ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester-Thesis

Quadruped walking/running simulation

Spring Term 2011

Supervised by:
Marco Hutter
David Remy

Author:
Dominik Näf

Contents

Abstract

This report describes the work that has been done for the semester project *Quadruped walking/running simulation*. During the work a 3 dimensional simulation of a trotting four legged robot was implemented in the MATLAB SimMechanics environment.
The work was split into two phases: First, a model of the quadruped was build with the main attention put on the creation of the ground contact model. A soft contact approach with a linear spring damper systems was chosen. During the second phase a simple and robust controller for the robot in a trotting gait was created. For that, a Raibert three-part controller was applied. To be able to do that for our four legged robot, a virtual model had to be defined for the legs.
For the determination of the best fitted parameters for the control algorithm, as well an empirical and an optimization method based on an eigenvalue analysis were used. The created controller delivered satisfying results in its reactions to disturbances. Specially roll and pitch angles could be corrected successfully, whereas stabilization of disturbances in horizontal velocities might be improved.

Symbols

Symbols

f_x, f_y, f_z	ground reaction force in x, y and z-direction
T_{st}	duration of stance phase
x_f	position of the foot relative to the hip in x-direction
$\Theta_r, \Theta_p, \Theta_y$	roll, pitch and yaw angle
$\mathbf{T}$	vector of actuator torques
F	force applied to CoG
F_{pitch}, F_{roll}	vertical force for pitch and roll control

Acronyms and Abbreviations

CoG	Center of Gravity of the main body of the quadruped
FL	Front Left (leg)
FR	Front Right (leg)
RL	Rear Left (leg)
RR	Rear Right (leg)
VL	Virtual Leg (leg)
TD	Touchdown point
ETH	Eidgenössische Technische Hochschule

Chapter 1

Introduction

The control of legged locomotion is one of the most challenging fields in robotics research today. Even though wheeled locomotion typically is very energy efficient and easy to control, legged locomotion is better suited when it comes to rough and unstructured terrain. While in the early years the focus was mostly laid in the definition of static controllers, the focus nowadays moved on to dynamic and energy efficient locomotion.

1.1 About the Project

This work is part of a research project of the ASL (Autonomous System Lab) at the ETH in Zurich. The main goal of this project is to build a prototype of a four legged robot which is specialized in energy efficient locomotion. This efficiency should be achieved by using passive dynamics for different dynamic motions such as fast walking, trotting, bounding and galloping. A CAD rendering of the robot developed for this purpose can be seen in Figure 1.1.

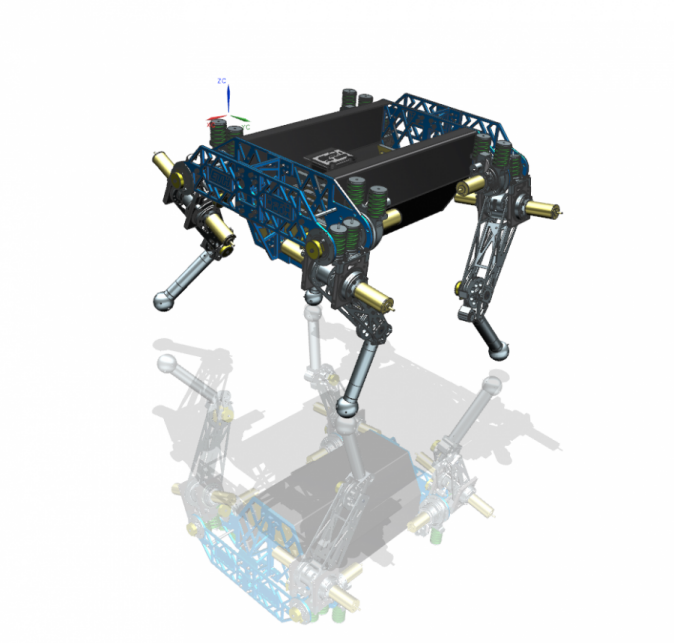

Figure 1.1: CAD rendering of the StarlETH robot.

1.2 Task

The task of this thesis is to implement a 3 dimensional model of the quadruped used in the project with MATLAB SimMechanics. The model should reflect the physical properties of the robot and its environment, be computationally efficient, and should allow the integration of different control algorithms in a simple way. Further, a control algorithm should be implemented which controls the created model in a trotting gait. The controller should be robust, i.e. it should be able to react to different disturbances in a stabilizing ways.

