Contents

1 Control Architecture **1**
1.1 Dynamical Systems 2
1.2 Control Basics 5
1.2.1 Proportional-Integral-Derivative Controller 7
1.2.2 Non-linear extensions to PID control 9
1.3 Cognitive Sensorimotor Loops 10
1.4 Advanced Non-linear Controllers for Multi-Joint Robots 17
1.4.1 Sensor and Motor Space Definition 18
1.4.2 Generalized Non-linear Controller 20
1.4.3 A simple bipedal gait pattern 25
1.4.4 Reducing the number of weights 27

Bibliography **31**

globaler arbeitsschritt: alle füllwörter rausballern

1 Control Architecture

Legged robots should be able to teach themselves to run. We should enable them to acquire adequate motions for various situations *by learning* ~~and~~ let them make their own decisions and experiences. This is because, even for robots with simple morphologies and only a few degrees of freedom, it ~~could~~ is already ~~be~~ challenging to manually create motions and gaits which are robust against environmental disturbances. ~~And usually,~~ we want these motions to be energy efficient and material-friendly at the same time, ~~Also~~, real robots' hardware is subject to wear and the controller ~~probably~~ needs to be adapted to subtle changes in hardware over time. ~~Most crucial,~~ how could we foresee every possible situation the robot could be faced with?

based on letting them

, making things more difficult.

hier den kategorischen unterschied betonen.

The control structure we are searching for should be mostly independent of the specific morphology, in order to be applicable to various robots with a different number of joints. Ideally, this structure would require only a few assumptions on the robot's hardware and should especially not depend on how the robot's joints are composed, where sensors and motors are located and how they influence each other.

das zweite bec ause ist der eig entliche killer

Instead of defining and modeling every detail, we should take the real world as she is—a beautiful mess of structure—and we should rather define a parametri~~zable~~ control scheme for our robots, for which we could find various distinct behaviors—but whose parameters could also be changed easily with common optimization techniques. Those behaviors should be non-trivial and should cover a large part of the robot's behavioral space. For instance, walking and running, forwards and in different directions, turning on the spot, standing stable, recovering balance and so forth. But, on the other hand, the controller must be transparent enough for later analysis of the parameters after optimization.

:) it is ok

c

versteh ich nicht ganz

?

A random search in parameter space is in general able to find useful parameters for arbitrary control structures given a suitable cost function. However, it is not guaranteed that the global minimum of the cost function will be found at all if the search space is large and has a lot of local optima. Reducing the occurrences of local optima helps to focus on finding the globally optimal solution since it allows us to use gradient information. Also, for learning, it is highly beneficial if similar parameters create similar behaviors, so that optimization algorithms can increase the search effort close to regions of the search space where the good performance was already proven. There is always the possibility to influence the design of the control structure in an optimizer-friendly way, in order to get cost functions with fewer local optima. We should be aware that non-linearities in the control structure may lead to unwanted local optima and that hidden states may bring additional challenges for the optimization since the controller's output probably has different outcomes given the same inputs, depending on the recent history of inputs.

With regard to the initial parameters, whose wise choice does dramatically influence

Initial parameters have a large influence on the optimization process and it should be possible to choose them based on what is known already. (nur so vorschlag)

the time needed for the optimization process, there should be the possibility to start from the already known. If the structure could be parameterized in such a way that it allows for embedding already working controllers, it would enable us to find good controllers much faster and independent from each other. For example, position controllers can be used to create initial situations for standing stably, the CSL defined in a later section is very suitable for robots to get up and recover balance, and oscillation capabilities could create the dynamics needed for initial walking movements. Also, reusing controllers that already show appropriate behavior could enable us to refine or generalize given behaviors or adopt a gait from one robot to another.

functional subordinate

ly

still?

upright

dont do this, define it, then use it. helps against defin ing unneeded stuff ;)

and transfer

Finally, the domain of mobile robotics is dominated by bounded energy resources which leads to various attempts to reduce power consumption as much as it is economically. Lighter hardware is inevitably less stiff and brings some sort of elasticity, which is mostly material-friendly but also more challenging to control with traditional approaches. Saving energy also means saving computational efforts, therefore it would be beneficial if the motion controller would work on small-sized microcontrollers and could be computed a hundred times per second to fulfill the real-time requirements for reactive motions like running or jumping which usually need high update rates. Furthermore, if the control structure is defined that it could in principle be parallelized and run on dedicated hardware the energy requirement could be again drastically reduced.

2020-12-25

flexible and elas tic

material-friendly is komisch. lass was besseres finden. gegenteil von 'abrasive'

a

latency

high-speed

kind, gentle, easy, soft, smooth, ..

cost is

able

Having all this in mind, we will now proceed to finding and defining the control structure which will be used for the rest of the thesis. For this, I will briefly recap the basic terms from dynamical systems theory and controller design. Afterward, I will proceed with the discussion of classical and recent controllers for systems with a single output and their application to real robots. And finally, I will present a learnable generalized controller structure as the result of the experiences in creating robot motions so far. This structure will henceforth be the foundation of the simulated and hardware experiments conducted in this thesis.

a

suitable

yes of course

used in all

The approach of choice for robotics and machine learning dealing with physical systems real time sensorimotor loops and neural networks is dynamical systems (refs 1,2,3). Falls sich wer fragt, das ist in opposition to symbolic formal language approa ches

1.1 Dynamical Systems

The theory of dynamical systems, referring to Guckenheimer and Holmes (1983); Thompson and Stewart (1986) and Strogatz (1994), is an important tool for researchers and engineers of robotics and machine learning to describe the properties of physical systems, sensorimotor control loops, and neural network dynamics.

In

A system is called dynamic if its state changes over time. Consider the dynamical system f with the update rule

the

$$\boldsymbol{x}(t) \leftarrow f(\boldsymbol{x}(t-1), \boldsymbol{u}(t), \boldsymbol{\theta}(t))$$

comprising the system's state $\boldsymbol{x}(t) \in \boldsymbol{M}$ at time step $t \in \mathbb{N}$, the input to the system $\boldsymbol{u} \in \mathbb{R}^K$ and the system's parameters $\boldsymbol{\theta} \in \mathbb{R}^D$. The manifold $\boldsymbol{M}$ of all possible system states is called the *state space* or *phase space* and is defined as $\boldsymbol{M} \subset \mathbb{R}^N$. The system starts at time $t = 0$ with $\boldsymbol{x}(0)$. The system is called *autonomous* since it does not directly depend on time.

Du meinst zeitinvariant? autonom heisst 'nur von sich selbst und nicht von externem input abhängig' bei dynsyses, check check.

All the components can have different multiple dimensions, $N, K, D \in \mathbb{N}^+$, and are denoted by convention as column vectors, e. g. $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})^T$. Within this thesis, we focus on discrete time dynamical systems only, so each variable carries a time index. For the sake of simplification, the index is omitted if it is identical on both sides of the equation. To further increase readability for update rules we denote $y(t) = g(y(t-1))$ simply as $y \leftarrow g(y)$ where on the right-hand side there is always the previous time step's value of the concerned variable. Hence equation (1.1) condenses to the more compact and convenient form

$$\boldsymbol{x} \leftarrow f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\theta})$$

A so-called *trajectory* of the system is an ordered set of successive states along which the system evolves. Trajectories never cross each other and can be imagined as the system's path through the state space. A trajectory becomes an *orbit* if it exhibits a certain periodicity, i. e. when the same states are recurring on this trajectory. There are different kinds of periodic orbits: We especially denote a *p-orbit* when the periodicity can be specified by $p \in \mathbb{N},\ p \geq 2$. When the recurring states are not exactly the same but arbitrarily close, we speak of quasi-periodic orbits. For instance, a stable oscillator can be described as a quasi-periodic orbit. In later chapters, such orbits will play a major role since gaits of walking robots can be regarded as multi-dimensional oscillators with the trajectory being a quasi-periodic orbit through the phase space of joint angles and angular velocities.

