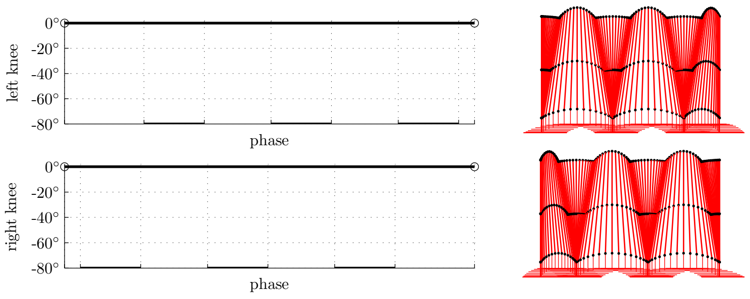
[4]Because the trajectories consist of splines, which are piecewise polynomials, the exact angular velocities and accelerations could be calculated. However useful, such a modification of the pattern generation program lies outside of the scope of this project.



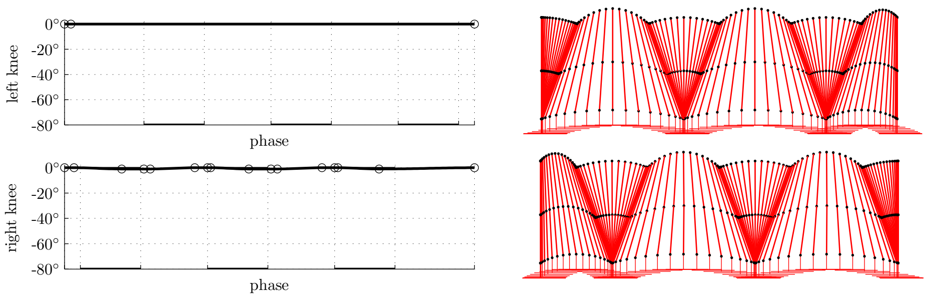
(a) Right foot lift



(b) Both feet lift



(c) Six 20 cm steps



(d) Six 40 cm steps

Figure 4 – Knee trajectories and walking patterns using angular velocity as fitness function ($F = \sum_p \sum_j \dot{\theta}_j^2(p)$)

These early models made an approximation by calculating the torque for each link individually. It was realized that the complete calculation depends on the interaction between all the links in the chain, so the Iterative Newton-Euler Dynamic formulation described in the theory section was used instead.

Because of the split link chain of the robot, the iterations have to split paths and join again. The outward iteration starts at the supporting foot, goes up the supporting leg and down the swinging leg, then an inward iteration goes up the swinging leg. After this, another outward iteration starts from the the supporting hip and goes up the upper body, and an inward goes down to the hip again. Finally, the inward iterations from the swinging leg and the upper body are combined, and a final inward iteration down the supporting leg completes the calculations.

The right foot is connected to joint 0 and the left foot is connected to joint 13. The joints 0–13 are the leg joints and 14–15 are the waist joints. The Iterative Newton-Euler dynamic equations require a joint and link preceding the first, and succeeding the last. In the implementation, therefore, there are "virtual" joints and links $i = j = -1$ for the right foot, and $i = j = 16$ for the left foot. Virtual joint and links $i = j = 17$ are also added after joint $j = 15$ to represent and end the upper body chain. The base of the robot is not rotating so we can set $\boldsymbol{\omega}_{\text{base}} = \boldsymbol{\alpha}_{\text{base}} = \mathbf{0}$. Thanks to Einstein's theory of General Relativity, we can set $\mathbf{a}_{\text{base}} = -\mathbf{g}$ to cleverly include gravity effects in the calculations without any extra effort. Finally we set $\mathbf{f}_{\text{free ends}} = \mathbf{n}_{\text{free ends}} = \mathbf{0}$ because there are no external forces or torques acting on the swinging foot or the upper body. Base in this context is the index of the virtual link preceding the supporting foot, and free ends are the two other virtual links: the swinging foot and the upper body.

The fitness function used was

$$F = \sum_p \sum_j \tau_j^2(p),$$

using τ_j from the Iterative Newton-Euler equations. Again, the square makes the values positive no matter the direction of the torque, and gives larger torques bigger impact on the fitness. The resulting trajectory plots and stick diagrams are shown in Figure 5.

In the (a) pattern, the right knee bends just the right amount to keep the hip position constant, however there is also some unexpected bending of up to about -10° of the left leg. The (b) pattern bends to the maximum angle during the swing phases but there is also some stance bending, and the trajectories are not entirely symmetric. Pattern (c) looks more like human knee trajectories, however there are some irregularities, and toward the end the right knee stretches to 0° and remains stretched through the last step.

Pattern (d) quickly bends to about $-60°$ and stays there for most of the pattern, but small variations representing the double knee action can be seen.

6.2.3 Spin Angular Momentum

Recent research in human walking suggests that humans walk not only in an energy-minimizing way, but also in a way that conserves the total angular momentum $\mathbf{L}$ about the center of mass during a walking cycle, $\frac{\partial \mathbf{L}_{\text{CM}}}{\partial t} = 0$. But the derivative of angular momentum is torque, which is already involved in the fitness function as an energy consumption measure. The researchers instead suggest minimization of spin angular momentum and the total sum of joint torque squared for generating biologically realistic leg joint movements [7].

The angular momentum is calculated as a last step of the outward iterations of the Iterative Newton-Euler calculations by

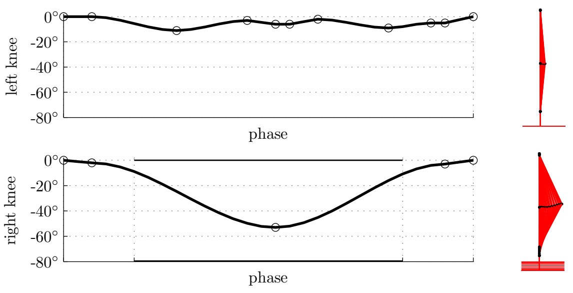
$$\begin{aligned} {}^{j}\mathbf{L}_j &= {}^{c_i}\mathbf{I}_i \, {}^{j}\boldsymbol{\omega}_j, \\ \lambda_j &= 10^3 \, |{}^{j}\mathbf{L}_j|, \end{aligned}$$

where λ_j is a scalar value for the angular momentum, comparable to τ_j for the torque. The factor 10^3 comes from the fact that the total torque values by experiment are about 10^3 times larger than the total angular momentum values, which will drown if not scaled to the same order of magnitude as the torques.

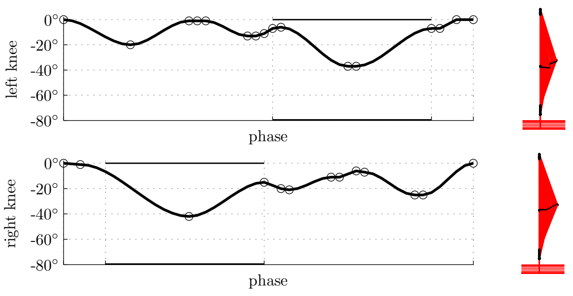
Trying to optimize using an objective function with several terms of different units (such as adding a torque and an angular momentum) can give unpredictable results. Unbeneficial changes to one term might still result in a better overall fitness value because of the other term, preventing the genetic algorithm to find good solutions. Therefore, some different forms of the fitness function combining torque and angular momentum had to be tested, to find a good cooperation between the terms.

First, the fitness function $F = \sum_p \sum_j \lambda_j^2(p)$ was used to see how the angular momentum by itself worked as a fitness function. The resulting trajectory plots and stick diagrams are shown in Figure 6.

