2 Simulated Environment and Robots

Simulation has long been a useful tool in robotics for testing ideas and implementations as impressively demonstrated by Sims (1994). The ability to run robotic processes faster than real-time is a critical factor, especially when it comes to learning algorithms. Simulated worlds are as robust as we need them, so it is possible to test algorithms on robot hardware wear-free and with low costs. With open-source physics engines like the *Open Dynamics Engine*[1] or complete simulation environments like *Gazebo* by Koenig and Howard (2004), it is also possible for anyone to have access to at least simulated robot hardware and to test and compare their algorithms.

There will probably always be a difference between simulation and the real world because the simulation can only be a model of the environment. To steadily improve these models is the mission of physical research, so with increasing precision of both, physics engines and models, the so-called *simulation-to-reality gap* can be made very small. But roboticists should not focus too much on it, but rather adapt the algorithms to robustly deal with a certain degree of model variation; Wear and low manufacturing accuracy can create deviations between two actually identically built robots which are of the same magnitude. Just to name a special candidate, friction is still hard to simulate or compensate for, because the bodies interact at the molecular level and the models for macroscopic objects are no longer exact. Hence, as a very non-linear effect, static friction is not exactly popular in mechanical control engineering.

In every simulation, a trade-off between speed and accuracy must be made. Elaborate models and small simulation step sizes require a lot of computing time and the advantage of being able to run the simulation faster than in real time is quickly consumed by the urge for precision. So the results of simulated systems will always be something different than in the real world, but if we build controllers that do not make explicit model assumptions and work with closed loops, the results can probably be better transferred.

This chapter introduces the simulation environment with which all simulated results of this thesis have been produced. We start with how the robots are modeled and set a special focus on the actuator design, especially when it comes to friction. After that, the robots and their environment used for the simulated learning experiments are presented.

2.1 Simloid

Simloid[2] is a robot simulation environment. It is a lightweight and fast which is crucial for evolution and learning experiments. We must be able to keep the total simulation

[1] Open Dynamics Engine, initial release by Russel Smith 2001: `www.ode.org`
[2] Project initiated by Daniel Hein.

time low, since most experiments will take many trials or iterations to come up with meaningful results. Simloid is written in C/C++ using OpenGL for visualization. The underlying physics is calculated by the Open Dynamics Engine (ODE), which is open source, widely used and well documented.

The version of ODE (0.13) used in this thesis only supports rigid body dynamics and collisions but can not simulate soft materials. However, there are available some tweaks which we could use imitate soft bodies dynamics by adjusting parameters to achieve a spongy behavior for collisions or creating bodies with hundreds of individual parts connected by spring-damper-joints. Nevertheless, within the scope of this work all the simulated robots will consist of rigid bodies only but with softer collision behavior and less strict joint axis constraints.

The simulation step size is fixed to 10 ms, i. e. 100 Hz which has turned out to be a good compromise between acceptable numerical accuracy of the physics and moderate computation speed. To further increase simulation speed we could switch off all kinds of visual features, like shadows or textures, or completely start the simulator in command line mode and run it on a dedicated machine.

Simloid has a TCP/IP interface to read status messages and send control commands to the robots. Also, the robot's characteristics are transmitted with this interface and the user program can set specific environmental settings or reset, safe and restore simulation states. Since the simulator is decoupled from the user's control program it is possible to start multiple instances of the simulation on different remote machines and run several simulations in parallel, e. g. for evolution experiments where trials with different individuals from the current generation can be evaluated separately.

Before each simulation step the user program sends its new control commands, then the simulation step is executed, and afterward the current sensory and environmental state is being transmitted. The user program is only expected to send new control commands if needed. The protocol allows for a thin communication channel. New commands for all or individual motors can be send, or even a simple `DONE` will suffice to instruct the simulator to execute the next step. If necessary, an *interlaced mode* can be enabled that makes the simulator execute the motor commands *while* the user program is calculating. This brings the simulation closer to the behaviour of a real machine and is useful to bridge the reality gap which will be discussed in chapter 6 on page 120.

Even though ODE provides versatile options for creating joints, only the so-called *hinge* (or sometimes *revolute*) joint-type is currently used for the construction of all robot parts. This is for two reasons: First, many types of joints needed for legged robots can be assembled as a combination of hinge joints which fairly reduces complexities in the implementation and testing of the simulator. But the second and more significant reason is, that using hinge joints only, keeps us closer to practice. Most of today's legged robots are constructed in a similar way. For instance, creating a ball joint with all three axes going to exactly the same point needs way more effort to fabricate than having three separate joints with slightly displaced axes. Commonly, a single type of servo motor is used throughout the robot's body and actuated joints are often just a combination of multiple motors.

Ground and body friction is approximated as Coulomb friction. The level of friction

can be adjusted by defining specific material-to-material coefficients. There is no air friction implemented, since the velocities achieved by the legged robots are currently too small to produce a significant amount of friction force, so for now it will be neglected here. Despite the simplicity of the ground friction model, within the scope of this work, there was implemented a more sophisticated joint friction model because the actuated joints will presumably have the most significant influence to the resultant robot motion. This model will be described in detail in section 2.3.

Gravity is usually treated as a homogeneous force field with the accelerating force $9.81m/s^2$ but can be changed by the user program. In order to increase the robustness of walking gaits against external disturbances, it is possible for the user program to instruct the simulator to apply forces to specific robot body parts. With this method, we can easily create scenarios in which the robot gets specifically disturbed with impulses, e. g. while trying to walk.

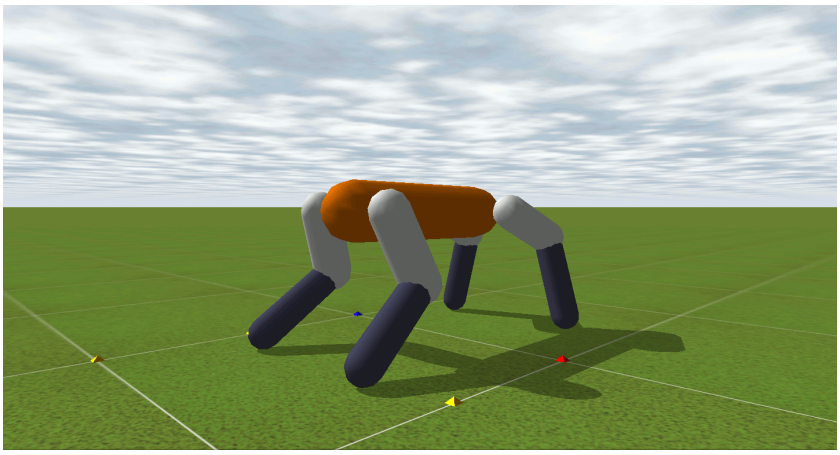

Figure 2.1: Illustration of the Simloid simulation environment.

2.2 Robot model

The simulated robots used throughout this work consist of simple body segments and as said, hinge joints only. Each of these joints is actuated by a simulated servo motor. For simplicity, everything which is not necessary for walking is omitted. These simulated robots have no counterpart in the real world and are not specifically designed to be actually built. But their simplicity applies to body design only. The components, such as the motors or sensors which take the major effect within the sensorimotor loop are simulated quite realistically.

2.2.1 Body Model

The robot's body can be created from simple segments connected by actuated joints. To keep the simulation time low, there were only used basic symmetrical shapes, e. g.

cuboids or capsules. To give them weight, there is the possibility to either set each segment's mass or density. The robot segments always all have equally distributed mass and the center of mass always lies in the center of the geometry. Each segment can be given individual surface properties, e. g. to have increased friction between the robot's feet and the ground. Furthermore, collision checking can be disabled to allow for placing segments with partially or fully overlapping regions.