The vector $\boldsymbol{\theta}$ constitutes the *configuration* of the system and we use it to change the system's properties according to some objective. For instance, we could consider a pendulum with a certain configuration of mass, length, gravity, friction and so forth. One may object, that the configuration should stay constant in the consideration, but since we are considering robots here, the system's configuration is expected to change continuously, e. g. because the hardware is subject to wear. However, we mostly assume the configuration to change way more slowly, so that we can regard them as quasi-constant parameters. For some variables, it may be convenient to either consider them as part of the configuration or as part of the system's state, e. g. the battery voltage.

The control input $\boldsymbol{u}$ is useful to externally manipulate the system. It is part of the system's state but is not updated by the system itself. We can use it to drive the system's state into a desired region of the phase space. Remembering the pendulum, we can imagine $\boldsymbol{u}$ as the force which initially deflects the pendulum before we release is and let it swing until rest.

Attractors

Each dynamical system is characterized by its attractors. Attractors are sets of states towards which the system evolves over time. Attractors can be *fixed points*, curves or even sub-manifolds of the phase space as the aforementioned orbits. Attractors can *co-exist* within the same system. There must be at least one attractor in any dynamical system. If there is more than one attractor in a system, then it depends on the initial conditions where the systems evolve to. The system can evolve in either way and all initial conditions leading to the same attractor comprise the attractor's corresponding *basin*. Each state of the system must inevitably evolve towards one of the system's attractors.

A virtual border between two adjacent basins is called *separatrix*.

The *vector field* describes the evolution of the system, it can help to identify different attractors, basins and separatrices. The vector field is the partial derivative of the system according to each dimension and assigns in each state, $\boldsymbol{x}$, of system a vector

$$x \mapsto \operatorname{grad} f(x) = \nabla x = \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x) \right)$$

Fixed points of the system are states for which the vector field (or the gradient) in that point is zero in all dimensions, $\nabla f = \mathbf{0}$. Hence the evolution of the systems stops in fixed points, since $\boldsymbol{x}(t) = \boldsymbol{x}(t-1)$. Depending on the properties of the field around that fixed point, we either have a stable or unstable fixed point. Stable fixed points are attractors, hence all trajectories starting within the corresponding basin are evolving towards the stable fixed point. So a small control input would have no effect, the system would return to the fixed point. For instance, consider a pendulum subject to gravity hanging towards the ground: A small nudge will cause the pendulum to swing for a moment, but it finally comes to rest in the same fixed point.

Unstable fixed points, on the other hand, are usually located on separatrices and the field is pointing away from them. So a small control input would probably suffice to drive the system away into the next corresponding stable fixed point. Imagine a balanced pole (or inverted pendulum) resting: Here, a small nudge would make it fall towards the next stable state. Unstable fixed points are repellers, the opposite of attractors.

Since quasiperiodic orbits are attractors, they are stable in the same meaning as stable fixed points. Small control inputs will go to the same attractor again. Also, such orbits have basins, i. e. there is a certain region of the phase space leading towards this attractor. If we again consider the gait pattern of a walking robot as such attractor of the system it is inherent to the system that we have certain stability margin (the separatrix). If we could design the gait so that this area of stability is maximized, e. g. by exposing the robot to disturbances during learning, we could create gaits (multidimensional oscillators) that are stable against disturbances and naturally find their way back to the attractor, i. e. the robot will recover balance after hitting a smaller obstacles with its legs.

Figure 1.1: Vector fields for illustration of common attractors: a) fixed point, b) quasi-periodic orbit, and c) two coexisting stable fixed points. For b) and c) there is also an unstable fixed point at $(0, 0)$.

Bifurcation

Given the parameterized system of (1.1), if we change the system's configuration $\boldsymbol{\theta}$, then the attractors of the system will probably change as well. The attractors may just deform, e. g. fixed points will shift and orbits change their size and form. But when attractors disappear, merge or suddenly appear in dependence of parameter changes, then we have found a so-called *bifurcation*. The parameter value at which this occurs is called the bifurcation point. Several bifurcations that occur in succession, are called a bifurcation sequence. When the attractor landscape of a system changes according to the system's parameters, we should be aware of that effect during learning. Hence, depending on the architecture of the system, certain small parameter changes can yield dramatic changes in behavior, e. g. when we apply learning rules and pass a bifurcation point. In particular, recurrent neural network dynamics are known to be prone to bifurcation especially when subject to weight changes during learning as examined by Doya (1993).

Determinism and Noise

When not noted otherwise, the system's transfer function f shall be considered deterministic, i. e. for a each given tuple of inputs $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\theta})$ the output is always identical. Even though this does not fully hold for the given robotic systems, there is no explicitly modeled stochasticity in f. When the function f represents a simulated or physical robot, the (simulated) sensors exhibit a certain degree of noise, obstacles may change the sensors' values in an unforeseeable manner. But even if there would be no noise at all, a deterministic system must not necessarily mean that it is easily predictable. The transfer function may be highly non-linear, so even for a low-dimensional system, small deviations in initial states may quickly lead to fully different outcomes. Remember the chaotic trajectory of a simulated double-pendulum, where the differential equations are purely deterministic, but the system's evolution is not easy to predict. Besides the sensors' noise, most probably the control input $\boldsymbol{u}$ will be stochastic, e. g. if $\boldsymbol{u}$ is created by a reinforcement learning agent, with a stochastic policy, randomly trying out different actions according to some preference given from value functions.

1.2 Control Basics

Controllers read the state $\boldsymbol{x}$ of the system, the so-called *process variable*, and create control inputs $\boldsymbol{u}$ of the *control variable* for each time step in order to reach some objective. In general the controller could make its decisions on the history of system states $\boldsymbol{x}(t), \boldsymbol{x}(t-1), ..., \boldsymbol{x}(0)$ but a more convenient approximation to this form is to provide the controller with a state on its on, memorizing its recent output $\boldsymbol{u}(t-1)$ an only using the current state of the system $\boldsymbol{x}(t)$. The general update rule for such a controller is

$$\boldsymbol{u} \leftarrow c(\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{w})$$

with $\boldsymbol{w} \in \mathbb{R}^L,\ L \in \mathbb{N}^+$, the column vector of the controllers parameters or *weights*. The dynamical system being controlled is often referred to as the *plant*.

Objectives

Objectives can be manifold. For instance, a simple objective could be to minimize the error between the desired value, $\bar{\boldsymbol{x}}$, and the measured value, $\boldsymbol{x}$, of the process variable. The desired value is termed the *setpoint* or *target value* and the error (control deviation) which is to be minimized is defined as:

$$\boldsymbol{e} = \bar{\boldsymbol{x}} - \boldsymbol{x}$$

If a controller's objective is to minimize the distance towards the desired system state, we can think of the system under control as having a new attractor, with the setpoint becoming a fixed point towards which the system evolves. Similar objectives could be to keep the state within certain limits and create control inputs accordingly. The objective must not necessarily be to bring the system into a single state but could also be to tie it on a sub-manifold of the state space. See Figure 1.2 for an illustration of a basic closed control loop with setpoint.

In the simplest case, the objective function considers only a sub-state of the system, e. g. controlling a single robot joint's target position. But the controller can as well affect all dimensions of the state variable and the objective could be a derived measure of the accumulated state, e. g. the average walking speed of a robot being 1 m/s. If we have an objective like this, we can not simply specify simple target states. Rather the state must undergo a certain sequence (trajectory), or in the case of walking, a periodic sequence, an orbit. Hence, in contrast, to create a fixed point as for the simple setpoint, we must shape the state space to have another non-trivial attractor.

Figure 1.2: Structure of a closed control loop with setpoint. The controller reads the system's state x compares is to a target value $\bar{x}$ and tries to change the system via the control input u in order to minimize the control error e.