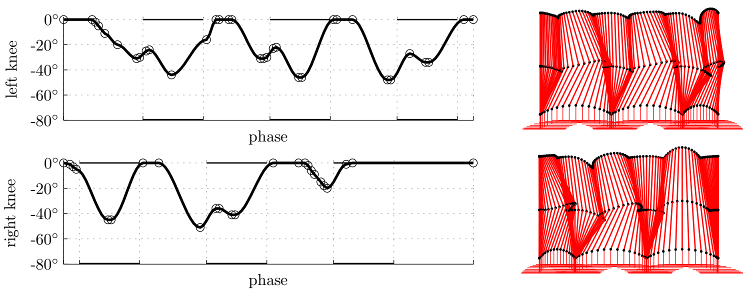
The (a) pattern correctly bends the right knee just enough not to move the hip, while keeping the left leg straight. The (b) pattern results in a symmetric pattern with adequate bending, but at the midpoint just between the lifts, both knees are bent instead of stretched. In (c), the pattern has the right attributes (traces of double knee action can be seen), but it fails to stretch the legs between the steps. Pattern (d) looks much like a human



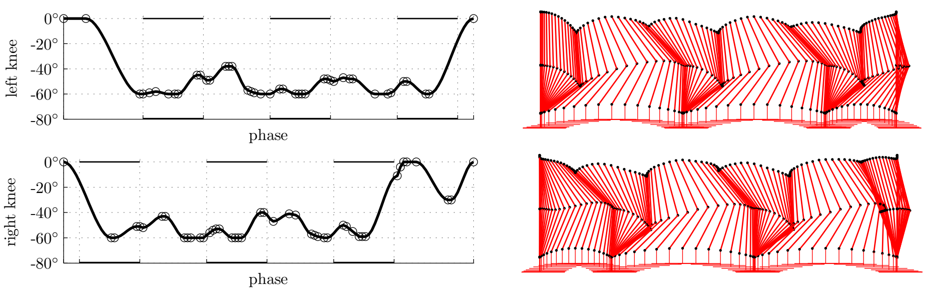
(a) Right foot lift



(b) Both feet lift



(c) Six 20 cm steps



(d) Six 40 cm steps

Figure 5 – Knee trajectories and walking patterns using torque as fitness function ($F = \sum_p \sum_j \tau_j^2(p)$)

pattern, however also fails to fully stretch the right leg during support after step 2.

In their article about spin angular momentum regulation in humans, Popovic, Hofmann and Herr successfully used the torque squared plus spin angular momentum, or $F = \sum_p \sum_j \tau_j^2(p) + \lambda_j(p)$, as an evaluation function to test their predictions on a planar (i.e. two-dimensional) biped model [7]. Inspired by this and despite the model differences, the same fitness function was tried on this WABIAN-2 simulation. The results are shown in Figure 7.

In pattern (a), the right knee bending is adequate but the left knee again has some movements. Pattern (b) bends well but again has bent knees between the foot lifts, and is not entirely symmetrical. Pattern (c) looks human yet somewhat irregular; the first left leg swing has larger bends than the rest, and the double knee action is reversed for the third right leg swing which results in the hip bump seen in the right stick diagram. In pattern (d), like in the pattern in Figure 5d, the knee quickly bends to $-60°$ with only small hints of double knee actions visible, resulting in a a jerky walk. All patterns look similar to the torque-only fitness function patterns, except that pattern (c) does not fail in the end.

Both $\sum_p \sum_j \tau_j^2(p)$ and $\sum_p \sum_j \lambda_j^2(p)$ seem to give good results, so the next fitness function tested was their sum, $F = \sum_p \sum_j \tau_j^2(p) + \lambda_j^2(p)$. The results are shown in Figure 8.

Lifting the right foot in pattern (a) again bends the right knee correctly but has some irregularities for the left. Pattern (b) has good knee bending that keeps the waist level, but are not entirely symmetric. However, the (c) pattern produces a smooth pattern having all the expected elements, such as double action and knee stretching. Pattern (d), Six 40 cm steps, also produces a good pattern despite a failure to fully stretch the left leg after the first step.

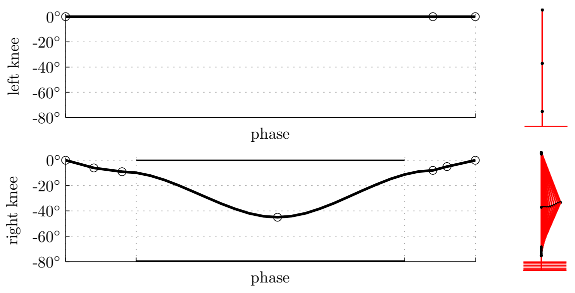
This is the only fitness function that gave good results both for (c) and (d), which are the most important patterns because they constitute walking. The small errors and irregularities were blamed on the "midpoint sweeping" algorithm, and the torque squared plus lambda squared,

$$F = \sum_p \sum_j \tau_j^2(p) + \lambda_j^2(p),$$

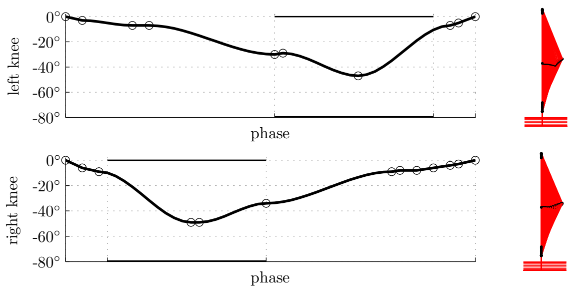
was chosen as the fitness function for the genetic algorithm.

6.2.4 Final tweaking

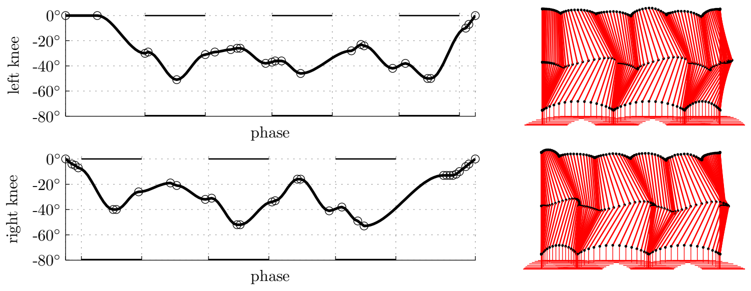
The fitness values returned for typical walking patterns turned out to be in the order of 10^6 or 10^7. Numbers this large are difficult for humans to