2.2.2 Joint Model

The joint model is kept simple as all actuated joints are hinges. The maximum range for each joint is $[-\pi, \pi]$. Every joint has configurable end stops which can be used to constrain the motion of the limbs. If the lower limit is set to $-\pi$ while the upper limit is set to π, the end stops are deactivated and the joint can run freely (e. g. for a pendulum). A joint always connects exactly two bodies and its center of rotation can be positioned arbitrarily, i. e. it must not necessarily be located within a body. To further simplify, all joints are either a pitch, yaw or roll joint. This means that all joint rotation axes are either parallel or perpendicular to each other. However, this restriction seems to be commonly used in robotics since it lowers the modeling, implementation and construction efforts a lot. The two bodies connected by a joint do never collide and are only constrained by the predefined stops or by collisions with either other (indirectly connected) bodies or the ground plane.

2.2.3 Angle and Velocity Sensor Models

Each joint is equipped with position and velocity sensors, measuring the current joint angle and angular velocity. The angle φ with range $[-\pi, \pi]$ is mapped to $[-1, 1]$. The angular velocity ω is normalized by a maximal value of 12 rad/s. The configured joint stops further constrain the value range of the position sensor. When using free run, we should have in mind that the angle has a discontinuity when moving from 1 to -1 and vice versa. This has to be taken into account when conducting experiments with freely running joints, e. g. pendulum experiments. To stay closer to reality the maximal resolution of the sensors is intentionally limited to 16 bit and can even be reduced down to 10 bit. Additionally, the sensors are made noisy by adding Gaussian noise with a standard deviation of 4 bit. This means, that there is a certain, yet acceptable loss of information which the controllers and learning algorithms have to deal with.

2.2.4 Acceleration Sensor Model

For each body segment, there is the possibility to attach a 3-dimensional accelerometer with the sensor's axes aligned to the segment's main axes. These sensors measure the acceleration forces influencing this body including gravity. The 3-dimensional output values of these sensors are restricted to $[-8g, 8g]$ and mapped to $[-1, 1]$ each. As with the position and velocity sensors, the resolution is artificially decreased to 16 bit and Gaussian noise is added with a standard deviation of 4 bit. The orange color of a robot's body in simulation figures denotes an attached accelerometer.

Real accelerometers are usually implemented by measuring the deflection of a proof mass which is attached to a reference frame with a spring-damper-system (see Figure 2.2). For instance, there are implementations (e. g. MEMS) where this proof mass forms a capacitor and the acceleration of the outer frame is transformed into a capacitance change (in turn then measured as a voltage change). The resulting output value is an estimation of the negative spring-reaction force $F = k\Delta x$. The damping of such a system should be close to critical, i. e. $\eta = 2\sqrt{km}$, to avoid unwanted oscillations.

For the implementation of the virtual accelerometers within Simloid, we create an acceleration value $\ddot{\boldsymbol{x}} = \frac{d}{dt}\dot{\boldsymbol{x}}$ from the corresponding body's velocity where the sensor is attached to. The velocity is provided by the simulation and we subtract the vector of gravity to get our sensor's output signal $\boldsymbol{r} = \ddot{\boldsymbol{x}} - \boldsymbol{g}$. Since when at rest, a real accelerometer measures the gravity, while when in free fall, it measures zero acceleration.

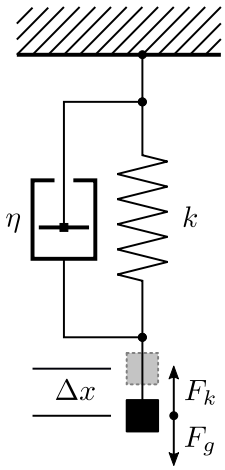

Figure 2.2: Simplified structure of a 1D accelerometer. The motion of the outer frame yields a deflection of the proof mass m, the sensor's output is proportional to the measured displacement Δx.

2.2.5 Electromechanical Motor Model

The robot's actuators are simulated electrical drives for which the input signals are conceived as pure motor voltages. It is considered a simulated gear motor. There is no additional internal control loop, no position or velocity controller. The motor's input, denoted as u, has a valid range of $[-1, 1]$ which is internally mapped to $\pm U_s$, the supply voltage.

The electrical characteristics are simplified in the sense, that some properties are neglected by the model, others are subsumed. For instance, the gear reduction is included in the motor constants and the motor's inner friction is subsumed in the joint friction model of the next section. Temperature effects are neglected fully. The basis for the parameter choice is the Robotis RX-28 smart servo. This versatile servo motor was used for all joints of the Humanoid Robot Myon by Hild et al. (2011). My choice of utilizing the properties of this servo is therefore based on personal work experience while creating motion control loops with this robot. Fortunately, Bethge (2014) already created a detailed model, inferred parameters and proposed an implementation and on whose results the used model is based.

A DC motor is usually characterized by the speed constant k_n and the torque constant k_M, the coil's resistance R and the supply voltage U_s. The inductance L would have a low pass filter effect to the current, $U_L = L\, dI/dt$, but is neglected since the simulation step size of $\Delta t = 10\, ms$ is too large to make this effect visible. Also, DC motors are usually driven by PWM outputs of H-bridges with frequencies in the kHz domain. Rather the voltage is assumed as constant for the time Δt in this model.

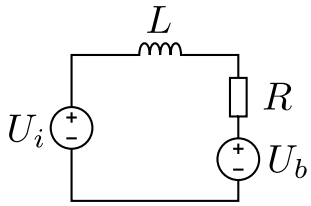

Figure 2.3: Equivalent electrical schematics of a DC motor with resistance R, inductance L, and back EMF voltage U_b.

The motor model is given as

$$U_i = U_s\, u \tag{2.1}$$
$$U_b = \omega / k_n \tag{2.2}$$
$$I = (U_i - U_b)/R \tag{2.3}$$
$$M = k_M I \tag{2.4}$$

which transforms the input signal u and the rotational speed ω into output torque M with current I, back EMF voltage U_b and. The model has no internal state.

symbol	meaning	preset
U_s	supply voltage	16 V
k_n	speed constant	0.4774 rad/Vs
k_M	torque constant	1.393 Nm/A
R	motor coil resistance at $65°C$	$9.59\,\Omega$

Table 2.1: Motor parameters determined by data sheets and measurements conducted by Bethge (2014). The gear reduction ratio of the servo is 1:195 with efficiency 62.5% which is already included in k_n and k_M.

The only mechanical component being modeled currently is the gear transformation with the given ratio of $G = 1 : 195$. Backlash and gear deformation were not modeled since the RX-28 uses metal gears with very low backlash and deformation. However, when using motors with plastic gears, deformation becomes strongly influential and cannot be neglected anymore, as I needed to learn in Kubisch (2008). A good overview and special implementation tips on modeling and handling the effects of backlash and friction in control systems is provided by Wescott (2014).

Even though there is a lot of friction in the gears and some in the ball bearings, the friction model is not considered as part of the actuator model but instead regarded a joint property. This way the friction model can also be applied to non-actuated joints. The next section describes a more elaborate friction model which I use to simulate the robots' joint friction.

2.3 Joint Friction Model

The actuators are the robot's interface to the world and naturally rely on friction since we have many mechanically moving parts such as bearings and gears. People applying

motor control loops to real machines inevitably discover the difficulties of the highly non-linear effects of friction, especially when using low-cost gear motors. Probably the most bothering friction effect is static friction or simply *stiction* which affects the domain of very low velocities and direction changes. Hence, it is crucial to incorporate an appropriate friction model in our simulations of walking robots to avoid that the results are based on too idealized actuators. If the results from simulation shall be applicable to real machines, we must also make efforts for convincing friction modeling. However, it turned out that current physics simulations often use very simple approximations and if we need a more realistic model we mostly have to implement it ourselves. Therefore, I took the—from my perspective—most promising modern friction model and made it work for the ODE-based Simloid environment. Since during this process the model and the approach of implementation changed, so I gave it a new name, but the original excellent idea is still kept.