A comment on notation: Equal to the systems being controlled, we also consider the controller acting in the discrete time domain. In the controller's structure diagrams, we explicitly denote values of passed timesteps with the unit-delay operator, z^{-1} (a common operator when using z-transformation). And just like with the equations, we omit the time index to simplify the figures and increase readability.

Common Controllers

A typical application is position control. So the objective is to minimize the distance $e = \bar{\varphi} - \varphi$ of the measured position, i. e. the joint angle φ, to the target position $\bar{\varphi}$. The objective often incorporates, that the process of distance minimization has further constraints, e. g. to also minimize the time or energy being used for this transition. For instance, if the system is a DC motor, the control variable could be the motor's voltage and we would expect a change in position according to the voltage applied.

The same way we can control the velocity of such a system. We choose a target velocity $\bar{\omega}$ and compare it to the measured actual velocity $\omega = \dot{\varphi}$. Controllers can be cascaded, e. g. a position controller outputs a target velocity which is then again fed into a velocity controller, creating voltage values. Controllers can as well operate in parallel, e. g. the aforementioned controller cascade is further enhanced by a limiter, that checks if the mechanical boundaries of the motor are respected and creates control values to protect the motor from breaking when the position is getting out of bounds. Another parallel controller could check for the maximal current consumed or the allowed temperature of the system.

1.2.1 Proportional-Integral-Derivative Controller

PID controllers, referring to Aström and Hägelund (1995), are excellent in bringing the system into certain target states, e. g. position or velocity control. With a given setpoint and the control deviation, the controller creates an attractor around the target value. Our desire is to create a fixed point there, and around it a non-curling strong vector field towards this fixed point. So, we usually want to minimize the time needed to reach the target and avoid large overshoots.

A *proportional controller* is stateless. Its output is directly proportional to the control deviation $u_p = k_p e$. Its control response is, therefore, a step function. The proportional controller is very reactive and very well suited to bring the process variable quickly close to the target value. If no arbitrarily small outputs are possible, e. g. due to technical limitations or friction, the P-controller may not always fully compensate the control deviation. For some purposes, it may be sufficient to simply increase the gain k_p and to limit the output. But we should be aware that excessive gain can lead to overshooting, particularly if the system being controlled has inertia.

An *integral controller* has an internal state which accumulates the control deviation. Such a controller produces an output proportional to the sum of the control deviations, $u_i = k_i \sum e$. A residual control deviation, as in the case of the P-controller, must after a few timesteps inevitably lead to a control output with a sufficiently large amplitude to cancel out the deviation. The name integral controller suggests the use of integrals, however, in the discrete-time systems we are considering, we can approximate the integral with a simple summation. An integral controller is somewhat slower than a pure P-controller (low pass effect), so it tends to overshoot. Permanent deviations lead to the so-called *wind-up*. In the case that the process variable is subject to static friction, it usually comes to limit cycles that could require special handling. The proportional and integral controller can be easily combined (PI), and partially compensate for each other's drawbacks.

Sudden changes to the setpoint lead to jumps in the control deviation, which may be further amplified by the proportional component. But if we remember the last error, and if we add a *derivative* term $u_d = k_d \Delta e$, we are able to catch the fast transients of the P-controller. The D-component is in principle a simple high-pass filter, if we subtract this from the signal, we create damping or further shape the control response. Considering time-discrete systems only, we use the difference as the simplest form of

discrete derivative. The differential quotient becomes the difference quotient

$$\frac{de}{dt} \approx \frac{\Delta e}{\Delta t} = \frac{e(t) - e(t - \Delta t)}{\Delta t}$$

where the factor Δt^{-1} is implicitly included in k_d. The D component itself cannot compensate for a control deviation, so it usually occurs in combination with the P controller.

Figure 1.3: Structure of the PID controller (left), responses of control output to constant deviation (right). The PID is a parallel structure, the error signal e is split into three branches, and the control outputs of the individual component is added up and applied simultaneously.

Combining all three basic elements results in the well-known *proportional-integral-derivative controller* as depicted in Figure 1.3. In fact, three controllers are acting in parallel on the same system and are hence influencing each other. For instance, the proportional term acts on the control variable like the restoring force of an ideal spring with the error as the deflection. The derivative term acts on the error's velocity, i. e. as a damper. The PD parts of the controller can thus be treated like a spring-damper system if the controlled quantity is a position. This is an interesting property in terms of legged robots since we can adjust separately the stiffness and the damping of the joints.

1.2.2 Non-linear extensions to PID control

Linear control is often limited to certain linearized domains of (in general) non-linear control systems. Or maybe the underlying system is just assumed to be linear for a given range of input values. However, at the edges of such a quasi-linear domain, we probably must change the controller's properties if the controller should be working beyond this domain. For instance, we can imagine a position controller, which is handled differently if we are far away or close to the target position.

When using proportional control, we usually want a high gain, to quickly reach the target. But with high gains, we usually must limit the output to avoid overshooting, so we have found our first non-linear element. When using an I-component, if the control output is limited, whether by the controller itself or by the system, this leads to wind-up. A common anti-wind-up technique is to freeze the I-part when reaching a predefined limit and thus to avoid an unnecessary delay when we have to do the rewinding.

If systems with friction are considered, permanent control deviations (P) or limit cycles (I) can occur. A simple deadband like

$$f_{db}(x) = \begin{cases} x, & |x| < \kappa \\ 0 & else \end{cases}$$

with $\kappa \in \mathbb{R}^+$ can help here. However, this merely forces the controller to give up and settle for the residual control deviation. For high precision systems, we need to find a different way to overcome the residual without causing potential limit cycles. A smart solution is provided by Wescott (2014), proposing to switch to a low-frequency PWM-like mode, if the system is very close to the target. Instead of further decreasing the control output (P) or winding it up (I), we simply keep the smallest output, which can change the system within a single timestep, and create pulses with decreasing duty cycle. This ensures, that the system will move because the output is high enough for the moment, but the pulses get shorter as we approach closer to the target. The controller somehow shakes the system closer to the target value, trying to avoid the stiction domain (refer to **??**).

So, even though most of our controller models are considered linear in their pure theory, the real-world controller applications are often highly non-linear and we need to do a lot of modifications to make them work as desired. To deal with non-linearities is the general case with real hardware. But before we continue with fully non-linear controllers we should consider to refine the objective and questioning the approach of controlling setpoints and then see what benefit we can get from this strategy.

1.3 Cognitive Sensorimotor Loops

Position control is an intuitive approach to motion control of robots in general. Here, the trajectories in the form of target angles are calculated in advance for each joint and then attempted to comply with the aid of PID position controllers. This approach is well suited for relatively static environments, such as in industrial applications with

Figure 1.4: Approaching the setpoint under friction, ideally vs. realistically (left), control output of (anti stiction) PWM mode (right).

good predictability, and also has the advantage that the design of the controllers and the calculation of the trajectories can be considered separately.

The basic disadvantage, however, is that typical environments for humans (where we want our robots to be) tend to be more unstructured, so it may be beneficial to move around with legs. Even if the robot already has a good model of the environment, and thus for many situations, it is clear where the next step could be taken, it poses the problem that for faster movements such as running, this step planning may run into its limits and produces less good predictions. Furthermore, the environment or the sensors could not allow for a step planning in principle, e. g. when walking in the dark or in the tall grass.

Forces and torques must hence be taken into account. Additional sensors and a sophisticated model of the robot allow it to record the active forces and torques during running and to include them in the step planning in order to compensate for any disturbances. However, as the number of joints increases, both modeling effort and inverse dynamics computation become more and more problematic. Furthermore, it can not generally be assumed that the model of the robot is immutable. Both wear and payload would have to be incoporated into the model beyond a certain extent.