Chapter 2

Model

2.1 The Model

In order to create a controller that regulates the quadruped, a reliable model is required. The main requirements to build this model were:

- a realistic reproduction of the multi body structure
- an acceptable calculation time
- suitable for simple integration of several controllers
- easily understandable and adjustable

The quadruped modeled is described in the next section.

2.2 The Quadruped

The robot modeled in this work is a four legged machine with three actuated joints per leg (Figure 2.1).
The three joint actuators are considered as optimal torque sources for hip abduction, hip flection, and knee flection. Six different coordinate systems are defined and used at the robot:

- The world coordinate system (CSw).
- The main body fixed system originated at the CoM of the main body (CSmb).
- Four local coordinate systems originated in the hip joints of the legs (CSloc) with a fix offset to CSmb.

The main body had a length of 0.5 m and a mass of 10 kg. For a more detailed specification please consult the following table.

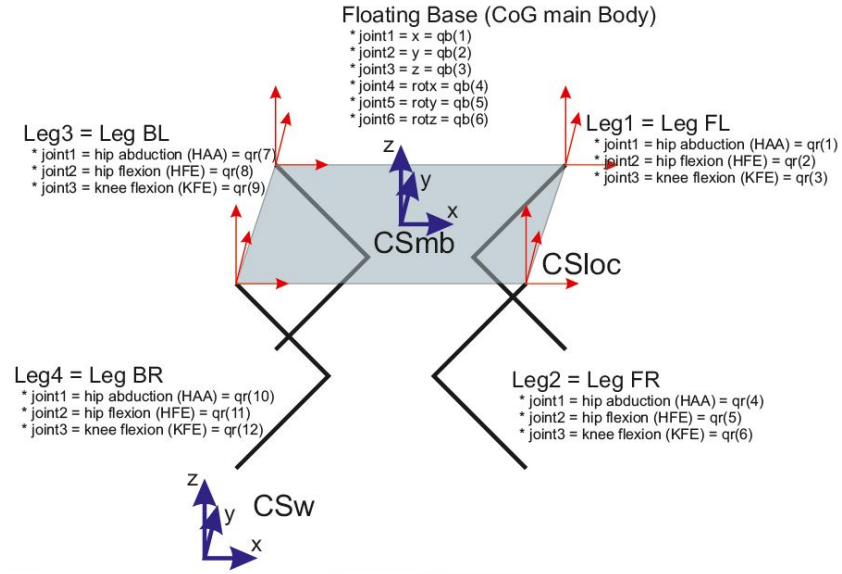

Figure 2.1: Definition of the Legs and Coordinate Systems

Parameter	Value	Description
param.mainBody.m	10	main body mass
param.mainBody.I	diag([0.128 0.171 0.9 0.171])	main body inertia
param.mainBody.b	0.3	main Body width
param.mainBody.l	0.5	main Body length
param.thigh.l	0.2	thigh length
param.thigh.s	0.03	CoG position in z direction from hip
param.thigh.m	1.78	thigh mass
param.thigh.I	diag([1e-3 1e-3 1e-5])	thigh inertia
param.shank.l	0.2	shank length
param.shank.s	0.1	CoG position in z direction from hip
param.shank.m	0.548	shank mass
param.shank.I	diag([3e-3 3e-3 1e-5])	shank inertia
param.hip.l	0.05	distance hip abduction to hip flexion axis in neg z direction
param.hip.s	0.025	distance hip abduction to CoG in neg z direction
param.hip.m	0.5	hip mass
param.hip.I	1e-3*eye(3)	hip inertia

2.3 SimMechanics

To achieve the attributes described in Section 2.1 MATLAB Simulink and the SimMechanics toolbox were chosen as simulation environment. This approach provides a straight forward way to create the model, which is important for later corrections or improvements of the model.