Different Types of Friction

Friction has several clearly distinguishable effects. When two objects are moving with non-zero relative velocity and their surfaces are in contact, there is a force that acts against this motion. If the properties of the surface and the velocity are constant then this force is also constant and is called *dry* or *Coulomb friction*. When the surfaces are lubricated or when we consider objects moving through liquids, this friction force becomes velocity dependent and usually increases with higher velocity. This is called *fluid friction*. In the domain of very low velocities, we have to deal with so-called *stiction*, i. e. a notable increase in friction force near zero velocity. And when the motion has fully stopped the friction force compensates for all external forces until a certain threshold is met, which is called the *break-away force*. If the external forces are larger than this threshold the motion begins abruptly and the friction force quickly decreases. It showed up, that when the bodies at rest are subject to external forces lower than the break-away force, they actually move a little but would return to its original position if the external forces disappear. They act similar to bodies connected with little springs and the deflection of their position is called the *pre-sliding displacement*. Stiction along with the break-away is usually the most significant effect of friction. It can easily be discovered in everyday life but the more subtle effect is the pre-sliding displacement. For an extensive overview of friction modeling for control applications, I recommend reading Olsson (1996).

2.3.1 Friction Models

Currently, we must accept that friction can only be modeled approximately. There are various models available from the literature covering most of the experimental observations. But there does not seem to exist a general model, so the choice is left to the experimenter to pick one out of the many and make it work for the desired application. Olsson et al. (1998) state it as follows:

> There are many different mechanisms. To construct a general friction model from physical first principles is simply not possible. Approximate mod-

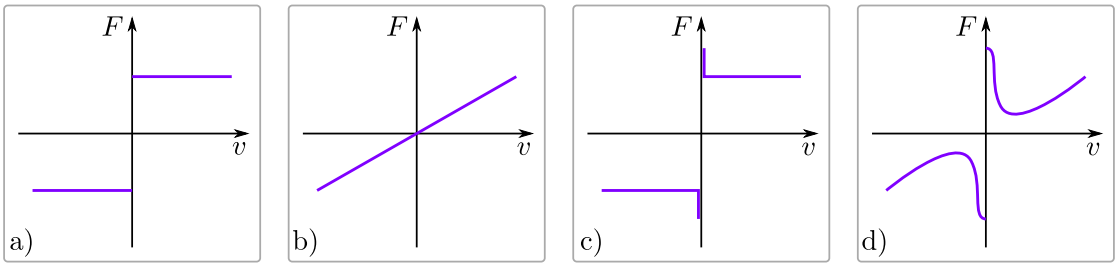

Figure 2.4: Static friction models: a) Coulomb (Dry), b) Fluid, c) Coulomb with stiction, and d) Stribeck's Curve: A combination of the effects of Coulomb, fluid and static friction.

els exist for certain configurations. What we look for instead is a general friction model for control applications, including friction phenomena observed in those systems.

Static Models

Fortunately, nowadays friction models can already deal with the most crucial effects observed in experiments but the underlying mechanics of the surfaces' particles are far too manifold and non-linear to be captured by a single equation. The simplest form of modeling friction is with static models. Such models do not have an internal state, instead, they are merely non-linear functions of velocity, $F(v)$. This is the relative velocity of the concerned bodies at the contact point. Static models are usually described by the so-called Stribeck's Curve as published by Stribeck (1902), and which is depicted in Figure 2.4 d). This experimental observation is a combination of the constant Coulomb friction (a), linear (to quadratic) fluid friction (b), and increased static friction (c) near zero velocity.

For use cases where we have to deal with steady velocities only, such static models are sufficient. But when using static models for dynamic motions, i. e. motions with a lot of changes in speed and direction, static models become inaccurate and numerically problematic. The problem is that we need to *detect* when the velocity is zero. Due to the discontinuity of the friction force at zero velocity, this can easily lead to oscillations. For near zero velocity, we need to know the external force F_{ext}, either measured or provided by the simulation. Consider the bodies are at rest, i. e. velocity is zero. Stribeck's curve is not defined in that point. But the friction model must react with a force F_R with the exact absolute value but the different sign in order to keep the forces equal, $F_{ext} = -F_R$. Otherwise, the bodies would never really rest. For some simulation environments, the external force may not be easily available. Approximating the external forces from the next time step's change in velocity is also not a good idea. A small time difference between action and reaction may produce unintended oscillations when dealing with larger restoring forces. Ideally, we would use models which do depend on the current velocity only.

Dynamic Models

Dynamic friction models have an internal state, or memory so to say. That means that they behave differently with respect to the current history of inputs, i. e. the changes in velocity. A promising modern dynamic friction model is the LuGre model by Canudas De Wit et al. (1995); Olsson (1996) and Olsson et al. (1998), which covers most of the static and dynamic effects caused by friction, such as dry, fluid, and static friction, as well as the sudden break-away, and pre-sliding displacement.

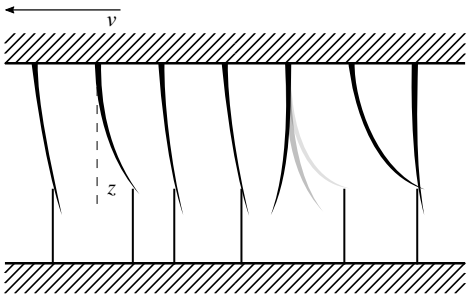

Figure 2.5: Simplified illustration of the LuGre friction model. Consider virtual bristles which are deflected when the surfaces move relative to each other with velocity v. An internal state, z, represents the averaged deflection of the bristles. Friction is conceived as a non-linear restoring force of that deflection.

The LuGre model is defined as

$$\frac{dz}{dt} = \dot{z} = v - \frac{|v|}{g(v)} z \tag{2.5}$$

$$g(v) = \frac{1}{\sigma_0} \left(F_C + (F_S - F_C) e^{-(v/v_S)^2} \right) \tag{2.6}$$

$$F = \sigma_0 z + \sigma_1(v)\dot{z} + \sigma_2(v) \tag{2.7}$$

where friction F is a function of velocity v, and the bristle deflection z and $\dot{z}$. The resulting friction force (2.7) depends on the velocity and this internal state z (the averaged bristles' deflection), is the sum of the parts for stiction and the fluid friction.

Parameters are $g(v)$, $\sigma_1(v)$ and $\sigma_2(v)$. The function $g(v)$ (2.6) is kind of a Stribeck Kernel and regulates the transition from stiction to Coulomb friction and the functions $\sigma_1(v) = \sigma_1 e^{-(v/v_d)^2}$ and $\sigma_2(v) = F_v |v|^{\delta_v} sgn(v)$ are the stiffness and damping of the bristles. The differential equation (2.5) forms the core of the model; it is an integrator which has the current velocity as input and accumulates the estimate of the deflection.

Problems with Dynamic Models

Admittedly, dynamic models such as LuGre bring other difficulties: They usually need a lot of parameters to be determined, or they need an additional discretization step when using these continuous time models in the discrete time domain. But most crucial, there are inherent problems with force-based approaches. High friction needs high restoring forces. In combination with lightweight objects and discrete time simulations, we have

another potential oscillation problem here. In previous work, Kubisch (2008), I extensively studied this model and found it not trivial to apply for my application. Ideally, a friction model should not output a force but a target velocity and we should separate the problem by using a velocity controller which is stable for the given simulation step size.

For walking robots, the joints' velocities will change with each step the robot makes. Also when the robot is just standing and merely performing balance recovery, zero crossings and low velocities are the general cases and steady velocities are mostly exceptional. In short, we have the need for dynamic friction models here. The next section presents a simple dynamic friction model for discrete time simulations with large step sizes which I use for the joint motor friction modeling within Simloid. The main reason for me to create my own model was, that I discovered oscillation problems with *force-based* approaches, when using higher restoring forces and lightweight objects. This problem further increases with larger simulation step sizes. So I considered a *target velocity* approach to circumvent stability issues. For the new model, I integrated the most important features learned from the LuGre dynamic model, but used a different approach to apply the friction force, in order to avoid stability problems right from the beginning.