Therefore, we need robust controllers that can cope with variations in the situation, inaccuracies in the model, and with light loads. But before it goes to learning, controllers are to be discussed, which can already generate very robust movements. Such controllers are for instance the Cognitive Sensorimotor Loops (CSL). CSLs are able to generate a wide range of different behaviors with only a few parameters, and these have always well-defined areas and evenly merge the generated behaviors as shown by Hild and Kubisch (2011); Kubisch et al. (2011a); Werner (2013); Bethge (2014), and Meier (2015). This is ideal for learning algorithms where small changes in the parameter space ideally should only lead to small changes in the behavioral space.

Definition

A sensorimotor loop generally refers to a connection of sensory inputs of a system to its actuators that affect the environment and, in turn, leads to measurable changes in the

inputs. CSL is such a sensorimotor loop and can be seen in Figure 1.5. The associated system in the following is a rotary joint with an axially mounted sensor and motor. The

Figure 1.5: Schematics of the Cognitive Sensorimotor Loop. The velocity signal is amplified using the input gain g_i and integrated. Depending on the configuration (g_i, g_f) this loop creates four distinguishable behaviors.

only sensory input is the measured absolute joint angular velocity, $\dot{\varphi}(t)$. The only output of the CSL is the motor voltage $u(t)$. The controller is defined with the discrete-time update rule:

$$\begin{aligned} z(t) &= g_f u(t-1) \\ u(t) &= -g_i \dot{\varphi}(t) + z(t) \end{aligned}$$

In the model it is assumed that the joint angle $\varphi(t)$ and the motor voltage $u(t)$ have the same direction of rotation, i. e. if $u(t) > 0$ the motor accelerates to $\dot{\varphi}(t) > 0$. CSLs are able to generate astonishingly complex behavior with minimal computational effort. They are closed loop controllers and have an internal state, z. If the joint velocity is not available as a raw sensor value, but the joint angle, or if the velocity would be anyway derived from the position sensor, then it is more convenient to use the following equivalent form of the CSL, as it was originally published:

$$\begin{aligned} z(t) &= g_f u(t-1) + g_i \varphi(t-1) \\ u(t) &= -g_i \varphi(t) + z(t) \end{aligned}$$

This form only uses the joint position and derives the velocity with the help of a simple differentiator, an FIR high pass filter, actually.

Configuration

Cognitive Sensorimotor Loops have four, clearly distinguishable behavioral modes: Contraction, Hold, Release, and Support. All four known modes are variable in their strength, they cover a 2-dimensional parameter range; if this range is traversed slowly in a certain way, the behaviors also blend smoothly into each other. Following this path through configuration space, as described in Figure 1.9, there are bifurcation points, where the behavior qualitatively changes.

Figure 1.6: Schematics of the Cognitive Sensorimotor Loop (original form) with position input. The position signal is derived by a simple differentiator. This form can be used if no velocity signal is provided. Similar to the direct form II of digital filters, the differentiator and the integrator share a common delay-unit.

Support	Release	Hold	Contract
$g_i < 0$	$g_i > 0$	$g_i > 0$	$g_i > 0$
$g_f = 0$	$0 \leq g_f < 1$	$g_f = 1$	$g_f > 1$

Table 1.1: Overview of different behavioral modes. Each configuration generates a different specific behavior. Figure 1.7 illustrates the behavior on a simple pendulum arm.

Support Mode This mode proportionally feeds back the positive velocity, i. e. in the direction of the current movement, thus allowing partial compensation of friction (which is negatively proportional to the direction of movement). It can also be used to generate comparatively slow oscillations, if there already is a restoring force around a stable fixed point in the physical system, e. g. to swing-up a pendulum subjected to gravity.

Release Mode This mode slows down movements, similar to friction it counteracts the movement, it decelerates and finally brings the system to halt, with $g_i > 0$ and regulating $0 < g_f < 1$ in combination, we can adjust the release mode from soft damping to instantaneous braking.

Figure 1.7: Behavioral modes of the CSL demonstrated on a single joint (from left to right): The *support* mode further amplifies the measured velocity and compensates for friction. The *release* mode negates the velocity and acts like additional friction. The *hold* mode remembers the relative position and tries to restore it while *contraction* mode detects applied forces and works against them, e. g. gravity.

Hold Mode This mode works in a similar way to a position controller, except that the target position is set at the moment that the mode is activated. Since only the joint's speed is input to the system, which is integrated and fully maintained (due to $g_f = 1$), the hold mode remembers the deviation from the target position to be held, and largely corresponds to a proportional controller.

Contraction Mode The most complex behavior is generated in contraction mode, where the system works against all external forces, friction, and gravity. The negatively fed back velocity is integrated and further amplified by the feedback parameter $g_f > 1$. The controller detects the direction of the applied force and applies reaction forces. This mode works against the current movement direction, but not in proportion. Due to the feedback weight of greater than 1, the contraction mode is able to *remember* the last applied force and goes against it. In addition, the contraction mode can move out of stable fixed points and hence converts stable fixed points into unstable and vice versa. The contraction mode works against gravity. It lifts a pendulum arm and stabilizes it in the previously unstable fixed point. When there are forces acting on the system, the contraction mode will convert the detected speed into an opposite force until the cause of the speed change was overcome. If this should not be possible, e. g. when hitting an obstacle or joint limits, the CSL in contraction amplifies to the maximum and retains its state, pressing with full power. Referring to the dynamical system's view: If contraction mode changes roles of fixed points from stable to unstable and from unstable to stable, what if there was no unstable fixed point beforehand? Or what if the force to be applied by the controller is not sufficient to go there directly? Pure application of contraction mode will let the controller get stuck in systems like these and should be avoided. There will be comments on this topic later on in this section.

Figure 1.8: Illustration of the contraction mode acting on sample joint configurations which are subject to gravity. Joints in contraction mode will defy gravity and raise themselves.

Implementation and Parameter Adjustment

The arithmetic effort for a joint with CSL is constant in time and is only a few lines:

```
z = gi * phi                # initialize
def loop():
    u = sat(-gi * phi + z) # output
    z = gi * phi + gf * u  # integrate

```

The initial values are $u(0) = 0$ and $z(0) = g_i \cdot \varphi(0)$ and the clipping function is $\mathrm{sat(x)} = \min(1, \max(-1, x))$. The central parameter is the input (or loop) gain g_i. If we keep the feedback weight constantly $g_f = 1$ (i. e. hold mode), we can gently increase g_i. In hold mode, we can use the analogy of a spring. The loop gain corresponds to the spring constant, i. e. a higher value leads to a stiffer spring. Starting at $g_i = 1$, we can slowly increase the parameter to the desired stiffness. At some point, the friction of the system is no longer sufficient, and an increase in loop gain will initially lead to slight overshoots and even further increasing it will end in a heavy oscillation. For instance, we could set the loop gain to reach a *critical damping*, hence if we deflect the joint in hold mode, it should return as quickly as possible but without overshooting. After having adjusted the input gain, we can go on with the feedback gain g_f. For contraction mode, this parameters must be greater than 1. Increasing g_f must be done very carefully since a too fast amplification of the internal state will quickly produce maximal power at the output.

Figure 1.9: Meaningful parameter values for different behavioral modes: The identified modes are located on a sub-manifold of the CSL's configuration space. Other areas might lead to undesired high-frequency oscillations (flip-bifurcation).

Experiments

So far I applied CSLs to the humanoid robot Myon by Hild et al. (2011) in three different experiments, and these experiments offered a lot of potential follow-up applications and heavily inspired the generalized control scheme presented at the end of the chapter as the result of this process.

The first experiment started simply as three independent CSLs were applied to a single leg of Myon. It started out as the attempt to stabilize the leg against gravity using contraction mode, hence considering the leg as a double inverted pendulum (refer to Figure 1.8). But when this leg was put into a horizontal orientation, it turned out, that it was able to raise itself a bit. After half an hour of manually tuning the CSL configuration the leg was able to fully raise from lying to standing upright. The capability to work against forces in contrast to reach setpoint positions is superior in reaching

certain positions in phase space without pre-defining a trajectory. Nothing had to be switched during this complex motion, the leg merely followed the path of strongest resistance which in this configuration ended in the upright position. Figure 1.10 shows the sequence of events in combination with the individual joints' data of angle and motor output.