MATLAB SimMechanics is an additional toolbox in the MATLAB Simulink environment, which is specially developed for the modeling of 3-dimensional physical problems. A library providing a wide range of body-, joint-, constraint-, and other blocks representing the physical structure of the model is offered to provide an intuitive approach. The blocks represent physical body parts, such as the main body or the hip to which you can assign certain dimensions, inertias and masses, but also joints or actuators to which you can assign angles or torques. Furthermore, you can also use blocks that serve as sensors and let you create an interaction point between your model and the control algorithm. Like that it is possible to build a model just as you would assemble a physical system in reality. SimMechanics calculates the mechanical motions using forward dynamics, inverse dynamics and kinematics, trimming and linearization.
By the fact that the SimMechanics environment is based on MATLAB Simulink, you can build subsystems based on Simulink models or even embedded functions. Like that, the model can be created entirely in MATLAB Simulink and SimMechanics, but the controller can easily be integrated as an ordinary MATLAB function. More about this topic can be seen in the Section 2.5.
For more information about MATLAB SimMechanics please consult [1].

2.4 Literature Research

Before starting to build the model, a literature research was done. For the main architecture of the model not a lot of information was accessible, since a very particular model was planned here. Some insights were gained from Philipp Omlins work [2], which addresses a similar problem with MATLAB Simulink. The main issue of the research was to find the most suitable way to implement the ground contact forces. In the papers [2], [3], [4], [5] and [6] several different solutions are discussed using soft contact properties.

2.4.1 Hard Contact versus Soft Contact

In the literature mainly two different approaches for the modeling of contact problems can be found - hard and soft contact models. The difference between hard contact and soft contact modeling is that in a hard contact model a geometric constraint is used (i.e. the body can't penetrate the ground), whereas in a soft contact model a constraining force is applied.

Hard Contact: For the simulation that means that in case of a hard contact model, the ground contact is modeled as an instantaneous discrete transition between the two states 'no contact' and 'contact'. That enables us to use different coordinates during different contact situations and hereby to use minimal coordinates [6].To do so, a hybrid state vector (x_c, x_d), which consists of the continuous state vector

$$x_c = (q^T, \dot{q}^T)^T \in \mathbb{R}^n$$

and the discrete state

$$x_d \in \mathbb{Z}$$

, is defined. Depending on the discrete state the continuous differential equations for the state vector are of a different form.

$$\dot{x} = f(x_d, x_c)$$

To define the discrete state, guard conditions are needed which tell us when a discrete transition takes place (in our case the event when the foot touches/leaves the

ground). Furthermore we need reset conditions for our state vector (e.g. the velocity in vertical direction $\dot{x_z} = 0$ at touchdown). This method is mainly used in simulations with few interaction points between body and ground ([6]).

Soft Contact: The soft contact is in our case modeled as a spring damper system between the first touchdown point of the foot on the floor and the foot itself (Figure 2.2). Since the force is applied between the initial position of the foot and the actual position, this method is also called penalty method. For the spring damper system as well linear and non-linear systems were used.

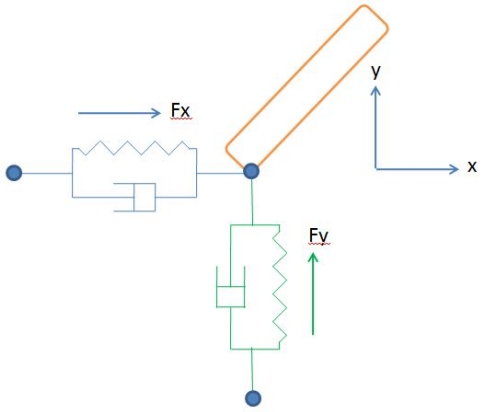

Figure 2.2: Soft Contact modeling in 2D

The non-linear form of the system is mainly used to prevent from two major weaknesses the system has with linear spring damper systems. The first problem is the discontinuity of the forces at impact. Whereas the force generated by the spring is zero right after the impact, the force generated by the damper then is at it's maximum. Physically, the force should grow continuously starting at the moment of impact. The second problem occurring are the so called sticking forces. The sticking forces basically are the opposite of the first problem mentioned above; they arise when the foot's velocity points in positive y-direction (Figure 2.2). In this case a force in negative z-direction is possible or even bound to happen shortly before the foot leaves the ground. The sticking forces are tensile forces holding the foot and the ground together, which is physically not correct. To prevent these two phenomena from happening, different approaches of non-linear spring damper systems can be found in the literature. Because both problems mainly affect the forces in vertical direction, we we'll look here solely at spring damper system with a linear part in the horizontal directions and a non-linear part in the vertical one. Two of them are presented below.