2.3.2 The BRUSH Friction Model

The Brush model is an easy to implement and discrete time dynamical friction model which is inspired by the LuGre friction model. As the LuGre model, it is based on the idea, to simulate the deflection of virtual bristles which are moved over an abrasive surface. These bristles are flexible and will bend when there is a relative movement of the surfaces. If the bristles are deflected they apply a restoring force that works against the cause of the movement. The internal state variable z denotes the current average deflection of the bristles. Many bristles may belong to the same brush, therefore the name.

Dynamic friction models usually only take the relative velocity as input. In order to calculate the deflection of the bristles, we need to integrate that velocity. Hence, the internal state forms an integrator and we get a relative position, i. e. the bristles' average deflection. This integrator has a hard limiter, so the bristles can only be deflected to a certain degree which is denoted as the maximal bristle deflection σ. The discrete time update equations for the Brush model are given as

$$z(t) = f\left(z(t-1) - v(t), \sigma\right) \tag{2.8}$$

$$\overline{v}(t) = k_B z(t) \tag{2.9}$$

with the input velocity v and the output velocity $\overline{v}$ and f being the clipping function which constrains $z \in [-\sigma, \sigma]$.

The bristles behave like non-linear springs; when deflected they work against the cause of their deflection. A factor k_B defines their stiffness. The Brush model is by design not based on forces. It outputs a desired target velocity. So rather than directly applying restoring forces, we instead use a velocity controller

$$u(t) = C\left(\overline{v}(t) - v(t), S(v)\right) \tag{2.10}$$

with $C : \mathbb{R}^2 \to \mathbb{R}$. The controller takes the difference in velocities (i. e. the error signal) as its first argument and tries to bring that difference down to zero using at most the force defined by the function $S(v)$ as the second argument (see Figure 2.11).

Using a controller is in general equivalent to applying the force directly but allows a smoother application over several timesteps with certain compliance in order to meet the stability affordances of discrete time simulations. When exerting spring reaction forces directly to bodies with small masses, we easily encounter numerical stability problems in simulations with comparably large step sizes, e. g. $dt = 10^{-2}$. The velocity controller helps to stabilize since it separates the problem of applying the force from the friction model towards well-understood structures such as PID controllers. Also, parameters for the controller could be adjusted separately. Furthermore, many physics engines already provide stable velocity controllers, so does the Open Dynamics Engine. So in some cases, we could already benefit from that. However, in cases where no controller can be taken out-of-the-box, we could implement such a controller as described in section 1.2.1.

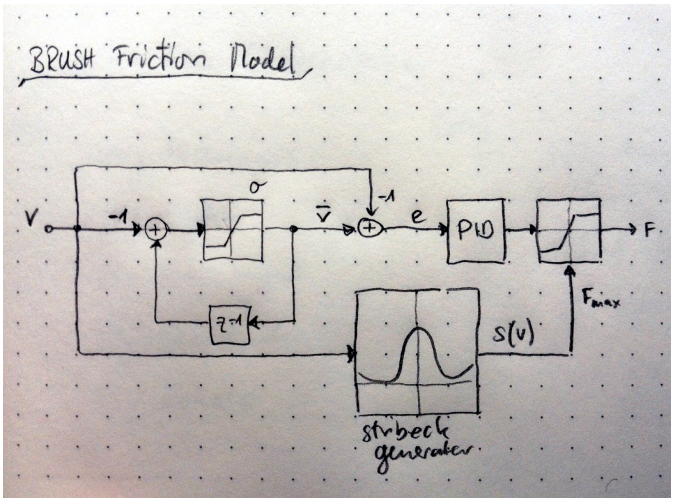

Figure 2.6: Schematics of the BRUSH friction model.

Stribeck Generator

So far we saw that the deflection of the bristles invokes a restoring velocity. The maximal force which can be used by the controller to reach the set-point, i. e. the restoring velocity, is determined by the function

$$S(v) = F_c + (F_s - F_c)\, e^{-(v/v_s)^2} + F_v |v|^\delta \tag{2.11}$$

which has a similar shape as Stribeck's curve but is defined over the full velocity range. The only argument v is the relative velocity of the surfaces in contact. Stribeck's curve illustrates friction for constant, steady velocity. Independently of the parametrization, its characteristic shape is valid for a broad palette of materials. With respecting $F_s \geq F_c$, the equation 2.11 can be read as: If the input velocity is in stiction range v_s (so-called *Stribeck velocity*), use stiction F_s, otherwise use Coulomb friction F_c. Nonetheless, the

transition between sticking and sliding and vice versa is continuous. Independently from that, we always add a velocity dependent term, which mostly affects domains of higher velocity. The shape of the fluid friction term can be manipulated with the exponent δ to get an either linearly or non-linearly increasing friction term.

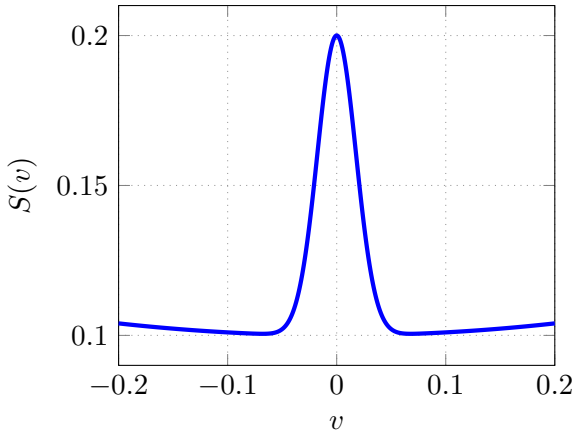

Figure 2.7
The Stribeck generator used for the Brush Friction Model. This function regulates the transition from stiction domain to Coulomb (dry) friction and fluid friction for higher velocities. In comparison to Stribeck's Curve it is symmetrical $S(v) = S(-v)$ and is defined over the full velocity range.

Parameters

Like for all friction models, the parameters needed may be difficult to determine. All parameters needed for the Stribeck function can be estimated by steady-state velocity experiments. However, carefully conducting these experiments may take time and need special equipment and some parameters as the maximal bristle deflection and the bristles' stiffness may be hard to measure at all. Thus, for our motor model, we use a machine learning approach. We collect enough representative data from hardware experiments and compare it to our simulation with a cost function. With manual tuning, we bring the parameters in an already close starting distribution and define limits for each. After that a stochastic search, e. g. a genetic algorithm is applied in order to fit the parameters (see subsection 6.1.2).

σ	max. deflection	F_c	Coloumb friction coefficient
k_B	bristle stiffness	F_s	stiction coefficient
v_s	stiction range	F_v	fluid friction coefficient
δ	viscosity exponent		

Table 2.2: Friction parameters for the Brush joint friction model including Stribeck's curve parameter.

Experimental Results

To qualitatively demonstrate the capabilities of the model we need to have a proof mass which is assumed to be subject to friction and exert an external force. The results of the following experiment are depicted in figure 2.8 and show the object's state, the

external forces, the acting friction forces and the internal state of the bristles. The state of the proof mass is (x, v) and the object rests in the beginning. There is a continuously increasing external force exerted to the mass and, due to stiction, the object still rests, but the internal state of the friction model accumulates. Actually, the object moves a little (the aforementioned pre-sliding displacement) but does not gain a higher velocity. After 500 ms the external force disappears and the object returns to its original position. Then at $t = 1s$ the force starts increasing again and within the next second, the internal friction state (the bristle deflection) has reached its maximal value and can not further *compensate* for the external force. So the proof mass slowly starts to move and since friction rapidly drops when velocity is increasing according to $S(v)$ the object begins to slip. During this transition, the friction applied drops quickly from the stiction to the Coulomb domain. Finally, when the external force drops again the proof mass slows down accordingly, friction raises to the stiction level, the internal state z rewinds and eventually the object comes to rest. But although the external force does not vanish completely the objects still rest. The force is fully compensated by the bristles and the mass sticks.