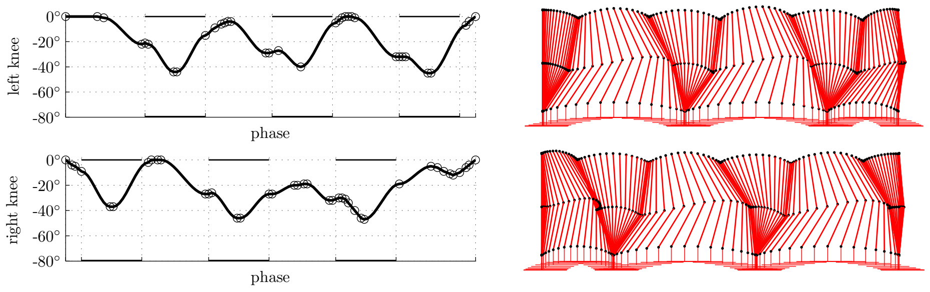
(a) Right foot lift



(b) Both feet lift

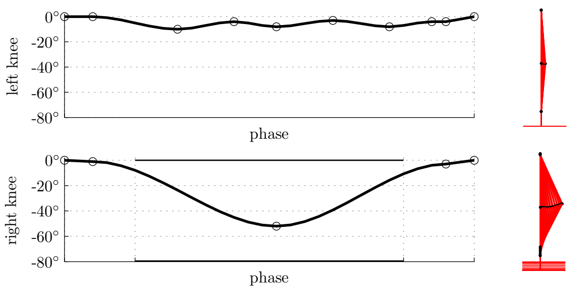


(c) Six 20 cm steps

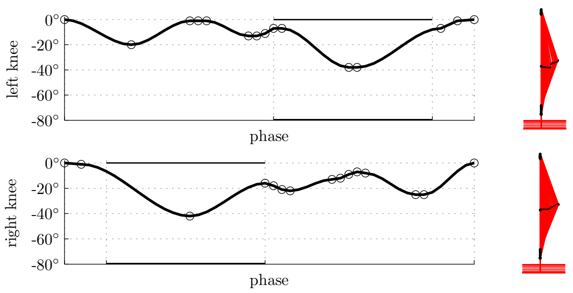


(d) Six 40 cm steps

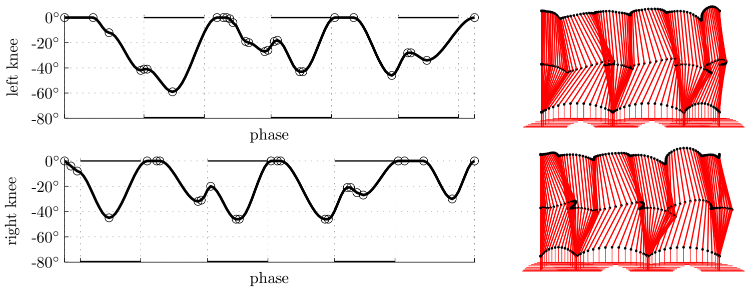
Figure 6 – Knee trajectories and walking patterns using spin angular momentum as fitness function ($F = \sum_p \sum_j \lambda_j^2(p)$)



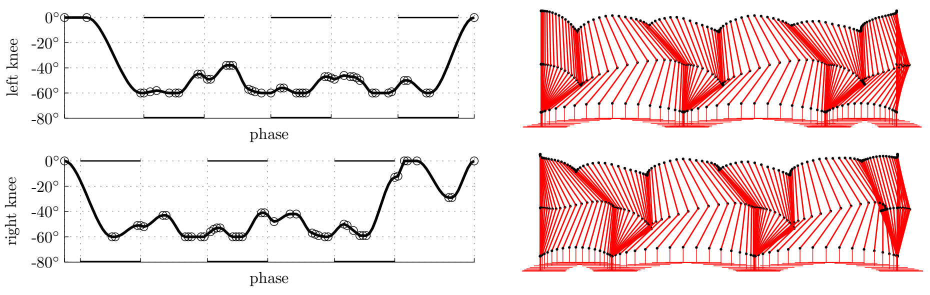
(a) Right foot lift



(b) Both feet lift

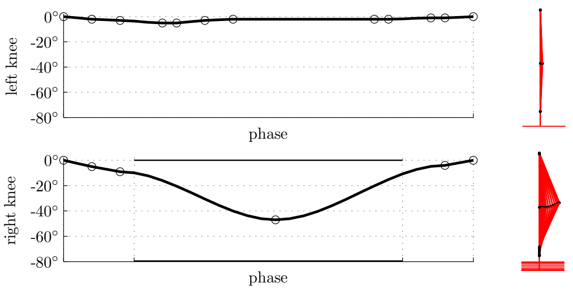


(c) Six 20 cm steps

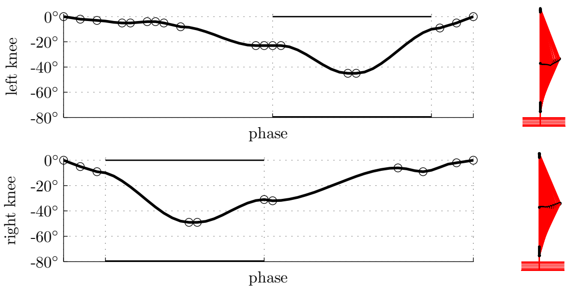


(d) Six 40 cm steps

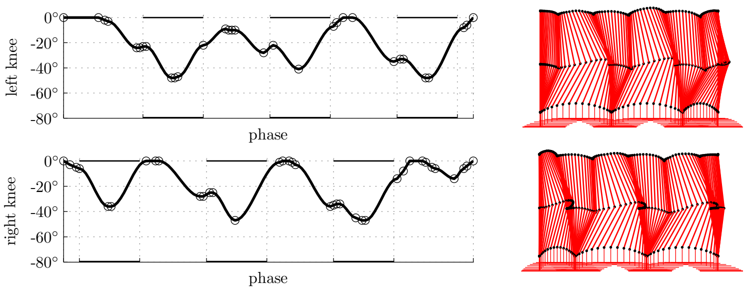
Figure 7 – Knee trajectories and walking patterns using torque squared and spin angular momentum as fitness function ($F = \sum_p \sum_j \tau_j^2(p) + \lambda_j(p)$)



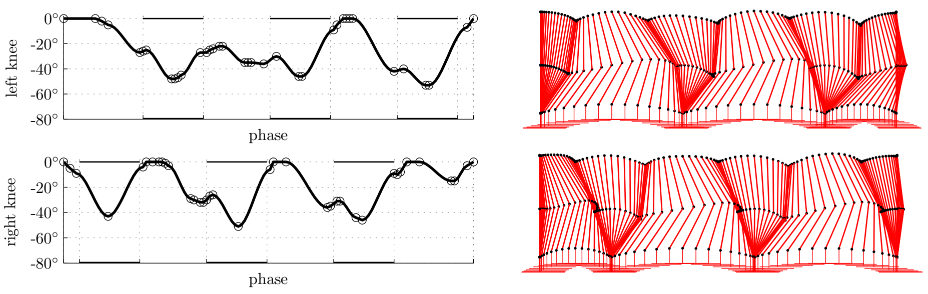
(a) Right foot lift



(b) Both feet lift



(c) Six 20 cm steps



(d) Six 40 cm steps

Figure 8 – Knee trajectories and walking patterns using torque and spin angular momentum as fitness function ($F = \sum_p \sum_j \tau_j^2(p) + \lambda_j^2(p)$)

overview and compare. Therefore, a scaling factor of 10^{-6} was applied to the fitness function.

Also, many of the trajectories in Figures 4 – 8 show excessive amounts of midpoints located close to each other. Preliminary test runs showed that the same problem exists for the GA. These formations are unwanted because a large number of midpoints have a smaller probability of all being positioned well in a certain amount of time. Often, a slight change in the position of a midpoint would lead to lower fitness, however the probability of the appropriate mutation occurring is very small and it is more likely that a midpoint is added there though crossover or mutation. In a try to counteract this and to favor patterns with fewer midpoints, a third term $10^{-3}n_{mp}$ was added to the fitness function. n_{mp} is the total number of midpoints in the right and left trajectories. The factor 10^{-3} makes the effect small, so that actual changes in midpoint setup are prioritized.

The final fitness function was

$$F = 10^{-3}n_{mp} + 10^{-6}\sum_{p}\sum_{j}\tau_j^2(p) + \lambda_j^2(p).$$

6.3 Running and fine-tuning

To find a good setup of genetic operators and parameters, standardized comparison tests were run. Gait pattern (d), Six 40 cm steps, from the Sweeping Midpoint experiments was used as the base, because it constitutes normal walking while the longer step length should require more knee bending than shorter step lengths, thus making any effects easier to see.

There is a vast amount of possible setups, some of which produce reliable patterns quickly and and others that do not. The parameters are not independent, and to find the best setup would be an non-linear optimization problem[5] in itself. Not knowing the underlying connections between the parameters, the most structured subdivision of parameter testing was chosen: To test several different settings for each parameter while keeping all other parameters constant, finding the best setting for each parameter in turn. In each test, the GA was run for 300 generations, saving all fitnesses and the best trajectory after each generation. Because the method is random, each test was run 30 times to get representative average values. The setting with the best evolution was used in all subsequent experiments, gradually improving the GA setup. In the event that several different settings were equally good, the one requiring the least amount of computer power was chosen in order to keep the GA as fast as possible.

[5]Perhaps a meta-GA could be used to solve this problem.