- [5] and [6] choose a spring damper system where the damper is also dependent on the displacement of the foot.

$$f_y = -(\lambda x^n)\dot{x} - kx^n \quad , \tag{2.1}$$

  where n is often close to one and is depending on the geometry of the surface. Easily it can be seen that this approach suppresses the problems mentioned above. The drawback of this approach is that the system gets more complex

and more uncertain parameters will have to be found to make it work.

- [3] and [4] choose a different kind of non-linear spring damper system of the form

$$f_y = \begin{cases} -k_y y - d_y \dot{y} & \text{if} - k_y y - d_y \dot{y} \geq 0 \\ 0 & \text{else} \end{cases} . \tag{2.2}$$

  As can be seen this definition of the forces also gets rid of the sticking forces, but the discontinuity at impact is still given. The big advantages of this approach are first it's simple structure - no variables or multiplications are added - and second, the fact that a lot of literature is available where this method was successfully implemented.

Additional to the two fundamental concepts mentioned above there are also some alternative contact models existing using a mixture between soft and hard contact models [7]. A hard contact model is used to describe the impact whereas the overall behavior is modeled as a compliant model. In this work these possibilities won't be discussed.

2.5 Architecture

The main goal of the general architecture was that it should allow the implementation of different controllers for the same model. Because of that, the idea was to separate the control algorithm from the model (Figure 2.3). The discrete controller receives the entire state vectors q and dq from the model at a constant time interval t and sends back the calculated torques $\mathbf{T}$.

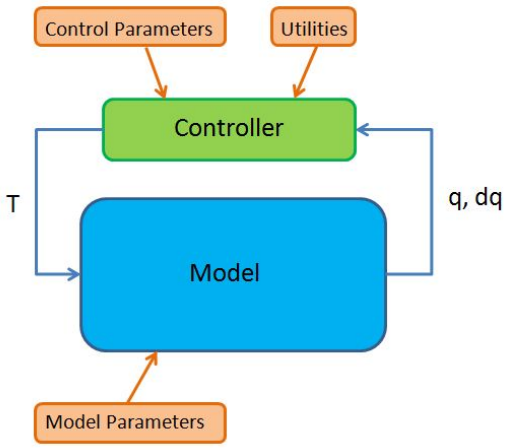

Figure 2.3: General architecture

The state variables q and dq consist of

$$q = (qb, qr) \in \mathbb{R}^{18}$$

$$dq = (dqb, dqr) \in \mathbb{R}^{18}$$

where qb is the 1x6 base coordinate vector. The entries $qb(1), qb(2)$ and $qb(3)$ are the x,y and z-coordinates of the position of the CoG of the main body. The entries 4 to 6 are used to describe the rotation of the quadruped about the according axes of the global coordinate system with Euler angles. The vector qr contains the joint angles as a 12x1 vector, where the first three entries are the angles for hip abduction, hip flection and knee flection of the first leg (front-left, Figure 2.1), the next three for the second leg, and so on. The vector dq has the same structure as q, except that its entries stand for the velocities, respectively the rotation rates. The input variable $\mathbf{T}$ consist of the torque vector, which has the same structure as the vector for the joint angles:

$$\mathbf{T} \in \mathbb{R}^{12}$$

This separation of control algorithm and model enables us, to use the same controller to control the robot in the simulation and in reality, since the same in- and outputs are available. By defining the parameters of the model in an external file, the model can easily be adapted or modified. The control parameters and the utilities will be discussed in Section 3.2.

2.5.1 2D Model

Before creating the 3 dimensional model of the quadruped, a simpler 2 dimensional one of a single leg was created. Like that it was possible to get used to the SimMechanics toolbox by implementing the ground contact model on a smaller scale first. On top of that it was possible to get a first insight about how to choose the right parameters.

Like in the final simulation, the model was split into two parts, the physical modeling with SimMechanics and the control algorithm as an embedded function. Figure 2.4 shows the SimMechanics model in a simplified version.