Summary and Conclusion

To summarize the features of the here developed Brush friction model, it can be said, that with comparably little efforts in implementation we gain a convincing sticking behavior without the need to access the exact external forces from simulation and without the need to determine zero velocity. The model supports break-away force and non-leaky resting as well as the more subtle effects of pre-sliding displacement and returning to the original position if the virtual bristles did not fully reach their maximal deflection. As a bottom line, the friction force for steady velocities is as described by Stribeck (1902). We split up the model into two parts, the first is concerned with the friction and creates a target velocity and the second is a velocity controller part which takes this target velocity as set-point and tries to keep the difference between current velocity and target velocity as small as possible under the constraint of the maximal force given by a non-linear function of velocity. The controller can be developed with standard techniques such as PID. With this *trick*, we keep the simulation out of the loop of our friction model. The model inputs and outputs only velocity signals. It is now easier to implement a stable system.

2.4 Simulated Robots

Within the scope of this work, I created a couple of simulated robots with different numbers of legs and joints. The goal was to provide a variety of platforms for testing gait controllers and learning algorithms. For instance, the generalized controller architecture defined in section 1.4 is designed to be independent of the number and orientation of the robot's joints, the lengths of the limbs, the mass distribution and so forth. This also holds for each part of the learning architecture: Neither the state space segmentation nor the state-action learner have dependencies on the robot's specific morphology. My hypothesis is, that we will probably find a powerful learning architecture when we start testing

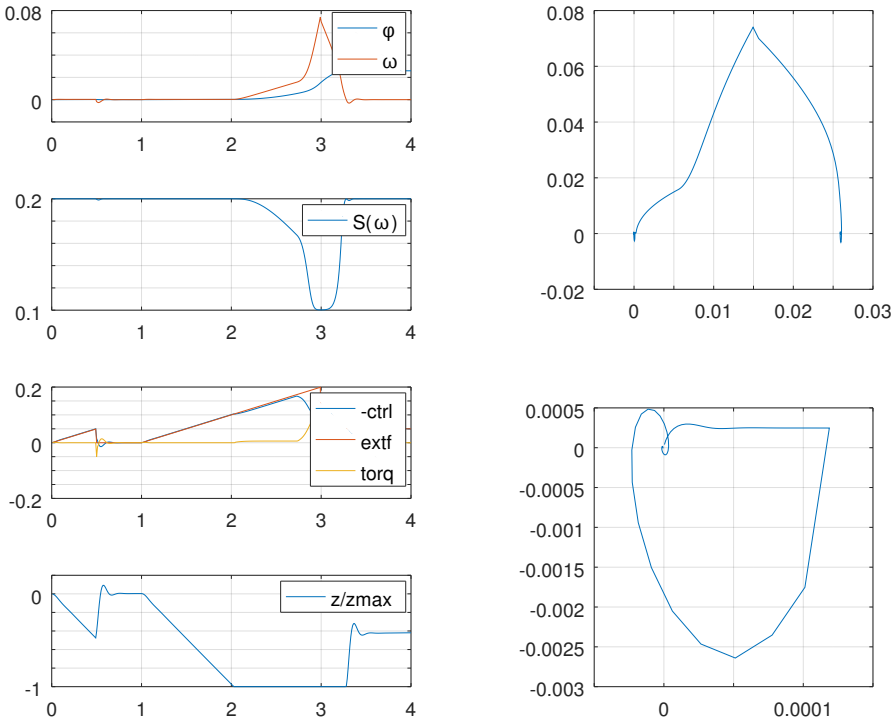

Figure 2.8: Experiments show the effects of stiction and break-away as well as pre-sliding displacement. Parameters of the model: $\sigma = 0.025$, $k_B = 1$, $F_c = 0.1$, $F_v = 0.05$, $F_s = 0.2$, $v_s = 0.025$, $\delta = 2$. Please note: The given parameters are choosen for demonstration purpose, in order to maximally visualize the effects covered by the model, and are not necessarily valid for real hardware.

algorithms with different tasks and on multiple morphologies right from the beginning, to prevent the introduction of unintended dependencies to specific construction details. The kind of control strategies needed for solving walking tasks may completely differ from the ones needed to solve common example problems such as the pendulum swing-up or pole-on-a-cart problem. Hence, we need to find an abstraction level for describing the goals, which incorporate easy and complex tasks at the same time. The only assumptions, which I wanted to state clearly, are that the robots created within this thesis rely on the actuation concept of revolute joints and electrical motors driving them, and that the robot's torso and limbs are made of rigid bodies. The algorithms may work fine with lots of other actuation types and with soft body dynamics as well, but for the sake of clarity, and since such morphologies were not explicitly tested within this thesis, we should expect at least some difficulties and the need for further research.

During the development of the robots, some of them were created with initial flaws, e. g. with improper joint configurations which significantly limited the robot's capabilities to

walk. Systematic studies for optimizing robot morphology in order to maximize certain goals like walking speed or energy efficiency are not considered as part of this work. So all robots finally selected and described here, were developed through a short series of trials and errors in order to remove the most crucial design mistakes. In order to reduce the dependencies of the mechanical design, the constructions of the robots are kept as simple as it was possible. Even though the learning algorithms are trying to get the best result out of any construction to maximize a certain goal like walking speed, we should not expect the algorithm to compensate for badly designed morphologies. The robot's design can, however, profit by the detailed knowledge of the control architecture in order to ease the search for good parameters.

With all this in mind, the following series of robots, as different as they might seem at first glance, provide a good basis for successively increasing the complexity in manageable steps. From one robot to the next, the number of joints is increased. This is worth mentioning since most algorithms directly rely on the number of sensory inputs. Also, the robot is able to move more versatile with increased degrees of freedom. The simplest systems are either fixed or can only crawl on the ground, so stability is not an issue. With increased complexity, though, the robots get enabled to actually walk with their limbs being usable legs. Finally, the humanoid robot is expected to walk on only two legs, which further increases the problem of keeping balance during walking. It must also be noted that the action scope dramatically increases with the ability to walk because with walking instead of crawling, the robot can gain significantly higher moving speeds and is enabled to step over small obstacles.

But with increased complexity in morphology, the analysis of the acquired controller parameters and behaviors gets harder. It begins with proper visualizing the data, as all the interesting data for a 2-DOF learning robot can still be displayed meaningfully on a single screen. This is sheer impossible for the humanoid robot with hundreds of controller weights and sensory values. But most crucial, searching good controller parameters from completely random initial distributions is mostly not a fruitful approach. In order to successfully find good parameters for fast walking, the learning algorithms should at least start with well-initialized parameters. For the crawling (and even for the four-legged) robot this task can still be managed by an experienced experimenter, whereas finding proper initial values for the humanoid can be in itself very challenging.

The following sections are organized as follows: It begins with a simple Pendulum in subsection 2.4.1, describing a test system that intentionally cannot walk but which is set up to get a basic feeling for the interdependencies between the different algorithmic components. Sections 2.4.2 and 2.4.3 introduce robots with only a few joints but which are already able to crawl on the ground. The last two sections, 2.4.4 and 2.4.5, describe more elaborate robots with up to 22 joints which are able to walk and run with versatile gaits. All robots presented in this chapter are physically simulated within the framework introduced in this chapter so far.

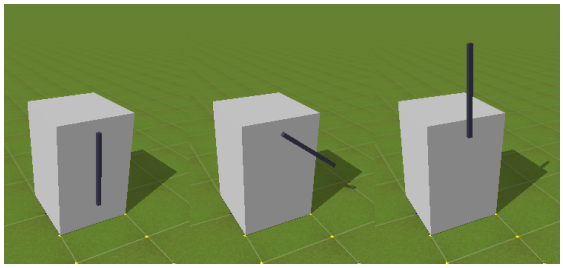

Figure 2.9: A physically simulated pendulum.