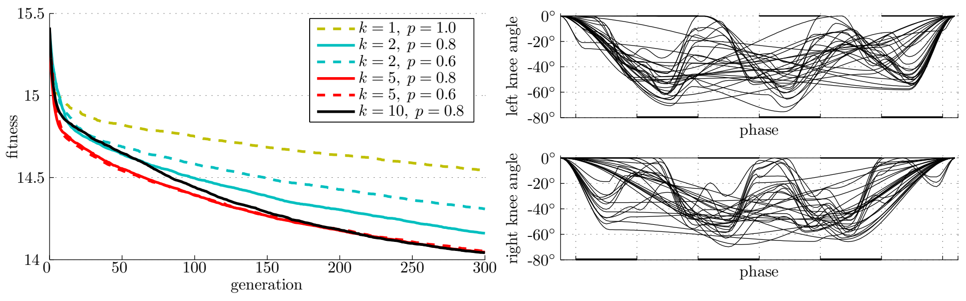


Figure 9 – Selection operators – best is tournament selection ($k = 5,\ p = 0.8$)

The experiment results were visualized by a generation plot, showing the average fitness of the best individuals for each generation, for each setting of the parameters, and a plot showing all 30 trajectories generated from the winning setting to visualize the coherence of the results. Optimally, the GA should reach the same result each time and the trajectories should all be equal.

6.3.1 The selection operator

Because of the many drawbacks of roulette-wheel selection, only tournament selection was used. Different combinations of the parameters tournament size k and selection probability p were tested. The other (constant) parameter settings were the somewhat arbitrary initial setup, inspired by the SGA: Children survival without differentiation, creeping mutation without add/remove, one point crossover without swap, population size of 20 individuals, crossover probability of 60%, mutation probability of 1%, and the inversion operator enabled.

The settings tested were ($k = 2,\ p = 0.8$), ($k = 2,\ p = 0.6$), ($k = 5,\ p = 0.8$), ($k = 5,\ p = 0.6$) and ($k = 10,\ p = 0.8$). For comparison, ($k = 1,\ p = 1.0$), equivalent to *random selection*, is also included. The results are shown in Figure 9.

The random selection performs worst, while $k = 5$ outperforms $k = 2$. $k = 10$ eventually reaches the same low fitness, but converges slower. ($k = 5,\ p = 0.8$) and ($k = 5,\ p = 0.6$) are very similar, but with $p = 0.8$ there is a larger chance of getting faster results. The winning setup is thus ($k = 5,\ p = 0.8$), and the trajectories generated using this setting are also shown in Figure 9. Some trajectories have the correct form but a larger number does not.

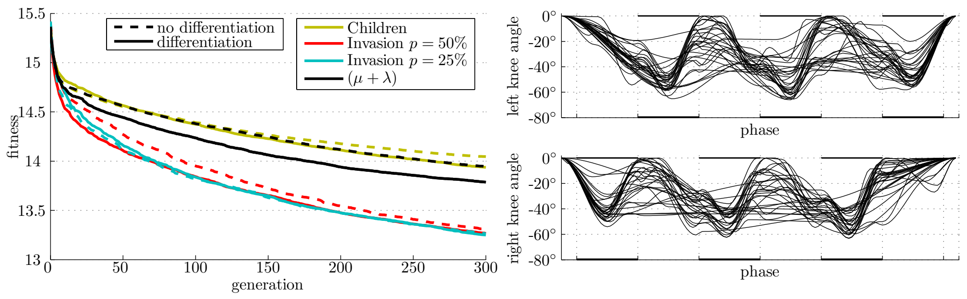


Figure 10 – Survival operators – best is invasion $p = 50\%$ with differentiation

6.3.2 The survival operator

The survival operators Children, Invasion $p = 50\%$, Invasion $p = 25\%$, and $(\mu+\lambda)$ were all tested with and without the differentiation operator, designed to keep diversity high. The other parameter settings were: Tournament selection ($k = 5$, $p = 0.8$), creeping mutation without add/remove, one point crossover without swap, population size of 20 individuals, crossover probability of 60%, mutation probability of 1%, and the inversion operator enabled. The results are shown in Figure 10.

Children selection performs worst, followed by $(\mu+\lambda)$. Invasion $p = 50\%$ and Invasion $p = 25\%$ perform about the same. Generally, using differentiation is better than not using it. Invasion $p = 50\%$ with differentiation converges slightly faster than $p = 25\%$, and the larger child survival percentage p means fewer invading, new individuals, and it thus requires less computing power. Invasion $p = 50\%$ with differentiation was chosen as the survival operator. Trajectories generated using this survival operator are shown in Figure 10. A slightly larger number of trajectories have the right shape, but many deviate largely from the expected.

6.3.3 The mutation operator

The Random, Creeping and Block mutation operators were tested in combination with and without the ability to randomly add or remove midpoints. The other parameter settings were: Tournament selection ($k = 5$, $p = 0.8$), invasion $p = 50\%$ survival with differentiation, one point crossover without swap, population size of 20 individuals, crossover probability of 60%, mutation probability of 1%, and the inversion operator enabled. The results are shown in Figure 11.

Block mutation performs the worst, followed by creeping mutation. Random mutation unexpectedly performs well. For block and creeping mutation,

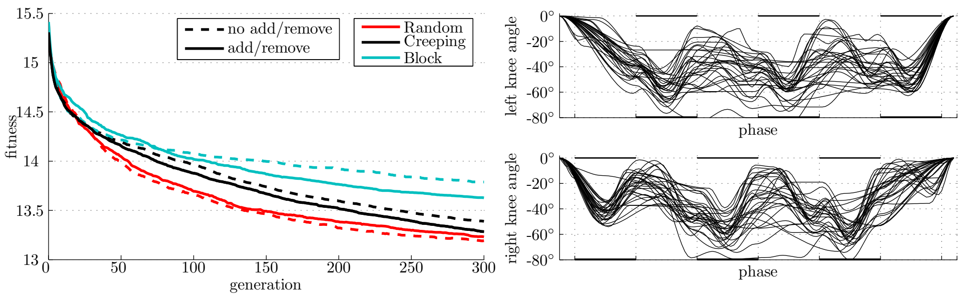


Figure 11 – Mutation operators – best is random without add/remove

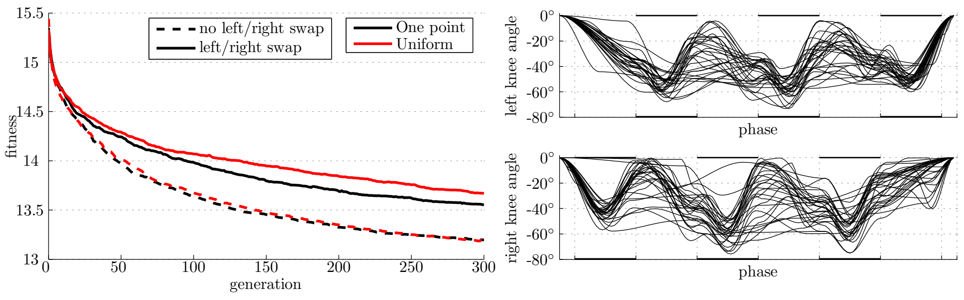


Figure 12 – Crossover operators – best is uniform without swap

using add/remove is better, while for random mutation, it is slightly better not to use add/remove. The best mutation operator is thus random mutation without add/remove. Trajectories from using this mutation operator are shown in Figure 11. Not many of the 30 trajectories have the desired shape, and the spread seems to have increased. For example, none of the trajectories fully stretch legs between steps.

6.3.4 The crossover operator

The one point and the uniform crossover operators were tested in combination with and without the ability to randomly swap material between the left and right trajectories. The other parameter settings were: Tournament selection ($k = 5,\ p = 0.8$), invasion $p = 50\%$ survival with differentiation, random mutation without add/remove, population size of 20 individuals, crossover probability of 60%, mutation probability of 1%, and the inversion operator enabled. The results are shown in Figure 12.