The second experiment shown in Figure 1.11 applies CSL contraction mode to the lower torso and legs of Myon to create a robust push rejection and balance recovery. In contrast to the first experiment, we now have two contact points with the ground. Thus, we have to introduce a coupling between symmetrically mounted CSL driven joints to avoid that joints work against each other but favor *cooperation*. The coupling measures the activity of the joints and synchronizes the torque direction, for details please refer to Kubisch et al. (2011a). Within reasonable bounds, the balance could be recovered even with moderate nudging. The balancing algorithm was heavily used for a lot of *Myon Language Games* within the ALEAR project, where the multiple robots performed gestures with the head and both arms yielding quite dynamical disturbance to the robot's center of mass.

The third experiment uses different CSL modes for creating said gestures for lifting arms and weaving similar to how aircraft marshallers do. CSL modes change according to the current arms and body pose so that sequences of motions could be performed, e. g. lift the arm and when reached the maximal position, move the arm away from the body. The CSL was modified to change modes from contraction to hold and finally to release with only modifying a single input. Concurrently, the body is stabilized using the algorithm described in the second experiment.

Even though the parameters for these experiments were defined manually, it could be demonstrated that applications with increasing complexity could be derived from a very simple principle. This encourages for more complexity when we consider more interconnections to modulate the behavior and the possibility of adapting several parameters w. r. t. an objective function.

Limitations

The CSL shows that complex behavior can be achieved in principle with the simplest sensorimotor control loops. But most of the time it needs a bit of sensitivity and implementation work around it to master a CSL. Particularly for the protection of the end stops of the motors must be taken care of separately, since a CSL in contraction mode tries to overcome this limiting resistance, if necessary with maximum power. Setting the CSL in release mode and resetting the internal state as soon as it comes close to the end stops has proven itself as a valid strategy for protecting the joints from damage.

The CSL is not a position controller and is not suitable for positioning tasks. Actually, it is not a controller in the classical sense, because there is no explicit target quantity. CSLs have very good potential for push rejection and balance recovery and demonstrated in the example of Figure 1.11 and for creating quasi-static movements, e. g. for body poses, which have to get over acting forces such as gravity and adapt to body model changes such as load in particular without absolute positioning as demonstrated in the last example in Figure 1.12.

Figure 1.10: Example 1) A single leg of the modular humanoid robot Myon, where each of the three joints is controlled locally by a CSL in contraction mode, While there is no direct communication between the controllers, complex behavior arises solely from the interaction of the acting forces.

However, if high-dynamic movements are to be generated, CSLs could not yet shine. There are attempts to make CSL motions faster, e. g. by Werner (2013), and create more multi-joint behaviors such as an assisted standup from sitting by Meier (2015). And although it is possible to create a stable oscillation with a contracting CSL with a low loop gain, there is so far no example of a full-body high-dynamic motion such as walking or running for legged robots.

Figure 1.11: Example 2) Balance Recovery of Myon's torso with legs. The top sequence shows the torso recovering from being pushed and the bottom sequence the balance recovery after being tilted. As can be seen, the hip and torso joints start from a situation where the CSL already adapted to the tilt.

1.4 Advanced Non-linear Controllers for Multi-Joint Robots

In the previous sections, we defined the basic terms for dynamic systems and presented common controllers, as well as more recent sensorimotor loops which mostly acted on single joints. And even though they were used in behaviors for multi-joint robots, mostly their interconnections between joints were sparse. I will now present an approach of how to generalize both, the PID and the CSL for multi-joint robots. We will see that both structures can be regarded special cases of the same more general structure and that the result will be even more powerful in the sense that it also contains a lot of more functions such as logic and oscillators.

For instance, with the following structure, we will find multi-dimensional PD controllers with their setpoint created from the full body state or we could identify CSLs that change their behavioral mode dynamically for one joint—e. g. from contracting to release—depending on the state of other joints. These structures will also have the capability to oscillate, either open or closed loop.

But before we continue, we need to specify the sensor and motor space, which is input and output to the generalized controller. So far we only needed to regard a single joint's state, its angle, and angular velocity. Now we do consider robots with an unspecified number of joints. Even though it is not mandatory to do so, we shall further on assume that for each joint the angle, velocity and the control output exist and will have consistent signs. When discussing the properties it is convenient to think of revolute joints with their usual actuators and sensors, but at the end of the chapter it will be quite clear, that the intention points towards a controller structure which does not necessarily depend on having labeled inputs and outputs, rather it could be handled as if we do not know what

Figure 1.12: Example 3) Sequence of gestures using CSLs. Arms are raised using contraction mode and taken down using release mode. The CSLs for shoulder pitch/roll and elbow joints are activated according to the position of the other joints in order to create different contraction/release sequences. The execution speed of each gesture sequence can be modulated by different contraction strength and even halted at any time using hold mode. Since the angular momentum generated by the moving arms would easily make the robot fall over, CSL-driven joints are used concurrently to stabilize the torso and legs as already described in Figure 1.11.

kind of inputs we have, what the actuator is doing and how it is influencing the system. So we should have in mind that algorithms that build on top of this structure will not care about the kind of inputs and outputs, especially when it comes to learning the parameters. Rather we will change the parameters according to some objective measure as if there were no information about the model available, i. e. unlabeled sensors and actuators.

1.4.1 Sensor and Motor Space Definition

For our robots presented in chapter **??** the system's state $\boldsymbol{x}$ comprises all sensor readings. It is the observable state of the system. To be close to hardware experiments, we should only incorporate state variables that could actually be measured or derived from basic measurements. The sensor vector may also contain higher level features but we should stick to features that are available on real robots as well.

The sensor vector is defined as $\boldsymbol{x} = (\boldsymbol{\varphi}, \dot{\boldsymbol{\varphi}}, \boldsymbol{v}, b)^T$ with $\boldsymbol{\varphi} \in [-1, 1]^K$ being the vector of all joint angles, $\dot{\boldsymbol{\varphi}} \in [-1, 1]^K$ the vector of their corresponding angular velocities, optionally the estimated body velocity $\boldsymbol{v} = (v_x, v_y, v_z)^T$ and the bias $b = 0.1$. The dimension of $\boldsymbol{x}$ is $D = 2K + 3 + 1$, where K is the number of joints. All sensors are normalized, quantized to limited resolution and a little noisy as described in **??**. Analogously, the

motor outputs are defined as $\boldsymbol{u} \in [-1, 1]^K$ and will be treated as raw motor outputs without additional control loops involved.

Requirements

Since the denoted controller can be regarded as a neural network and since there are potentially many network architectures, we will stick to the simplest, minimalist form that generalizes the aforementioned architectures. So there are some requirements to the controller that should be named here:

We need some form of non-linearity to create saturation (limiting), stable oscillators and logic. Saturation is mandatory since we have probably many interconnections between inputs and outputs of several joints, which could easily lead to overexcitation and large numerical values, or even worse, systems with positive feedback will quickly grow out of bounds if there is no limiting. On the other hand, the non-linearity should ideally be monotonic and differentiable to make it easier for optimizers to find suitable parameters. Good examples are the *tangent hyperbolicus* or *logistic function*, but also the rectified linear unit or the *clipping function* will suit our needs, even though they are not differentiable in every point, their piecewise linear parts do so.

For the beginning, we should try to keep the explicit state information low. We should not put too much into the controller, but rather consider to switch or blend controllers (automatically) if the situation changes and the current controller is not suitable anymore. This is important because we have feedback in the system, we therefore have a recurrent network topology which easily can produce chaotic behavior with unsuited parameters. Hence our network will be shallow, i. e. will have no hidden layers. This way, we could (pre-)learn controllers for different situations or walking directions in advance and combine them later in an appropriate way. As will be shown in the experiments, to create periodic patterns of a certain complexity the state information of this structure is already sufficient.