2.4.1 Pendulum

The simplest system under test used for the experiments in this thesis is a physically simulated pendulum as depicted in Figure 2.9. This system has only one degree of freedom and will probably never walk, but is suitable for testing the functionality of the applied learning algorithms on a small scale. Indeed, for such a simple system, there are already rich dynamics and a lot of behaviors can be differentiated. Using simple systems like this to start with is crucial for studying the interdependencies of the algorithmic components and furthermore to manually adjust the free parameters which are needed for successfully applying optimizers. Moreover, finding good initial values for self-adjusting parameters is much easier when the system under test has low complexity. And finally, a pendulum can be regarded as a single-joint robot.

Since for experiments with legged robots, behaviors such as walking in different directions and gaits or standing up and recovering balance are of major interest. For a simple pendulum, an equivalent set of expected behaviors must be defined to see, if, in principle, the algorithms work and produce the anticipated behaviors. Since a pendulum has only a single joint and actions are limited, I choose three different tasks for the experiments: *1)* swing-up with restricted torque, *2)* return to resting position, and *3)* intrinsically motivated learning. In order to achieve a comparable moving speed and swinging frequency, we view the pendulum as a reduced leg with only one joint and choose the mass m and length l of the lever as well as the maximal torque accordingly (Table 2.3)).

The vector (φ, ω) defines the state of the system which is the deflection angle φ and angular velocity $\omega = \dot{\varphi}$. The angle is not restricted by limits and so the lever can rotate freely, but the velocity is limited by friction and the maximal torque applied. The state changes according to the torque M applied to the system which is the sum of gravity acting on the lever, joint friction (according to subsection 2.3.2) and the torque exerted by the motor according to the control input.

Mass	Length	Eigenfrequency
$m = 0.47\,kg$	$l = 0.5\,m$	$w_d \approx 1.5\,Hz$

Table 2.3: Pendulum parameters.

2.4.2 Crawler

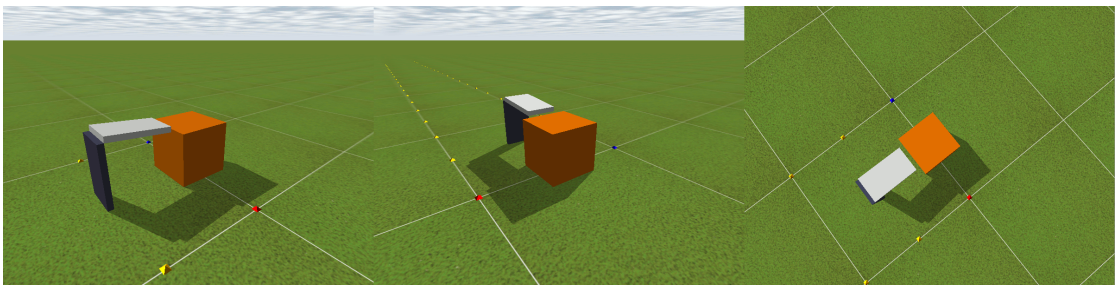

Figure 2.10: The Crawler robot from different perspectives.

The *Crawler* is the simplest of the systems worth calling it a robot. It is actually a torso equipped with a 2-DoF arm, with the help of which the robot can just *crawl*. Since the torso and the arm are made of boxes only, the movement abilities of that robot are strictly limited to crawling forward and backward. However, with applying large torques to the joints it is nonetheless possible that this robot performs small jumps. Furthermore, it can happen, that this robot performs actions that result in finally laying on the back or side. Also, it may happen that Crawler gets stuck with its arm clamped under the torso and is not able to move anymore. (In the conclusion of this chapter, in section 2.4.5, it is discussed how to handle or avoid such terminal states.) This robot actually slides over the ground with its body, so the friction forces acting on the ground-touching part of the arm must be higher than the friction forces acting on the torso in order to slide easily.

Name	Axis	Joint Limits		Def.
Elbow/Shoulder Pitch	x	$-60°$	$150°$	$0°$

Table 2.4: Joint ranges for the Crawler robot.

2.4.3 Tadpole

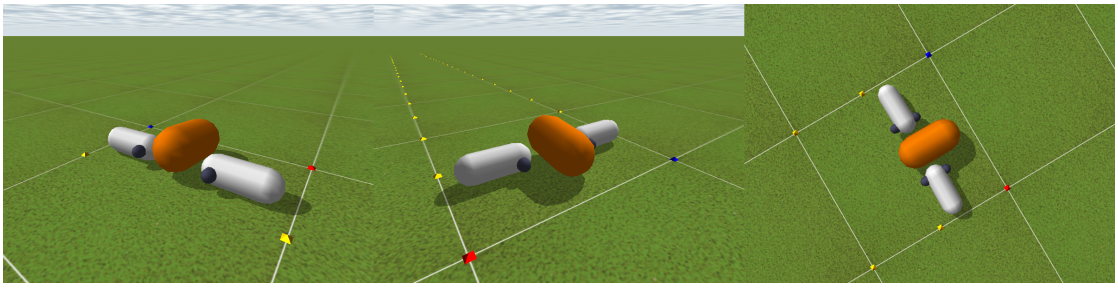

Figure 2.11: The Tadpole robot from different perspectives.

Even though this robot called *Tadpole* is still kept simple with only four joints, it is already more robot-like. The shapes are rounded which introduces the problem of balance recovery. It has two arms with two joints each. This robot is in principle able to move everywhere if the ground is flat and if there are no obstacles. It can move forward, backward, and can turn on the spot. Furthermore, combinations of such behaviors such as moving in circles will be possible. Therefore we can expect from this robot to move forward, moving backward, turn left and turn right. Also, the tadpole may accidentally fall on its back and will probably not be able to roll back to the initial position on its own. Likewise the Crawler, Tadpole also has increased friction on the arms and reduced friction on its torso to facilitate sliding over the ground.

Name	Axis	Joint Limits		Def.
L/R Shoulder Roll	y	$-90°$	$90°$	$-10°$
L/R Shoulder Yaw	z	$-90°$	$90°$	$0°$

Table 2.5: Joint ranges for the Tadpole robot.

2.4.4 Quadruped

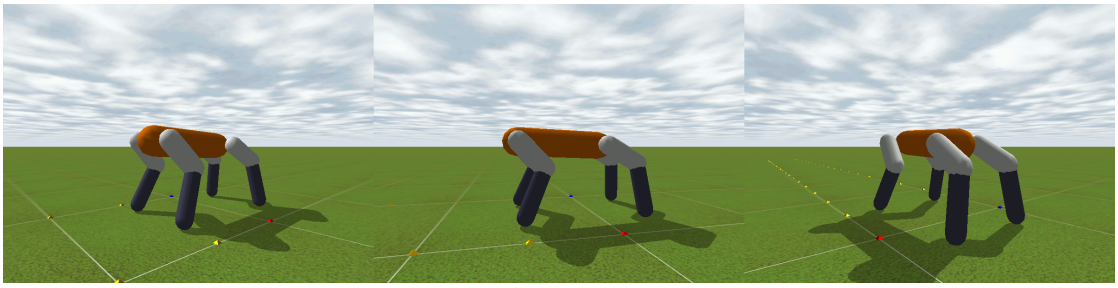

Figure 2.12: The four-legged robot from different perspectives.

With its 12 degrees of freedom, the four-legged robot is already complicated enough to demonstrate the capabilities of the algorithms but is not too complex for good access to the analysis of the results. Also, it is the simplest of the mentioned robots which actually walks. This robot can move around freely with different gaits and speeds, and it is able to turn in place and move sidewards. It can run comparatively fast and make large steps with all four legs lifted from the ground simultaneously. All twelve joints of the four-legged are arranged in the simple perpendicular scheme in order to keep the results comparable. As the Crawler and the Tadpole, the four-legged robot can also fall on its back but is able to recover and turn back to the initial position. It can lay down, stand-up and roll over. Due to its good tradeoff between versatility and simplicity, it plays a central role in this thesis since most of the experiments were conducted with the four-legged robot.