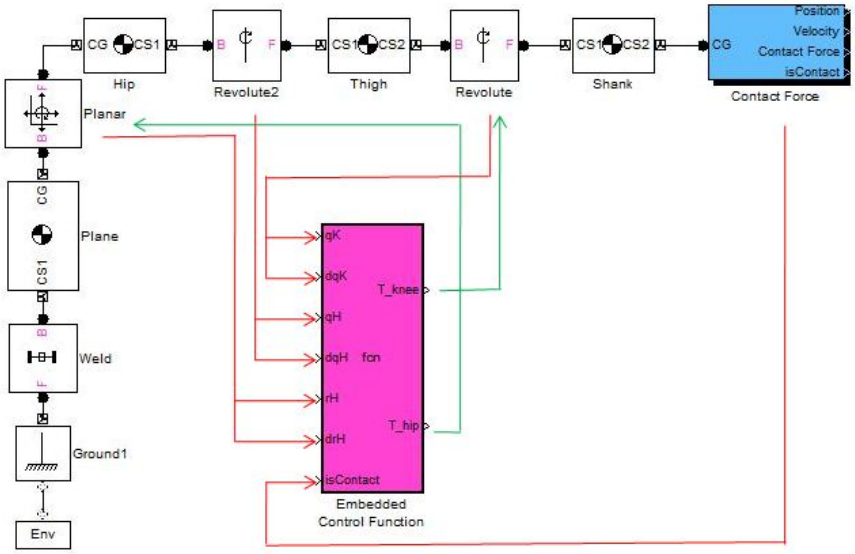

Figure 2.4: Simplified Model of the 2D leg

The red arrows show the state variable consisting of the joint angles, joint velocity, vertical position and vertical velocity, whereas the green arrows show the input signals to the actuators in the joint. To get the state variables from the model and

to be able to apply the desired torques, additional blocks in the model are needed. Figure 2.5 shows the knee joint in detail with the additional sensor, actuator and initial condition blocks.

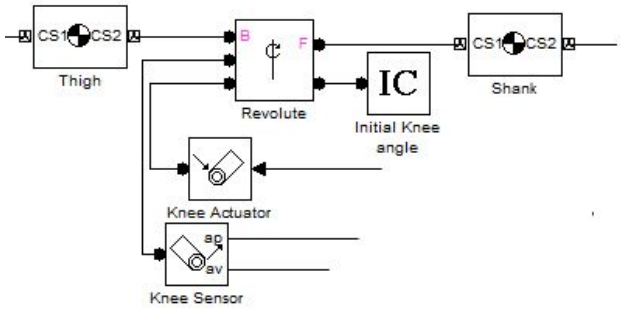

Figure 2.5: Knee joint with sensor, actuator and initial condition blocks.

In the blocks for the hips, shanks or thighs the properties of these body parts (i.e. mass, length, inertia, etc.) are assigned. Also in these blocks, the definition of the coordinate systems was done. It was crucial to stick strictly to the predefined order of coordinate systems defined in Section 2.2. The initial conditions, as well as the parameters for the body parts and the ground contact model, are defined in the parameter file. The ground contact model itself was designed based on the bouncing ball MATLAB tutorial as a separate block. Details to the ground contact model can be found in the Section 2.6.

2.5.2 3D Model

As introduced in Section 2.5 the model was split in two parts: the physical model and the control function. In Figure 2.6 this distinction can be seen.

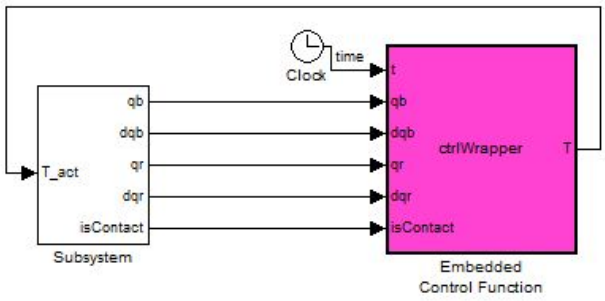

Figure 2.6: Structure of 3D model.

The model in 3 dimensions was build using the already built model for a single foot. For that matter it was necessary to allow motion and calculate the forces in the third dimension. Further, the robot had to be composed by combining 4 legs, add

the physics of the main body, and implement the hip abduction joints between the main body and the legs. In Figure 2.7 this structure can be seen.

The block subsystem containes the whole model of the quadruped. On a first level the connection between the main body and the environment and the connection between the main body and the legs was modeled (Figure 2.7). Again the input torques were colored green and the output state variables were colored red.