Using left/right swap performs significantly worse than not using it, with one point crossover slightly better than uniform. However, when not using swap, one point and uniform crossover are very similar, but uniform crossover

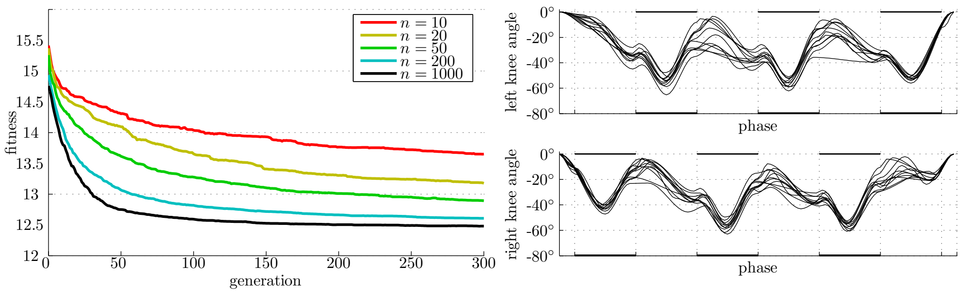


Figure 13 – Population size – best is $n = 1000$

is slightly faster and is conceptually simpler. Therefore, uniform crossover without swap was chosen, and the trajectories generated from it are shown in Figure 12. Most trajectories have the right overall shape, and almost all trajectories correctly bend around midswing, but some fail to stretch during early stance.

6.3.5 Population size

Population sizes of 10, 20, 50, 200, and 1000 individuals were tested. This experiment was only run 10 times because of its extreme time consumption, yet the results seem conclusive. The other parameter settings were: Tournament selection ($k = 5$, $p = 0.8$), invasion $p = 50\%$ survival with differentiation, random mutation without add/remove, uniform crossover without swap, crossover probability of 60%, mutation probability of 1%, and the inversion operator enabled. The results are shown in Figure 13.

It is clear that for larger population sizes, the convergence is faster and the fitness reaches lower. The best results were acquired with $n = 1000$, but the time consumption (a full weekend for just 10 runs) makes this setting practically impossible. The population size for subsequent experiments was left at $n = 20$, to allow for comparison with the previous experiments. The trajectories generated using the best setting $n = 1000$ are shown in Figure 13. All trajectories correctly follow the same shape, with only small deviations. However, about half the trajectories still fail to stretch the left leg after the first step.

6.3.6 Mutation and Crossover probabilities

The mutation and crossover probabilities are related, and are therefore tested at the same time. All combinations of crossover probabilities 30%, 60% and 90% with mutation probabilities 0.1%, 1% and 10% were tested. The other

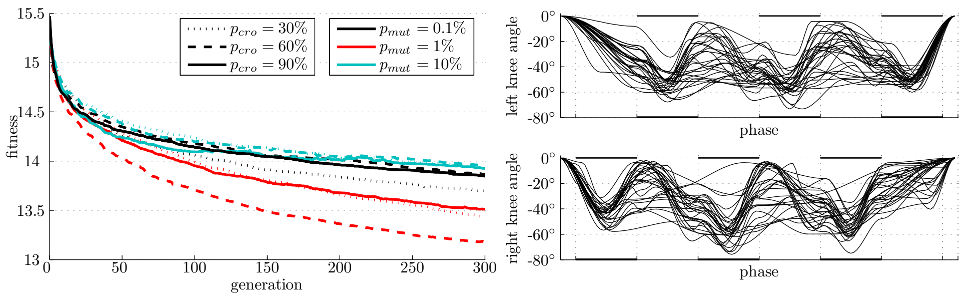


Figure 14 – Mutation and Crossover probabilities – best is $p_{mut} = 1\%,\ p_{cro} = 60\%$

parameter settings were: Tournament selection ($k = 5,\ p = 0.8$), invasion $p = 50\%$ survival with differentiation, random mutation without add/remove, uniform crossover without swap, population size of 20 individuals, and the inversion operator enabled. The results are shown in Figure 14.

Mutation probabilities of $p_{mut} = 10\%$ perform the worst, closely followed by $p_{mut} = 0.1\%$, but $p_{mut} = 0.1\%$ with a crossover probability of $p_{cro} = 30\%$ performs slightly better. Using $p_{mut} = 1\%$ with $p_{cro} = 30\%$ or 90% berform slightly better still, but $p_{mut} = 1\%,\ p_{cro} = 60\%$ is significantly better, and was chosen for subsequent experiments. The trajectories generated using these probabilities are shown in Figure 14. Again, they seem more spread out than Figure 12, for example. Some trajectories miss some bends or stretches entirely, but most follow the overall shape.

6.3.7 The inversion operator

The GA was tested with inversion on and off. The purpose of the inversion operator is to rearrange the chromosomes to form short-order schemata, but this is hardly meaningful with uniform crossover and random mutation. It is expected that inversion will have no effect. The other parameter settings were: Tournament selection ($k = 5,\ p = 0.8$), invasion $p = 50\%$ survival with differentiation, random mutation without add/remove, uniform crossover without swap, population size of 20 individuals, crossover probability of 60%, and mutation probability of 1%. The results are shown in Figure 15.

As expected, inversion does not have any effect on the fitness evolution. Since using inversion is more computationally expensive than not using it, inversion is turned off. Trajectories generated with inversion turned off are shown in Figure 15. As before, they follow the same general pattern but have some deviations, with failure to stretch slightly more prevalent than failure

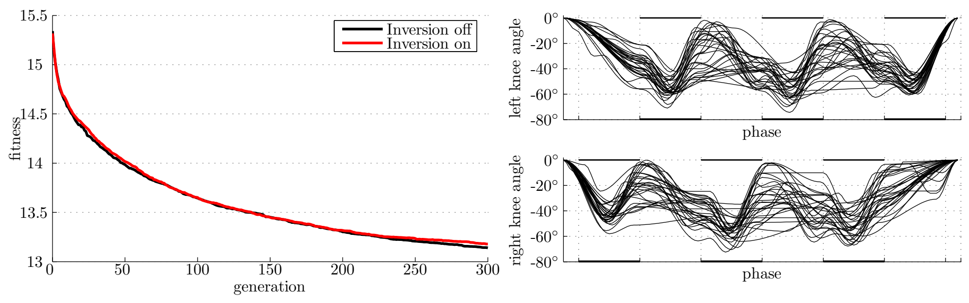


Figure 15 – Inversion operators – best is inversion off

to bend.

6.3.8 Individuals vs. Generations

A population size of $n_i = 1\ 000$ individuals[6] was clearly the setting giving the largest impact on the evolution, and producing the best results. Over $n_g = 300$ generations, a total of 300 000 individuals were in existence, and an even larger number of fitness evaluations were made because of rejected individuals and the 50% invasion. By contrast, using a population size of $n_i = 20$ merely considered a total of 6 000 individuals, and the comparison seems unfair. To give them equal chance, ($n_i = 1\ 000,\ n_g = 300$) was compared with ($n_i = 20,\ n_g = 15\ 000$), i.e. using the same total number of tested individuals. The other parameter settings were: Tournament selection ($k = 5,\ p = 0.8$), invasion $p = 50\%$ survival with differentiation, random mutation without add/remove, uniform crossover without swap, crossover probability of 60%, mutation probability of 1%, and inversion turned off. The results are shown in Figure 16, with number of tested individuals instead of generations on the horizontal axis.

($n_i = 1\ 000,\ n_g = 300$) reaches a lower final fitness, while ($n_i = 20,\ n_g = 15\ 000$) initially converges faster. The trajectories generated using the larger number of individuals are shown in Figure 16. These are the same trajectories as shown in the population size experiment in Figure 13. For comparison, the trajectories generated using the larger number of generations are shown in Figure 17. These trajectories deviate more from the expected knee trajectory pattern, and are not much better than trajectories from previous experiments using $n_i = 20, n_g = 300$. The conclusion is that larger population sizes produce better results.

[6]An index i is added here to tell the number of individuals n_i apart from number of generations n_g.