Name	Axis	Joint Limits		Def.
L/R Shoulder Pitch	x	−90°	90°	−30°
L/R Hip Pitch	x	−90°	90°	−60°
L/R Shoulder/Hip Roll	y	−90°	90°	10°
L/R Elbow/Knee Pitch	x	0°	120°	60°

Table 2.6: Joint ranges for the four-legged robot. Left and right joints come always in pairs and are symmetrically arranged to the sagittal plane.

2.4.5 Humanoid

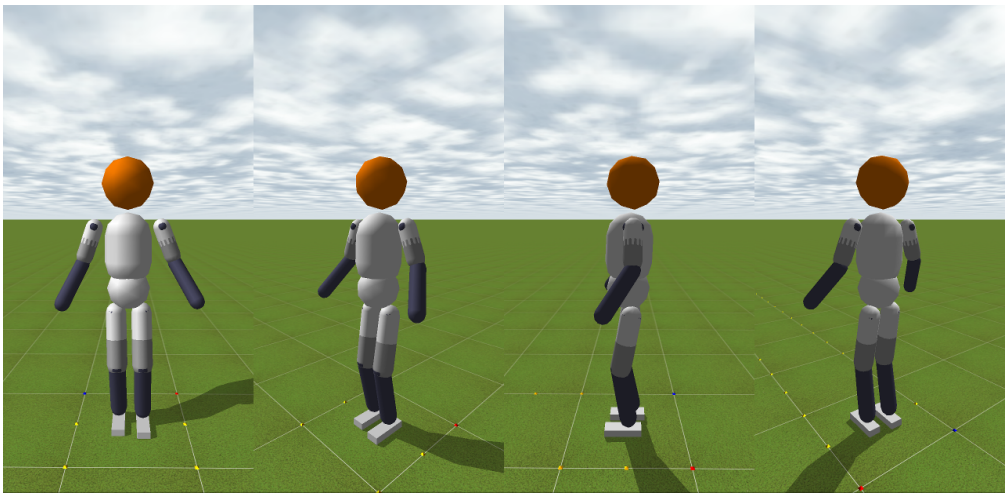

Figure 2.13: The Humanoid robot from different perspectives.

The Humanoid is the most sophisticated robot within these experiments. It has 24 joints and similar-to-human, child-like proportions. The number of joints is significantly greater than with the aforementioned robots. This makes the control problem much harder and it is also an interesting case to see if the algorithms scale in a behaved fashion. Additionally, the problem of keeping or recovering balance will be way more difficult to solve, since the stability margin is reduced to only two feet and the relative height of the center of mass has increased. Thanks to the many degrees of freedom, this robot is expected to show the most versatile movements, but finding them will also be a harder task for learning algorithms. The number of free parameters for the joint controller is already too high, that manually finding good parameters for walking is sheer impossible. And even to find good parameters for the algorithms to start from is already a challenging task.

Conclusion

The robots described in this section were created and selected to demonstrate that all algorithms designed within this thesis generalize seamlessly to quite different *platforms*

Name	Axis	Joint Limits		Def.
Neck Pitch	x	−60°	45°	−10°
Neck Roll	y	−45°	45°	0°
Waist Roll	y	−45°	45°	0°
Waist Pitch	x	−10°	60°	5°
L/R Shoulder Pitch	x	−120°	60°	0°
L/R Shoulder Roll	y	−90°	150°	20°
L/R Shoulder Yaw	z	−90°	90°	10°
L/R Elbow Pitch	x	0°	120°	30°
L/R Hip Pitch	x	−30°	90°	8°
L/R Hip Roll	y	−90°	90°	1°
L/R Hip Yaw	z	−30°	60°	0°
L/R Knee Pitch	x	−120°	0°	−16°
L/R Ankle Pitch	x	−45°	45°	9°
L/R Ankle Roll	y	−45°	45°	0°

Table 2.7: Joint ranges for the Humanoid robot.

independently of their individual number of joints. All robots have in common that they are built from revolute joints and rigid bodies. So in the current state of development, no statements could be made on how these algorithms will perform with different types of actuation or when using compliant body segments.

The simulated robots provided range from a low to a high number of joints, from easy to difficult to balance and from very restricted to quite versatile movement abilities. So, in general, there is always the tradeoff between complexity in control and mobility.

Name	DoF	Size	Weight	Height CoM
Pendulum	1	L:50 cm	0.47 kg	
Crawler	2	L:15 cm	2.15 kg	
Tadpole	4	L:20 cm	1.77 kg	
Quadruped	12	L:40 cm	4.72 kg	21 cm
Humanoid	24	H:105 cm	9.49 kg	60 cm

Table 2.8: Overview of the used robot platforms and their properties.

The robots all have a comparable size within the same magnitude. This constraint comes from having a single type of simulated servo motor which is identical for all robots. Each robot has exactly one motor per revolute joint. However, the Tadpole is restricted to use 1/5, Pendulum 2/5 and Crawler 3/5 of the maximal available torque a motor can provide, the humanoid has doubled the torque. If the size of the robots is kept moderate, it is also easier to reproduce the results with real hardware, since hardware inevitably produces cost while usually costs increase if the parts are either too large (much material) or too small (high integration).

How to handle undesired terminal states

Just to give a brief note: Each hardware design of mobile robots faces the problem of (undesired) *terminal states*, which are states of either body or environment in which the robot may get stuck and is henceforth not able to recover itself regardless which action it takes and needs external help. Terminal states can be partially avoided by hardware modifications or environmental changes but it is hard to avoid all terminal states or even foresee them. But the performance of learning algorithms can easily suffer from such situations. Fortunately, with an increased number of joints the risk of getting stuck often declines. But for simple robots used in unsupervised learning scenarios, terminal states can be a severe problem since it implies that we need to know about these states, must be able to detect them, and provide a mechanism to reset the robot to a safe position.

However, it is not always desirable to avoid all terminal states. Since we could also think of incorporating them into learning and always give a penalty in order to avoid them in the long run. Another approach would be to allow the robot to send a *cry-for-help* signal which will cause some external mechanisms to help the robot to reset it, as sort of an action which can be taken actively by the robot. In nature, there might be others, e. g. parents, which could help. Surely, it comes to mind that we should also give a lower reward when using this action very often.

2.5 Scenarios for increasing the robustness of walking

The flat and even ground is the simplest world for walking robots. There is nothing in the way, the robot could stumble over. Provided enough time, it could just move infinitely far in the same direction without ever reaching an end of the track. So when the task of walking on even ground is solved with moderate speed and low chance of falling, we should subsequently raise the level of difficulty. Hence, we should provide a variety of scenarios in which the robot can move under more complicated conditions.

For the sake of increased robustness of the learned gaits, we need to create external disturbances. For instance, we could push or pull on the legs or torso while the robot is walking. We could place some obstacles in front of the robot and it must successfully step over them, or we could introduce slopes. Whatever, there must be sufficient noise in the world, if we want the controller to be able to deal with it. Otherwise, evolution or learning algorithms will always adapt to the clean, flat and even ground conditions and the acquired behaviors will always be on the edge of stability.

This adaptation towards critical stability must inevitably happen if we optimize for speed or efficiency only. It is for some simple reason: Stability has costs. It has costs to search for more robust parameters and the optimizer will find better solutions sooner if the robot does not stumble and fall often since the objective function is less noisy. If we optimize for speed, faster walking naturally increases the risk of falling. But also if we optimize for efficiency. The essence of efficient behavior is a reduction to the bare, minimal effort needed. Therefore, when trying to minimize the control output needed in order to save—let us say electrical power as a real-world equivalent—then it will have

more costs to react to disturbance and recover balance and find the track again. That also means a simple linear controller cannot be minimizing control costs and be rock solid at the same time. A trade-off must be found between stability on the one hand and efficiency or speed respectively on the other.