There are a lot of possible extensions to that system, we could easily introduce more hidden layers of recurrent connections, to improve the versatility of a single controller and make it more robust. But with every bit of complexity we add, we make the analysis a lot harder. If we keep the controller simple we gain a better understanding, a better pre-training, easier implementation, and testability, as well as less computational resources needed.

Current research is heavily focused on so-called *end-to-end learning*, where they use a single network which is a union of state detection, action selection, and controller. And there already exist methods to train these networks. But following this approach we create networks that are very hard to understand why they're doing this and that. If we stick to a modular system, where state detection and action selection is not considered part of the controller, then we always have the chance to fully understand what the system is capable of. Modular architectures with clearly distinguishable components can also benefit from the fact that modules are replaceable, and we could increase the system's performance in step by step and test the modules separately.

1.4.2 Generalized Non-linear Controller

For the experiments we define the *generalized controller*,

$$\boldsymbol{u}_t = f(W\boldsymbol{z}_t) \qquad \boldsymbol{z}_t = \begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_{t-1} \end{pmatrix}$$

a simple to implement controller structure which is powerful enough to create complex locomotion behavior. The controller's core is the weight matrix $W \in \mathbb{R}^{K\times(D+K)}$ whose *weights* are subject to optimization and shall be found during learning. The transfer function is $f(\cdot) = tanh$ which squashes the motor output smoothly to the interval $[-1, +1]$. This controller can be seen as a simple neural network with fully forward connected weights and a differentiable transfer function, i. e. it is learnable with (stochastic) gradient descend. It is a closed loop controller featuring short term memory through recurrent connections.

Figure 1.13: A generalized controller structure: The controller input comprises the states of all joints as well as the controller's last outputs (similar to Networks by Elman (1990)). The mapping from input to output is fully connected and a non-linear squashing function creates saturation.

This controller structure was selected in order to reduce the assumptions about the robot platform under test. A big plus is that this structure is general enough to embed classical PI and PD controllers as well as the more recent *Cognitive Sensorimotor Loops* (CSL). Hence, we are able to initialize our controller's weights with known parameters and control loops which have already proven to work.

As will be seen, commonly used PI and PD controllers, as well as the CSL, are special cases of (1.4.2). This feature can be used to ensure appropriate initial robot conditions for learning experiments, e. g. we can place the robot safely standing upright when starting evolution or reinforcement learning algorithms. We will make use of this in **??** either by giving PD setpoints or use the hold mode ($g_f = 1$) of the CSL. However, this embedding will only be exact within the approximately linear domain of the transfer function.

Relation to Cognitive Sensorimotor Loops

The embedding of CSL is trivial, which can be seen from the short form of (1.3), which is $u(t) = -g_i\dot{\varphi} + g_f u(t-1)$. Thus everything we need for a CSL for each joint is already there. Furthermore, cross-connections will allow for CSLs which are able to influence each other, e. g. inhibitory or excitatory connections, which promisses enormous potential.

Relation to PI and PD Control

The embedding of PI, PD, or even PID controllers is not directly obvious. Even though the necessary sensory inputs are available in form of the individual joint angles φ_i and their corresponding velocities $\dot{\varphi}_i$, it is neither evident what the setpoint will be nor how the integral of the deviation is calculated.

For an arbitrary joint, let u_{pd} be the output of the linear PD controller which tries to minimize the output error $e(t) = \bar{\varphi}(t) - \varphi(t)$ between a reference angle $\bar{\varphi}$ and current joint angle φ, and let k_p, k_d be the constant parameters for the proportional and derivative terms. Furthermore, let the reference angle be constant, i. e. the setpoint shall not depend on time here:

$$\begin{aligned}
u_{pd}(t) &= k_p e(t) + k_d \Delta e(t) \\
&= k_p \left[\bar{\varphi} - \varphi(t)\right] + k_d \left[(\bar{\varphi} - \varphi(t)) - (\bar{\varphi} - \varphi(t-1))\right] \\
&= k_p\bar{\varphi} - k_p\varphi(t) - k_d \left[\varphi(t) - \varphi(t-1)\right] \\
&\approx k_p\bar{\varphi} - k_p\varphi(t) - k_d\dot{\varphi}(t)
\end{aligned}$$

We now associate the weights $w_{ii} = -k_p$, $w_{i(i+M)} = -k_d$, and the bias $w_{iN} = k_p\dot{\varphi}$. All other weights are set to zero.

For the PI controller, we make use of the z-transformation to bring the right side of the equation in a more convenient form and proceed similarly as in (1.4.2).

$$\begin{aligned}
u_{pi}(t) &= k_p e(t) + k_i \sum_{n=-\infty}^{t} e(n) \\
u_{pi}(t) &\ \circ\!\!-\!\!\bullet\ U_{pi}(z) \\
U_{pi}(z) &= k_p E(z) + k_i \frac{1}{(1 - z^{-1})} E(z) \\
(1 - z^{-1}) U_{pi}(z) &= kp(1 - z^{-1}) E(z) + kiE(z) \\
u_{pi}(t) - u_{pi}(t-1) &= k_p \Delta e(t) + k_i e(t) \\
u_{pi}(t) &= k_p \Delta e(t) + k_i e(t) + u_{pi}(t-1) \\
&= -k_p\dot{\varphi}(t) + k_i\bar{\varphi} - ki\varphi(t) + u_{pi}(t-1)
\end{aligned}$$

If PID controllers shall be embedded, we would furthermore need $\varphi(t-2)$, which is in the current form not available.

While the aforementioned controllers work mainly in the linear domain of the transfer function and are only approximations of their original form in the non-linear domains, we can, on the other hand, make explicitly use of these non-linear regions. If we do so, the proposed controller structure gets a wealth of additional functionalities.

Figure 1.14: Tangens Hyperbolicus + model of a neuron.

Logic and Relational Operators

Looking at individual neurons of this structure, we can see that Boolean logic can be represented easily. Figure 1.15 shows, that in principle all logical relationships can be composed of the elements contained in the controller. It is thus possible for the controller to activate or inhibit outputs depending on other outputs. Common to all of them is a high preamplification in order to saturate the signal and thus to quantize the values to (tanh) binary $[-1; +1]$, which represents the truth values, true $(+1)$ and false(-1). For the sake of convenience, in the figures, the bias is written into the neuron. Noteworthy is the implementation of the XOR gate. As the only exception, a further time step and multiple neurons are needed. This is due to the non-linear separability. The mapped logic operations are only exemplified with two inputs, but in most cases can be easily generalized by adjusting the bias accordingly, e. g. to form multiple-input OR gates.

Figure 1.15: Examples for basic logic operators created using neural networks within the structure of the generalized controller. Thanks to the transfer function we can saturate the inputs by large weights and make them binary. Negative signs create inversions of logic inputs. Using the bias we can offset the neuron and create simple logic gates, which can be composed to form practically any gate as demonstrated with the XOR.

When it comes to comparing values, we need relational operators such as $<, >, =$. By subtracting two inputs, and amplifying the outcome, we can create a *comparator*, which tells us in a binary output, e. g. if $A > B$. If we further add a self-coupling of greater than one, we add some hysteresis and hence stabilize the result and avoid flickering around if both inputs are already close to each other. This is a neural equivalent of the Schmitt-Trigger gate as shown in Hild (2007).

Figure 1.16: Relational operators build using neurons: The difference between inputs is amplified to either get 1 if $A > B$, -1 if $A < B$ or zero if $A = B$. When more robust representations are needed we can create some hysteresis by simply adding a self-coupling weight $w_s > 1$. The breadth of the hysteresis depends on the input gain and strength w_s (for more details refer to Hild (2007)).