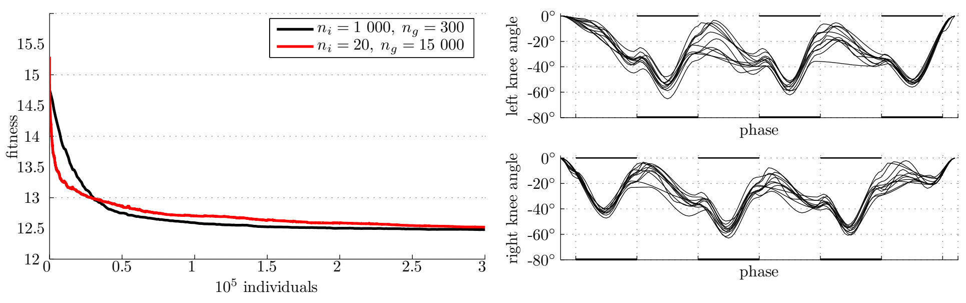


Figure 16 – Individuals vs. generations – best is ($n_i = 1\ 000,\ n_g = 300$)

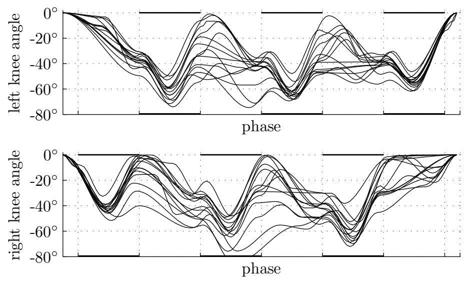


Figure 17 – The trajectories generated using ($n_i = 20,\ n_g = 15\ 000$)

6.4 Real-world application

It is said in some groups that "if you talked the talk, you gotta' walk the walk". The genetic algorithm had been constructed and fine-tuned to produce as good patterns as possible, and it was time to test the results on WABIAN-2.

The gait pattern normally used in demonstrations of the robot has 20 cm steps. To compare a GA pattern with a reference pattern, a six 20 cm steps pattern similar to the (c) gait pattern from the Midpoint Sweeping experiments was used. Because the pattern was going to be really used and not just simulated, 128 still phases was inserted in the beginning and the end of the pattern to allow for the necessary waist and trunk compensatory motions. The number of phases for each step was changed from 20 to 32 to allow for better timing, and double support was introduced as the first two phases of every swing interval because instantaneous foot switching is not physically possible during walking. Thus, the final pattern length was 448 phases, or 13.44 seconds. The corresponding knee pattern, used in demonstrations with the described gait pattern, is shown in Figure 18.

However, the GA does not produce reliable, clean results. Not a single trajectory generated in the previous experiments is without noise or deviations. Therefore, a knee pattern *inspired* by the results was manually created. Un-

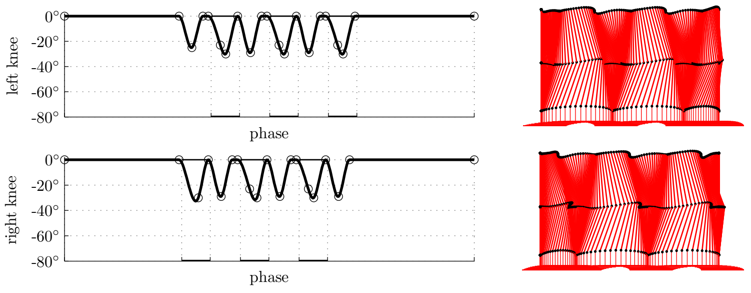


Figure 18 – The previous knee pattern, used in demonstrations of the robot

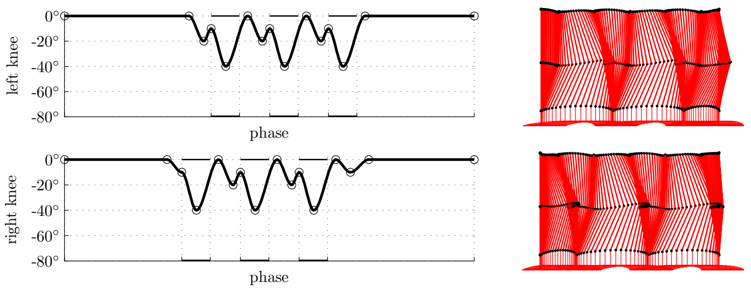


Figure 19 – The new pattern, based on GA experience and output

fortunately, the waist roll angle has a mechanical limit of about $\pm 10°$. While the original GA-inspired knee pattern resulted in small waist roll angles only, the waist compensatory motion needed to keep balance resulted in waist roll angles of up to $\pm 15°$, which is an invalid robot posture. The knee pattern was manually modified to reduce this effect, mainly by decreasing the maximum knee bend angle by 20° (from $-60°$ to $-40°$) and the smaller initial double action bend angle by 20° (from $-40°$ to $-20°$). The final GA-inspired knee pattern is shown in Figure 19.

The robot was programmed with the two walking patterns, and the walking experiments were performed three times each. Movies of the walking was recorded for further study, and sensor data from the robot was collected. Snapshots from the movie was taken at about 0.3 second intervals, shown in Figures 20 and 21. The 20 frames in a Figure should be read left to right, top to bottom. The differences between the patterns are very small.

Close inspection of the old pattern frames in Figure 20 show that when the swing leg bends to take a step, the stance leg also slightly bends with it, creating a somewhat bouncy walk. This is most clearly visible in frames 14–15–16, where the right leg takes a step but the left leg also bends visibly. This effect can also be seen in the knee pattern in Figure 18. The new pattern frames in Figure 21 show a smoother walking style, almost as if the robot is sneaking. The waist roll movements are larger than for the old pattern, however this is hard to see in the frames but easily visible watching the movie. The human-likeness of the new pattern is very hard to judge. The walk still looks artificial, but the somewhat bouncy walking style of the original pattern has been smoothed out.

The robot sensors reported a lower limb energy expenditure of 20.4 kJ using the old knee pattern, and 14.1 kJ for the GA-inspired, which is a saving of 6.3 kJ, or 31%. The energy expenditure for the whole robot was 57.2 kJ for the old pattern and 50.2 kJ for the GA-inspired, a total saving of 12%. Thus, the new pattern is much more energy efficient.

7 Discussion

The task of optimizing is controversial when the optimization goal is not clear. There is no obvious definition as to what is human-like, considering that most people have gaits so different that you can tell people apart from the movements alone. Even if you do choose a “standard gait”, it is very difficult to describe the gait mathematically, and even harder to find a function that scores different gaits such that good gaits get the best scores after passing through the GA. In this project, the fitness function was constructed by making guesses based on assumptions on human walking. However, the experiment results are not consistent, and the most probable error is the fitness function. It would be a difficult and time consuming task to find better fitness functions.

Also, the robot model is only somewhat human-like. The links have approximately human proportions, and the robot has the same basic degrees of freedom as a human. But humans use contractive actuators (muscles) and the robot uses rotational actuators (motors). The contact between the lower limbs and the physical world is largely though the feet, and humans have soft feet with several degrees of freedom, using the toes, heels and foot pitch motions to make a smooth walk. The robot, by contrast, has solid feet that normally keeps parallel to the ground. This makes any pattern look slightly unhuman, but more importantly it might also affect the energy consumption so that an optimal, energy efficient pattern still does not resemble human walking.