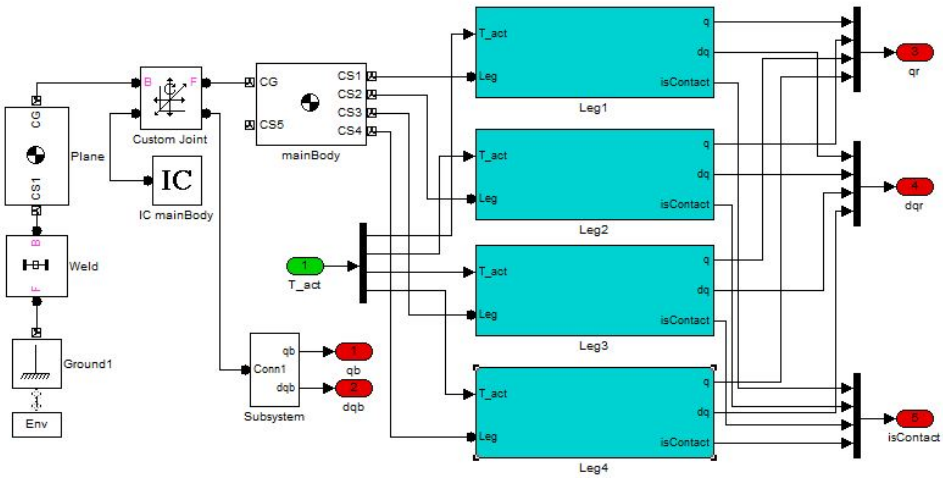

Figure 2.7: Quadruped model - Main body level.

On the next level the dynamics of the legs (cian blocks in Figure 2.7) were defined in a similar way as shown in the 2 dimensional example (Figure 2.8).

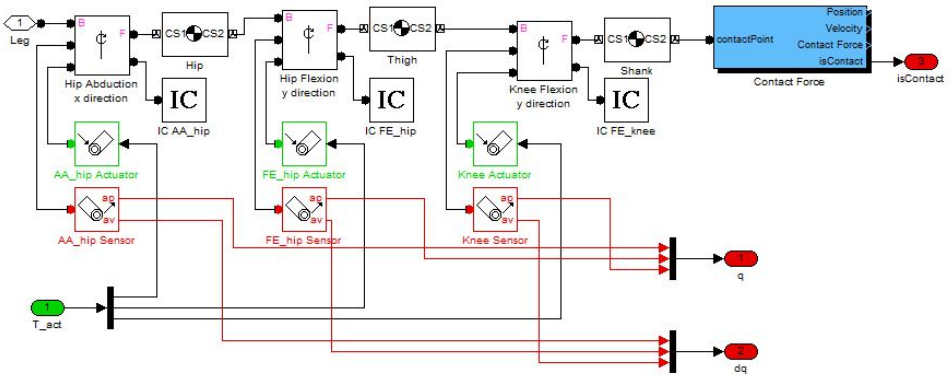

Figure 2.8: Quadruped model - Leg level.

2.6 Ground Contact

We decided to use a soft contact model of the form

$$f_x = -k_x(x - x_0) - d_x\dot{x} \tag{2.3}$$

$$f_y = -k_y(y - y_0) - d_y\dot{y} \tag{2.4}$$

$$f_z = \begin{cases} -k_z z - d_z\dot{z} & \text{if} - k_z z - d_z\dot{z} \geq 0 \\ 0 & \text{else} \end{cases} . \tag{2.5}$$

To implement this approach in the model, we needed to hand over the actual position of the foot and the velocity of the foot to the ground contact subsystem. By checking the vertical position, two different discrete states were defined:

$$State = \begin{cases} \text{Contact} & \text{if } z \leq 0 \\ \text{No contact} & \text{else} \end{cases} . \tag{2.6}$$

If the discrete state was 'No contact', the ground reaction force immediately was defined as zero. In the case 'Contact' the resulting forced had to be calculated. While the hight in vertical direction always was compared with zero (the hight of the ground), the forces in the horizontal direction were compared with the touchdown point. For that, first the touchdown point was stored (Figure 2.9). Once calculated a saturation block was used to make sure the force in vertical direction is greater than zero. The last step was to hand back the discrete state to the model, since this variable is also used in the control algorithm.