Oscillation Capabilities

As stated earlier, gaits of a robot can be regarded multidimensional oscillators, thus our proposed control structure must be able to create an oscillation, either on its own in a feedforward mode, i. e. a central pattern generator, or better must oscillate together with the connected system as a whole. For the first case, the capability can be demonstrated by utilizing an SO(2)-network as proposed by Pasemann et al. (2003) and Hild and Pasemann (2007). To create a stable oscillation we only need to fully interconnect two outputs by weights r (denoting the ring coupling) and s (denoting the self-coupling). The SO(2)-network works in principle as two interconnected low pass filters with negative feedback. Due to the *inertia*–the low pass filter effect—the negative feedback has no immediate effect, but a delayed one.

For the second (preferred) case we incorporate the hardware and naturally get the inertia since neither the motors will immediately react to inputs but take some time to gain velocity nor the attached limbs. Due to their mass and inertia, they withstand an instantaneous acceleration and act like the required low pass filter. In contrast to the feedforward generated oscillation, we will refer to them as *inherent oscillators* as they are able to stop when the hardware does. They are able to react to the environment and oscillate always in synchronization with the limbs being controlled.

Referring to Figure 1.17, we see a single simplified limb in the form of an actuated revolute joint acting on a lever with mass. We have at least two possibilities to create an inherent oscillation with such a simple system. First, we can measure the deviation angle and derive the velocity from that, and positively feed back the velocity to the motor.

This correlates to the CSL in support mode but due to the restoring force caused by gravity, the system will soon start to oscillate if we deflect it a little from the origin. If we manage to adjust β just as high as to compensate for the dry friction, we got a stable oscillation.

The second possibility to create an inherent oscillation is similar to the SO(2)-network, but instead of having two neurons, we replace one by the hardware and again measure the angle, but then feed it back negatively and delayed, so that we create a somewhat mistuned I-controller. In contrast to the usual case our setpoint is implicitly given here and the integral amplification is too high so that we explicitly create overshooting and increase the I-gain until the oscillation got stable.

Figure 1.17: Demonstration of two mechanisms to create inherent oscillators with a single robot joint: Positive velocity feedback (on the left), when there already is a restoring force e. g. gravity, and negative delayed position feedback (on the right).

1.4.3 A simple bipedal gait pattern

The basic idea behind the walking behavior of Myon is to generate a clock signal from the robot's sensory data and to feed it back to the PD-controllers of the motors. This idea is based on the assumption that during a periodic walking motion, the sensors also generate periodically recurring signals and that negative feedback of these signals, including a phase shift induced by the body inertia, leads to an oscillation (refer to section...). With this principle, periodically recurring movements can be generated as has already been demonstrated by Werner (2008) and Kubisch et al. (2011b) for a small-scale humanoid robot. These motions include many types of walking movements, stepping sideward, backward, turning and the like. Exceptions are very slow or non-periodic movements. There is some similarity to the central pattern generator approaches with the difference that the periodic pattern is generated by the robot itself, hence this approach is considered closed-loop.

Figure 1.18: Myon Walking, Basic idea: A periodic signal is generated from the sensor data and negatively fed back to the joint motors. Depending on the quality of the parameters, this leads to more or less stable periodic movements. By suitable choice of the parameters, it has so far been possible to generate forward, backward and turn-around.

For walking, the created behavior is symmetrical, i. e. the left and right half of the body perform the same movements apart from phase shifts, Hence, the structure requires a sensor space calibrated almost symmetrically to the sagittal plane. Joints are always considered in pairs and also parametrized in pairs, so the symmetry is always given and the parameter space is reduced meaningfully to most desirable symmetrical running behavior. However, there is nothing wrong with partially softening this symmetry, for example, to achieve directional or sideways steps. The Figure 1.19 shows the idea schematically.

For the gait presented here, the hip roll angle sensors were identified as appropriate sensory input data. As the joint angles get larger as a leg moves outward, the signals are subtracted from each other and give the strength of the shearing of the legs. This signal is saturated, phase-shifted and fed back to the joints. The joints are controlled by a set of simple PD-controllers which compliantly try to keep their target angles. The

Figure 1.19: Myon Walking, principle: Simplified illustration of the practical implementation of the basic idea: The signal is generated at the hip joint sensors, processed, and changed separately for each joint pair and fed back as setpoints to the joints controlled by PD-controllers.

signal to be injected is then added as an offset to this target angle in order to shift the desired position of the leg. By using simple PD-controllers, this approach provides a desirable compliant control because of the missing I-part. For each joint pair, there must be defined as the default target position, the coupling strength, and the phase shift.

The resultant gait is depicted in the picture series of Figure 1.20. Despite that this walking had the appearance of a penguin-like gait, it demonstrated that even with a very limited number of parameter tunings, a considerably simple and stable walking pattern could be achieved, even without any help of automated parameter search or modeling. In relation to the controller architecture discussed in this chapter, this motion principle is fully compatible since all building blocks, except the phase-shift are available. However when feeding the inputs not exclusively from a single source—the hip roll joint angles in Myons case—but rather feeding them from several sources, e. g. other joint angles or velocities, there is huge potential for finding sources with the correct phase.

Figure 1.20: The Myon robot while walking with small steps. A penguin-like gait pattern, similar to toddling, emerges from reconstructing a periodic (clock-)signal from sensory data and feeding it back to the leg motors with individual phase and amplitude. The created dynamical systems exhibit co-existing attractors such as a fixed point for standing and a quasi-periodic orbit for toddling. An intended laterally applied nudge suffices to leave the fixed point and enter the stable gait pattern. Vice versa, gently holding the robot, stops the oscillation again.

1.4.4 Reducing the number of weights

The number of weights for a robot using the above defined generalized controller is $n = K(D + K)$, for K joints, e. g. a simple 4-DoF robot already has $n = 52$. So a major drawback of the aforementioned controller is that the weight matrix grows quadratically with the degrees of freedom. In order to reduce the number of weights, we apply the following two constraints:

Symmetries

First, we share weights by using inherent body symmetries. For behaviors which are symmetric to the sagittal plane such as walking forward or backward, we can constrain

$$w_{ij} \stackrel{!}{=} w_{ji}$$

if i, j are symmetric joints. For instance, imagine the motor output of the right shoulder joint is composed using the velocity input of the left knee, then, in turn, the left shoulder's output must be composed using the right knee's velocity input. We will refer to this as the *symmetry constraint*. However, this is only suitable for unsymmetric behaviors too, such as turning, if there is enough unsymmetrical sensor input to help to break symmetry.

The symmetry constraint helps to reduce the computational effort and the learning time a lot. For instance, walking forwards using symmetrically constrained gaits is more likely to move with less curvature. If the fitness function is to move straight without leaving the track, then it probably takes more time for the unsymmetrical gait to get straightened. In the end, the result might be similar, but it usually takes longer and the resulting weight matrix has more unique parameters. Even though the behavior may finally evolve symmetrically, the parameters must not necessarily evolve symmetrically too. There is plenty of possible configurations yielding the same behavior for the general case, but this could lead to different behavior in the edge cases, depending on the inputs.

Figure 1.21: Sensory data of Myon Walking: Depicted is one minute of Myon joint angle data while walking approximately 42 steps. The roll/pitch angles of hips and ankles are majorly involved, the other leg joints are not involved in the gait pattern and just compliantly hold their positions. As can be seen, due to friction there are several fixed points for standing. The robot was walking over common office carpet, so small disturbances due to irregularities could be compensated for and the gait returns to the attractor again.

So e. g. stepping over small obstacles will look different depending on whether it is the left or right leg stepping first.