There is also the problem of genetic algorithms not easily reaching optimal solutions. Usually, GAs are good at quickly finding an area close to a global optimum, but slow at actually converging to the optimum. This is because a GA is not as “geographically aware” in the fitness landscape as some other

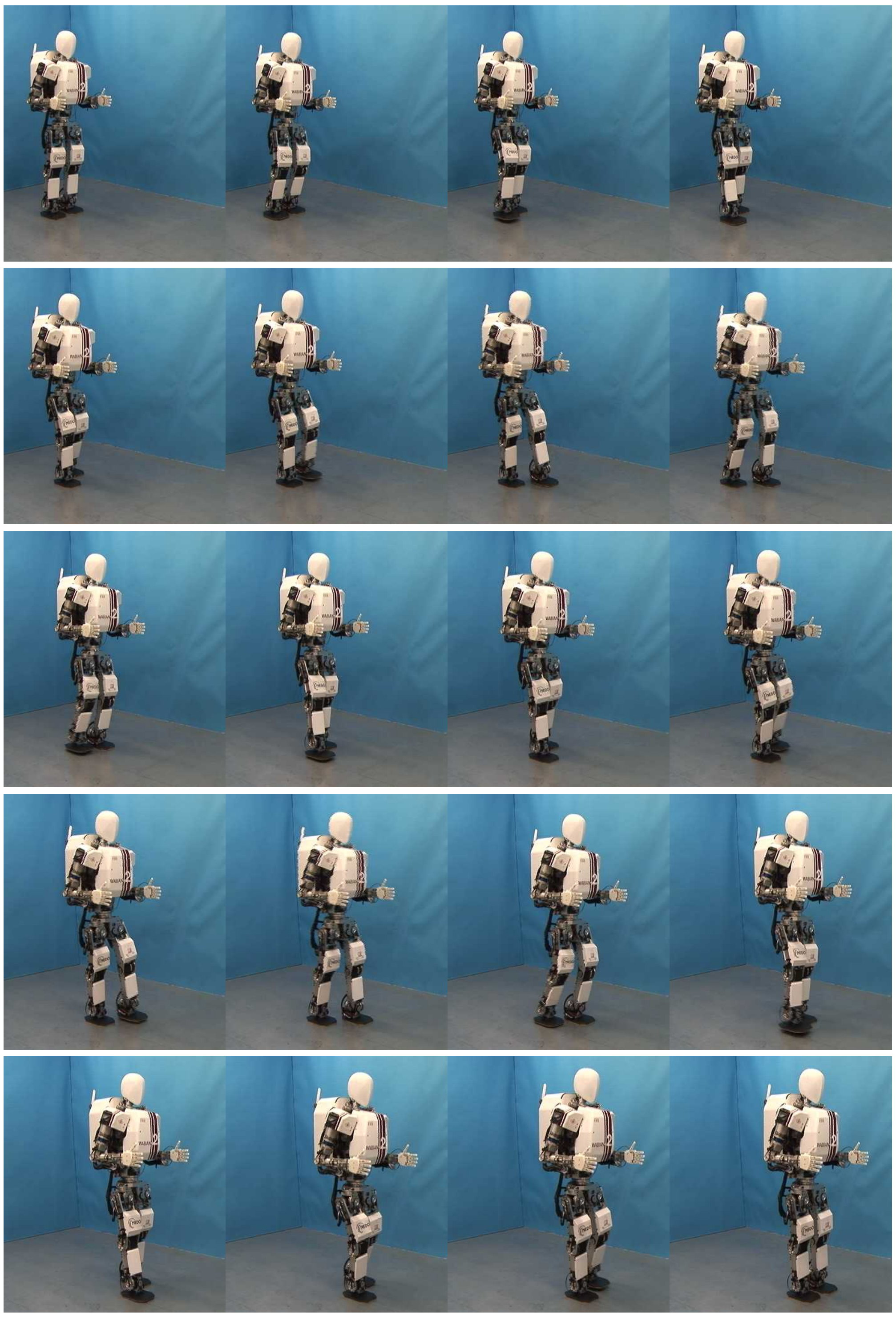


Figure 20 – Snapshots of the video captured when using old knee pattern

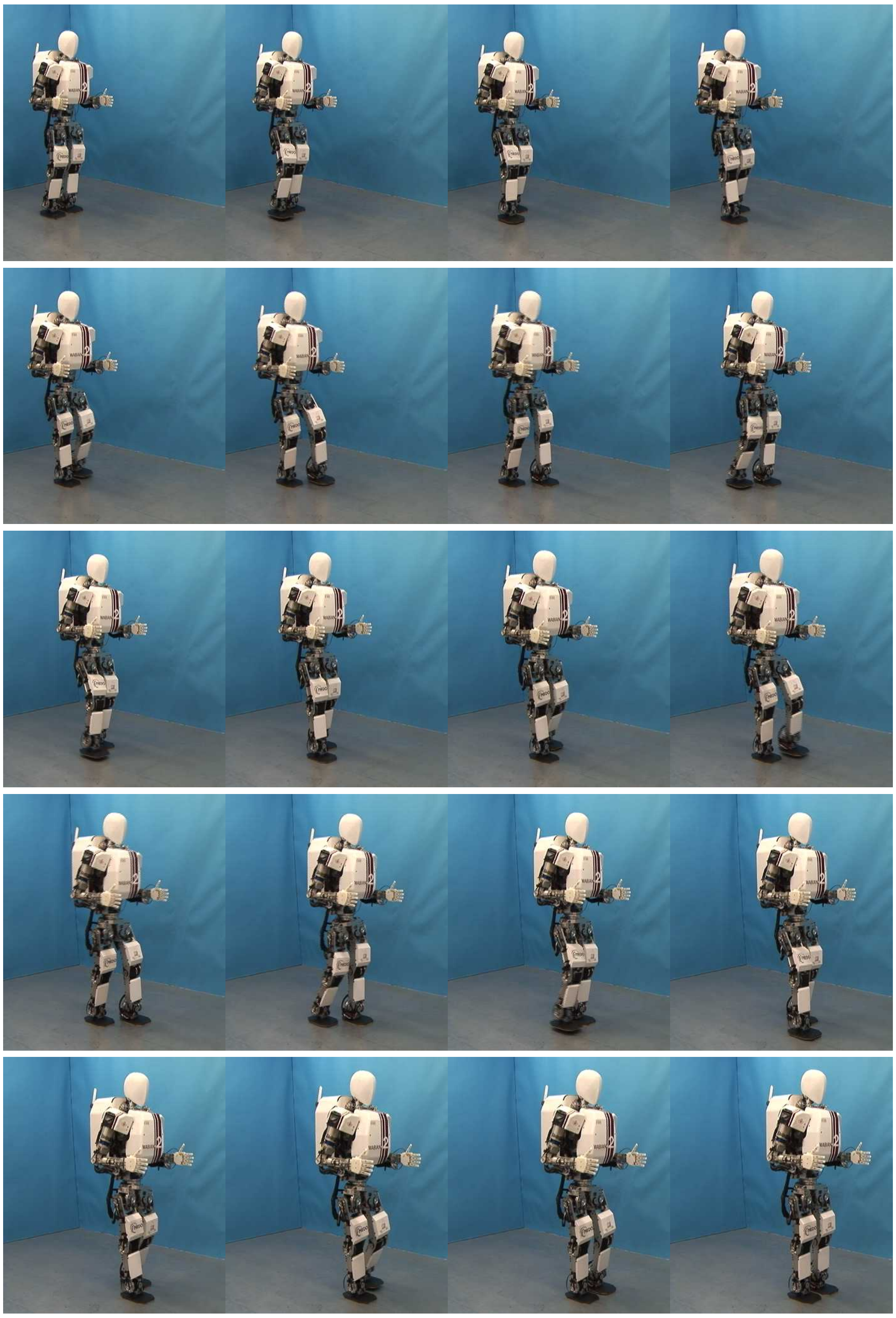


Figure 21 – Snapshots of the video captured when using new knee pattern

optimization methods, such as gradient descent. Usually, a better way to solve an optimization problem is to first use a GA to find the general area of the solution, avoiding local optima, and then use another method that more quickly converges to the global optimum. However useful, there was not enough time to perform such experiments within this project.

The fact that the compensatory calculation changes the walking pattern also impacts the effectiveness of the method. Having to manually modify the GA patterns to be able to use them on the robot is not practical. These compensatory movements might also change the torques and spin angular momenta of the robot, counteracting the GA results. The solution would be to include the compensatory calculation in the fitness simulation of the robot, but this is not possible for two reasons. The workflow of the PG program is to set the ZMP trajectory after the knee trajectory, actively hindering an automated process. This could easily be changed by a slight redesign of the program. But more importantly, the compensatory motion calculation is extremely slow, usually taking 10 seconds or more for a single pattern. 300 generations with a population size of 20 individuals typically took 5 minutes to complete; it would take an extra 17 hours to include the compensation. Including compensation in the longer experiment with $n_i = 1\ 000, n_g = 300$ would put an additional 5 weeks to the 4 hour single GA run time[7].