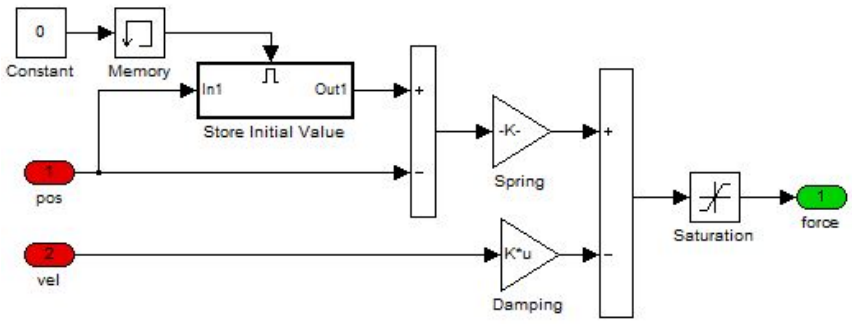

Figure 2.9: Soft Contact modeling in 3D

There occurred two main difficulties while implementing this ground contact model. The first one was to find the right parameters to get the desired results. Only a certain penetration dept was wanted and the damping rate shouldn't be to high either. In addition to that, when the spring and damping constants were chosen too high, the differential equations got very stiff, which caused the next problem: to find the right solver. The same parameters can result in completely different behaviours just by changing the solver used. It was even possible to observe that with one solver the simulation terminates, and with another one singularities occurre. The best solver found was the ode15s solver, which is specially fitted for ordinary differential equations which are stiff.

2.7 Visualization

The visualization of the model was done automatically by MATLAB SimMechanics, another great advantage of this toolbox. The model in its intial configuration in flight phase can be seen in Figure 2.10.

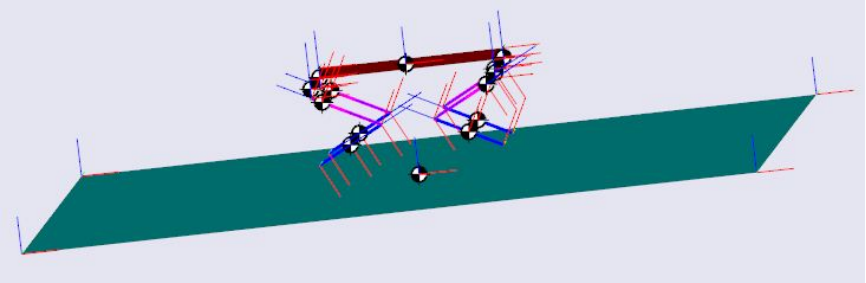

Figure 2.10: Soft Contact modeling in 3D

Chapter 3

Controller

The second big part of this work was the implementation of a control algorithm which stabilizes the quadruped while trotting. In a trotting gait the left foreleg and the right hind leg touch the ground in unison, followed by a flight phase where all four legs are in the air, followed by the stance phase of the remaining two legs (Figure 3.1).

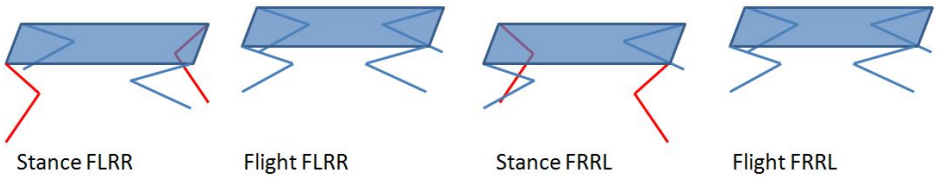

Figure 3.1: Different phases of a trotting gait. FL = Front Left, FR = Front Right, RL = Rear Left, RR = Rear Right

Since trotting is a dynamic gait and the support polygon in stance phase was reduced to a 1 dimensional line connecting the two feet, a simple static locomotion control or a ZMP (Zero Moment Point) based control strategy was not possible. We were looking for a control algorithm for dynamic locomotion, which considered actuation and sensor feedback, so another literature research was necessary.

3.1 Literature Research

The literature research turned out to be more time consuming than expected, because there are many papers about legged locomotion control algorithms, but only a few examples of trotting quadrupeds where the control strategies are easily accessible. Most of the papers found were dealing with four legged static locomotion where always at least three legs are in contact with the ground. Since we were looking for a robust, but also a simple algorithm, the work of M. H. Raibert turned out to be highly interesting ([8], [9]).