Normalization

Second, we can encourage sparse weight matrices using L_1 *normalization*. By adding a penalty term

$$P = L_1 : ||W||_1 = \sum_{ij} |w_{ij}|$$

to the cost function C we get a new cost function $C^\mu = (1-\mu)C + \mu P$, with μ as the regularization strength. Two things happen here: Large weights are discouraged, which is good since they tend to drive the inputs towards saturation of the transfer function where learning is usually slower because the gradient starts to vanish. On the other hand, using the absolute value function for penalizing individual weights, tend to drive one weight towards zero in favor of another weight. For example, the original cost function C may look similar to the one depicted in Figure 1.22 with each line denoting the points of the same value (contour plot) for two arbitrary individual weights w_i and w_j. If we add a penalty term P using the absolute value function this penalty term will overlay a rhombus-like contour and the resulting cost function C^μ will have its minimum closer to the axis, where one of the weights may become zero. And finally, weights that are sufficiently small (i. e. beyond a certain threshold) are considered as not needed and are set to zero (truncation). Hence the zero weights are computationally more efficient than very small weights and they also point out where the relevant structure is. With

their help, we can identify inputs or outputs which are less important for a certain stable gait.

Figure 1.22: Enforcing sparse weight matrices by L1-Normalization. Increasing the penalty term drives weights closer to zero. The red line shows the trajectory of two weights while fading from C to P, it is the $\min C^{\mu}$ for $\mu \in [0, 1]$.

Summary and Conclusion

The proposed controller generalizes the structures with which a wealth of motions for real robots like Myon can be successfully implemented. However, these motions were created in mostly isolated experiments in which only a few limbs moved at the same time, or where the comparatively slow gestures were compensated by stabilizing motions of the legs using the CSLs. Also, simple reactive walking patterns with Myon's torso could be demonstrated, mainly fed by body-initiated oscillation. In all cases, the parameters for this were selected manually and fine-tuned by much observation. The generalized controller now holds the potential not only for these structures but also enables us to use much more inputs and cross-connections between joints. The multitude of new parameters can hardly be determined by hand anymore and must be found with the help of optimization algorithms. Nevertheless, starting with good initial values is still

possible because the known structures can be used almost unchanged. The controller is designed so that learning can take place in principle, but this does not mean that the optimal parameters can be found by any method. In the next few chapters, the point is to show what potential the controller can achieve beyond what has been seen so far.

Discussion

The requirements for the controllers were created from a minimalist point of view with posing the question: What is a general but minimal structure needed to create sufficiently stable and but highly dynamic gaits for legged locomotion? And as we went through this chapter we were able to identify common sub-structures from control applications. But what about biologically inspired modules? Creating neural gait controllers for robots was heavily driven by the results of neuro-biological research of the neural circuits of e. g. stick insects. Biologically inspired reflex loops and simple gait patterns could be successfully analyzed and transferred from nature to robots (for details please refer to von Twickel et al. (2011); von Twickel (2012)). In biological systems, when there is an involuntary, rapid, and similar reaction to a certain stimulus then we usually call this a *reflex*. Such reflexes could be coarsely modeled using comparators, for threshold detection of the stimulus, and using monoflops for the constant playback of the response. Hild et al. (2007) presented a neural one-shot to approximate the monoflop functionality with only two neurons including a timing parameter to adjust the length of the pulse. So for the controller presented here, if we consider additional neurons that are not directly used as motor outputs, we could include handcrafted, stereotyped reactions to certain stimuli, e. g. for retracting the swinging leg.

Possible extensions for the current control structure are manifold. In general, all types of neural architecture elements could be considered, such as recurrent connections and multiple layers, or even non-linear expansions and time embeddings. But the most promising extension, from my point of view, is to add more *context* neurons $\boldsymbol{c} = (c_0, c_1, \dots)^T$ which are not dedicated as motor outputs. This adds additional *memory* to the structure but is neither expected to shrink the current solution space nor does it require changes in the learning system or badly increases computational complexity. The generalized controller with context neurons would be defined as

$$\begin{pmatrix} \boldsymbol{u}_t \\ \boldsymbol{c}_t \end{pmatrix} = f\left(W \begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_{t-1} \\ \boldsymbol{c}_{t-1} \end{pmatrix} \right)$$

as if there would be additional output neurons that are not connected to motors but whose state will serve as input to the next step. Also compliant with the current approach is adding additional inputs which do not directly stem from sensory measurements but rather encode a user-desired behavior such as target speed or direction. Which extension we take depends significantly on the requirements of the target behavior. In the next section, however, it quickly becomes clear that the current controller architecture is already very powerful and can produce a variety of behaviors.

Bibliography

Aström, K. J. and T. Hägelund
1995. *PID-Controllers: Theory, Design, and Tuning.* International Society for Measurement and Control.

Bethge, S.
2014. *ABC-Learning: Ein Lernverfahren zur modellfreien Selbstexploration autonomer Roboter.* Institut für Informatik, Humboldt-Universität zu Berlin. Diplomarbeit.

Doya, K.
1993. Bifurcation of Recurrent Neural Networks in Gradient Descent Learning. *IEEE Transactions on Neural Networks.*

Elman, J. L.
1990. Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

Guckenheimer, J. and P. Holmes
1983. *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields.* New York: Springer.

Hild, M.
2007. *Neurodynamische Module zur Bewegungsteuerung autonomer mobiler Roboter.* PhD thesis, Institut für Informatik, Humboldt-Universität zu Berlin.

Hild, M. and M. Kubisch
2011. Self-Exploration of Autonomous Robots Using Attractor-Based Behavior Control and ABC-Learning. In *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence*, Trondheim, Norway.

Hild, M., M. Kubisch, and D. Göhring
2007. How to Get from Interpolated Keyframes to Neural Attractor Landscapes – and Why. In *Proceedings of 3rd European Conference on Mobile Robots*, Freiburg, Germany.

Hild, M. and F. Pasemann
2007. Self-Adjusting Ring Modules (SARMs) for Flexible Gait Pattern Generation. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI).*

Hild, M., T. Siedel, C. Benckendorff, M. Kubisch, and C. Thiele
2011. Myon: Concepts and Design of a Modular Humanoid Robot Which Can Be Reassembled During Runtime. In *Proceedings of the 14th International Conference*

on Climbing and Walking Robots and the Support Technologies for Mobile Machines, Paris, France.

Kubisch, M., C. Benckendorff, and M. Hild
2011a. Balance Recovery of a Humanoid Robot Using Cognitive Sensorimotor Loops (CSLs). In *Proceedings of the 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Paris, France.

Kubisch, M., B. Werner, and M. Hild
2011b. Using Co-Existing Attractors of a Sensorimotor Loop for the Motion Control of a Humanoid Robot. In *Proceedings of the International Conference on Neural Computation Theory and Applications (NCTA).*

Meier, J.
2015. Entwicklung und Implementierung einer interaktiven Verhaltenssteuerung fur den humanoiden Roboter Myon. Masterarbeit, Freie Universität Berlin, Institut für Informatik.

Pasemann, F., M. Hild, and K. Zahedi
2003. SO(2)-Networks as Neural Oscillators. In *Proceedings of the International Work Conference on Artificial and Natural Neural Networks (IWANN)*, Pp. 144–151.

Strogatz, S.
1994. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering.* Sarat Book House.

Thompson, J. M. T. and H. B. Stewart
1986. *Nonlinear Dynamics and Chaos.* Chichester U.K.: Wiley.

von Twickel, A.
2012. *Embodied Modular Neural Control of Walking in Stick Insects: From Biological Models to Evolutionary Robotics.* PhD thesis, University of Cologne.

von Twickel, A., A. Büschges, and F. Pasemann
2011. Deriving neural network controllers from neuro-biological data: implementation of a single-leg stick insect controller. *Biological Cybernetics*, 104(1-2):95–119.

Werner, B.
2008. Sensomotorische Erzeugung eines Gangmusters für humanoide Roboter. Studienarbeit.

Werner, B.
2013. Entwicklung eines adaptiven sensomotorischen Algorithmus zur dynamischen Bewegungssteuerung autonomer Roboter. Diplomarbeit, Humboldt-Universität zu Berlin, Institut für Informatik.

Wescott, T.
2014. Controlling motors in the presence of friction and backlash. Technical report.