The results from the Sweeping Midpoint experiments show that approximate trajectories can be constructed using this simple model. As pointed out, the method did not produce optimal trajectories, because the added midpoints were fixed while their would-be optimal positions might shift when adding more midpoints. However, the trajectories obtained were similar to the resulting GA trajectories of later experiments, and also similar to the human knee trajectories. A possible improvement of this method might be to also use a “shake” function, repeatedly randomly moving all midpoints slightly but only keeping the changes that lead to better fitness. This is analogous to shaking a box full of oranges to make them occupy the smallest volume possible, and might shift the midpoints to their would-be best positions after adding more midpoints. If this method works, it would almost certainly be faster than a genetic algorithm.

The reason that most of the trajectories generated in the GA experiments deviate from the expected knee trajectory seems to be that there are too few individuals in the populations. The experiments with $n = 1\ 000$ gave much better trajectories, but still had variations and even errors in them. Possibly, using large population sizes *and* larger number of generations might make the

[7]A 3GHz dual CPU computer was used; the program is single threaded and can only use one CPU, but the system tasks were running on the other and thus not interfering.

trajectories converge. However, the time consumption required makes this approach impractical. Another improvement approach might be to find and use a fitness function that more clearly favors only really good trajectories.

It is interesting that the GA turned out to work best using uniform crossover and random mutation. These operators resemble the random walk optimization method more than genetic algorithms. Uniform crossover randomly distributes the genes of the parents to the children, and random mutation randomly replaces some genes. Although debated, the strengths of genetic algorithms is said to be the combination of a crossover and mutation operator, crossover providing possibility of combining the good properties of both parents into a child, and mutation randomly introducing new better material or reintroducing genes that have previously been lost. Goldberg and Mitchell argue that some problems are not difficult enough for a GA. Genetic algorithms uses short-order schemata to construct good solutions, so if there are no meaningful short-order schemata for the problem, a GA does not perform better than a random method [5][6]. If this is the case here, changing the data representation or fitness function might make the GA take more advantage of the GA operators.

The data representation of using midpoints was chosen to be flexible, however other representations might give better results. For example, isolating and optimizing a single step (a stance and a swing phase) and then repeating the result to create the full pattern would give more symmetric and regular patterns, however the method would be restricted to patterns with only one type of step. This would present problems during start and stop motions. Another possibility is to use frequency and amplitude information as genes. Knee trajectories can be seen as periodic functions having relatively few defining attributes, and Fourier transforms might be used to convert between the time domain knee pattern for simulations and the frequency domain genome. Again the problem is start and stop motions, i.e. restricting the pattern to standstill in the ends.

There are also many types of genetic operators that would be interesting to try on this project. Using variable probabilities for allowing big changes early and only smaller later might improve performance. The Boltzmann tournament selection operator, for example, has a global temperature parameter that is constantly decreasing. The temperature determines the probability p in the tournament selection, thus more readily choosing suboptimal individuals early, when diversity is needed, and choosing more optimal individuals later, when faster convergence is needed to find the solution. There are also a number of interesting crossover operators, like arithmetic crossover where the genes are not distributed but arithmetically combined (e.g. added) to produce children that are the averages of the parents in some sense, or multisexual crossover in which more than two parents share their genes to produce children. Two point or n-point crossover are also interesting alternatives to one point and uniform crossover. However, if the problem does not take advantage of the genetic operators used, these operators might also be of no use, and the best setup might still be closer to a Random Walk.

8 Conclusion

This project was an attempt to generate optimal knee trajectories for the humanoid robot WABIAN-2.

As a first step, the data representation was chosen and the base classes were constructed to define the workings of the GA. A generation keeps the individuals, and the individuals each store a left and a right knee trajectory. A flexible data representation uses midpoints to define key angles in the trajectories, and the full knee angle sets can be created by cubic spline interpolation when needed. A specialized version of the existing kinematics class was created for calculating the fitness of an individual through simulation of the robot walk.

Next, a fitness function giving lower fitnesses to more human-like knee trajectories was constructed. A midpoint sweeping algorithm was invented to evaluate fitness functions using four specially designed gait patterns. Fitness functions based on joint angular velocities and different combinations of torques and spin angular momenta were tested. The best fitness function was $F = 10^{-3} n_{mp} + 10^{-6} \sum_p \sum_j \tau_j^2(p) + \lambda_j^2(p)$, which was the only function generating acceptable patterns both for 20 cm step length and for 40 cm step length walking.

Many different GA parameters and operators were tested to fine-tune the GA performance. Each parameter was tested while keeping the others constant, and the best choice of parameters was

- tournament selection with size $k = 5$ and probability $p = 80\%$,
- invasion survival with child survival $p = 50\%$ using differentiation,
- random mutation without random add/remove of midpoints,
- uniform crossover without left/right trajectory swap,
- population size of $n = 1\ 000$,
- crossover probability of $p_{cro} = 60\%$, mutation probability of $p_{mut} = 1\%$,
- inversion operator turned off.

It was also tested and concluded that having a large population size but few generations was more beneficial than having a small population size but many generations, even though both methods tested the same number of individuals.

Finally, a pattern inspired by the GA results was tested on the robot and compared with a pattern previously used in demonstrations. The new walking style was smoother but still looked artificial. The robot reported using 31% less energy for the lower limbs with the new pattern.

It seems that the method works fairly well, even though the final setup more resembles a random walk through the fitness landscape than a GA. Some approximations have been made, and many improvements are possible. For example, a more specialized fitness function or another data representation might be better suited to create short-order schemata and thus take better advantage of the properties of genetic algorithms.

References

[1] Ed Ayyappa. Normal human locomotion, part 1: Basic concepts and terminology. *Journal of Prosthetics & Orthotics*, 9(1), 1997.

[2] Guy Bessonet, Stéphane Chessé, and Philippe Sardain. Optimal gait synthesis of a seven-link planar biped. *The International Journal of Robotics Research*, 23(10–11):1059–73, October–November 2004.

[3] G. Capi, Y. Nasu, L. Barolli, M. Yamano, K. Mitobe, and K. Takeda. A neural network implementation of biped robot optimal gait during walking generated by genetic algorithm. In *9th Mediterranean Conference on Control and Automation*, June 2001.

[4] John J. Craig. *Introduction to robotics: mechanics and control*. Pearson Prentice Hall, third edition, 2005.

[5] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[6] Melanie Mitchell. *An introduction to genetic algorithms*. The MIT Press, 1998.

[7] Marko Popovic, Andreas Hofmann, and Hugh Herr. Angular momentum regulation during human walking: Biomechanics and control. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pages 2405–11, April 2004.

Bibliography

- Ed Ayyappa. Normal human locomotion, part 1: Basic concepts and terminology. *Journal of Prosthetics & Orthotics*, 9(1), 1997.
- Guy Bessonet, Stéphane Chessé, and Philippe Sardain. Optimal gait synthesis of a seven-link planar biped. *The International Journal of Robotics Research*, 23(10–11):1059–73, October–November 2004.
- G. Capi, Y.Nasu, L. Barolli, M. Yamano, K. Mitobe, and K. Takeda. A neural network implementation of biped robot optimal gait during walking generated by genetic algorithm. In *9th Mediterranean Conference on Control and Automation*, June 2001.
- John J. Craig, *Introduction to Robotics: mechanics and control*. Pearson Prentice Hall, third edition, 2005.

- Clive Davidson. Creatures from primordial silicon. *New Scientist*, 156(2108):30–35, November 1997.

- David E. Goldberg. *Genetic Algorithms in search, optimization, and machine learning.* Addison-Wesley, 1989.

- Melanie Mitchell. *An introduction to genetic algorithms.* The MIT Press, 1998.

- Marko Popovic, Andreas Hofmann, and Hugh Herr. Angular momentum regulation during human walking: Biomechanics and control. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pages 2405–11, April 2004.