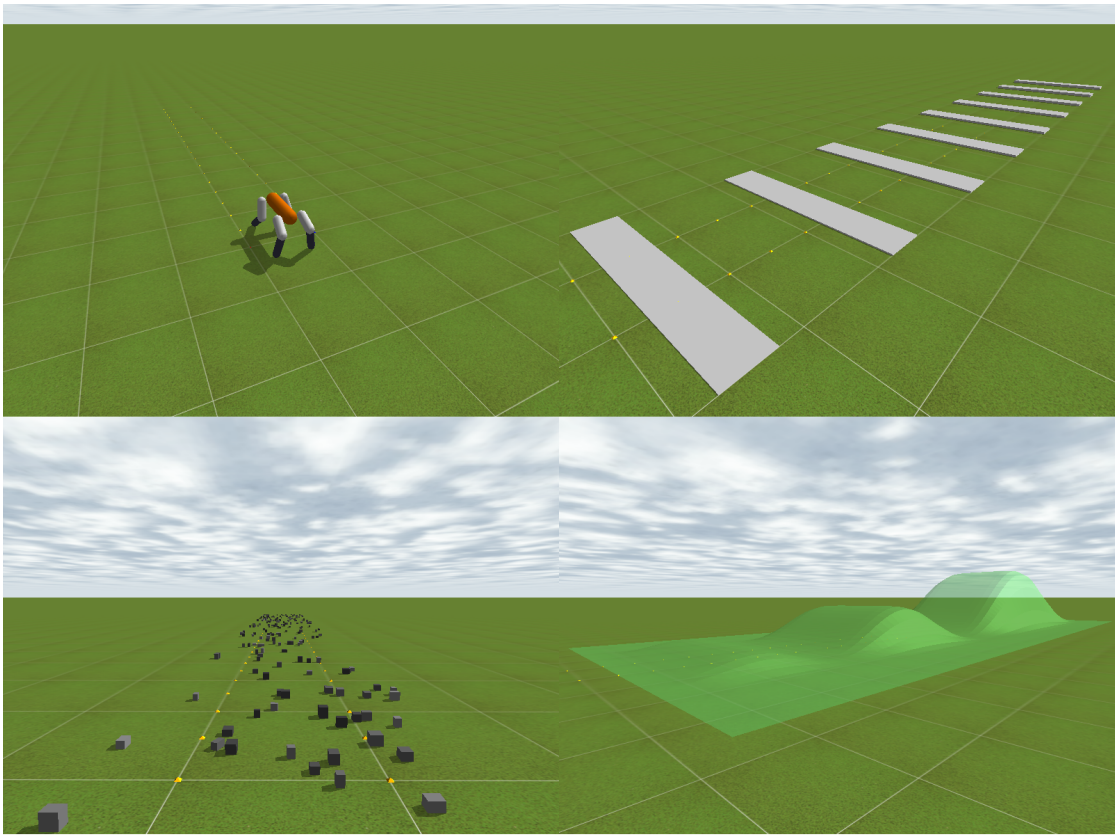

Figure 2.14: Available predefined scenarios: Flat, hurdles, rocky field, and hills.

Hurdles with increasing step height

The scenario called *hurdles* is a lane with subsequently increasing step height of a fixed breadth. Every next step is increased by 5 mm with the last step having a final height of 5 cm to be overstepped. For the given walking robot platforms, this is approximately 10-15% of their leg length. At first glance, this does not sound much a problem but considering only proprioceptive and no visual input to prepare for a heading collision, this is already quite challenging for the given systems. The hurdles are placed in intervals of 1 m, with and breadth of 25 cm of each hurdle. The hurdles are heavy and cannot be moved by the robots.

Rocky field

The *rocky field* consists of many freely movable rectangular small sized rocks whose position is defined randomly at startup and uniformly distributed over the main expected

pathway of the robot. The robot can coincidentally push against a rock or can step on them. However, the rocks may tilt if the robot's foot is placed very next to the edge, which can lead to even more destabilization. The size of the small rocks are limited to have side length from the restricted interval of $a = [2\,cm, 4\,cm]$. The rocks are heavy with a density of $d = 2800\,kg/m^3$ similar to granite.

Hills

As the hurdles, the *hills* was created to provide a challenge of subsequently increasing difficulty. It starts with a small easily manageable bump and evolves to a moderate hill with a steep slope. It has shown to be the most challenging scenario for the given robot platforms. With the hills, the controllers shall be driven towards an invariance against ground slopes and the resulting gravity forces pushing or pulling the robot in or against the current walking direction. The hills are flattened on top to provide a minimum save walk corridor with an approximate breadth of 1 m. Therefore the robot will drop down the hill if this corridor is left significantly.

Conclusion and Discussion

All simulated robot experiments are conducted using a physical simulation environment called Simloid. This project was initially started by Daniel Hein for his diploma thesis Hein (2007); Hein et al. (2007). Since then, large parts were rewritten and extended by myself to match the needs of my thesis. Many new features were included, but also a few were removed. This decision, not to use a highly developed simulator off-the-shelf, but instead take this project and develop it further as the foundation for my experiments, was based on a simple argument: I wanted to be able to give it the most suitable form for my experiments. And I did not want to spend any time in struggling with the usual pitfalls of higher developed simulators. Since I started, the *Gazebo* simulation environment became quite popular, is now a powerful open source project, and by today it would probably be a good choice for similar ventures to start from. The effort is too high to continue developing Simloid, especially when it comes to more elaborate models of multiple robots in complex environmental situations.

However, we could consider extending the motor model to make the actual electrical power P accessible. From electrical power, we could make a rough estimate for the energy needed to get the robot somewhere. And therefore it gives us an estimate on how much battery power, capacity, and therefore mass and volume we need for a given platform. The traveled distance is a function of the payload, and the battery's size, given its energy density (Wh/m^3) and specific energy (Wh/kg) is fixed. So increasing the battery size will increase the energy but will also consume more energy to walk. A tradeoff must be found. Electrical Power consumed by motors is defined as

$$P = UI = U^2/R, \tag{2.12}$$

with voltage U and current I, and roughly assuming the resistance R to be constant, then U^2 is a fairly simple approximation of mechanical power. We will see this term to be

used as part of different objective functions in later chapters. Possible extensions of the model could use R as a function of the temperature $R(T)$. Since excessive current flow heats the motors, which is especially true for near-stall conditions, e. g. when standing up. Increased temperature increases the resistance and hence decreases the momentary usable power of the motor and also decreases its lifetime in the long run. We could also consider using this as kind of a negative feedback signal for learning, similar to pain (peak currents) or exhaustion (heated motors).

Another major influence to the model would be to take the input voltage U as a function of the battery's state of charge. Since this would only affect longer periods of walking where battery voltage is usually decreasing, an immediate battery voltage drop always occurs when other motors source power from the battery. Hence, there is a coupling between motors via supply voltage. The voltage drop is proportional to the current drawn and the inner resistance of the battery, which itself is merely a function of the battery's temperature and age. So there is huge potential that would justify putting further efforts in modeling electrical properties of the robot's motors and energy system when regarding the system as a whole. The interesting part would be to see how locomotion gets affected by a continuously decreasing voltage level, and what the effects on learning are. How can we find controllers that can deal with largely decreased voltage without losing the capability to walk? How will the gait be adapted to different levels with identical controller weights? An important question when considering real robots, since constant supply voltage on mobile robots' electronics comes with losses anyway—since mostly battery voltage is passed to the motor drivers without further regulation.

Possible extensions to the mechanical model range from flexible or soft-body dynamics to different ground structures. But it appears highly interesting to find out the effects to learning when using (simulated) Strain Gauges (sensors for indicating the strain of a material or structure at the point of attachment). Either directly used as further position feedback, since bending legs, for instance, can be regarded an additional degree of freedom, or they could be incorporated into objective functions in order to reduce mechanical stress to the material. In the current simulation, the objects are not deformable, but at the joint connections, the solver must apply forces to meet the constraints given by the model. It must keep together the body parts when subjected to load and apply joint correction forces. So even in a rigid-body dynamic simulation like the ODE used by Simloid, we have the chance to get sensory information on the point where real legged robots' hardware would be affected the most by material stress: the joints.