3.1.1 Raibert Control

In [8] a one legged robot which contained of a springy leg and a gimbals-type hip, which allowed to control the orientation of the leg, was examined (Figure 3.2). The advantage of this model is that it combines the relatively simple dynamics of

a spring-mass model with additional control possibilities. For further information about the spring-mass theory, please refer to [10]. Later in his research, he extended these principles to control the locomotion of a four legged robot, which makes it interesting for our project.

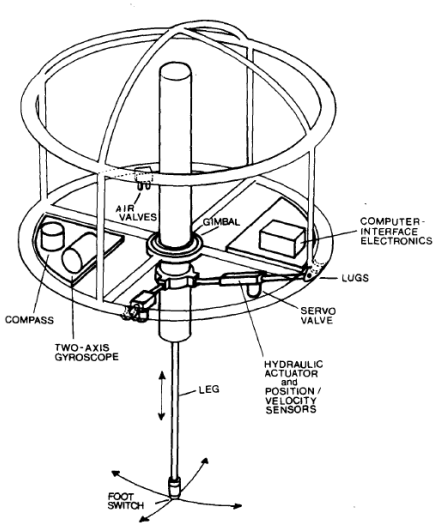

Figure 3.2: Raibert Mono hopper from [8].

The controller used to stabilize the hopper contained of three parts:

- The velocity in horizontal direction was controlled by an appropriate foot placement during the flight phase. The right position of the foot relative to the hip at touchdown is important since it controls in which direction the thrust applied in the leg acts. To calculate the right position two factors were considered. First the actual forward velocity. For a constant forward movement, spring-mass model theory dictates a symmetrical stance phase. To find the right touchdown point for a symmetrical stance phase, an estimation for the next stance phase has to be made by considering the predicted mean forward velocity and the predicted stance time (Figure 3.3).

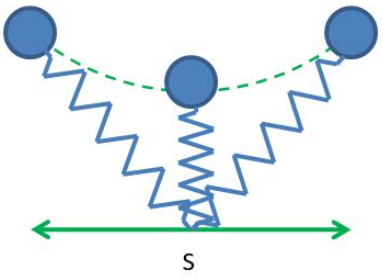

Figure 3.3: Spring-Mass Model during stance phase with $s = \dot{x}T_{st}$ and $\dot{x}$ = forward velocity and T_{st} = Duration of stance phase.

The hereby found touchdown point will be called TD in the next subsection.

The second factor considered was the error in forward speed. To be able to control the forward velocity, the offset of the foot at touchdown relative to the foot placement calculated before (TD) can be used. By making the stance phase asymmetrical, different behaviours can be caused; E.g. if the foot is placed in front of TD, the period of time where the mass decelerates in horizontal direction is smaller than the period where the mass accelerates. Like that an overall acceleration can be caused. Three different scenarios are possible, illustrated in Figure 3.4. The offset is calculated using the difference between the actual forward velocity and the desired one, multiplied with an experimentally found gain $k_{foot} > 0$.

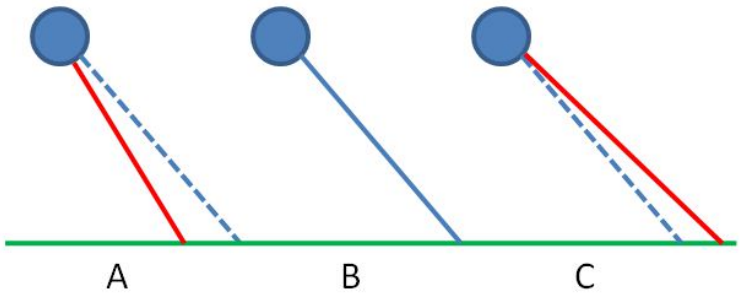

Figure 3.4: Case A: Foot placed in front of TD, the velocity will increase. Case B: Foot placed at TD, no change in velocity. Case C: Foot placed behind TD, the forward velocity will decrease.

Combining these two factors results in the following equation for the foot placement, valid in x- and y-direction:

$$x_f = \frac{\dot{x} T_{st}}{2} + k_{foot}(\dot{x} - \dot{x}_{des}) \tag{3.1}$$

- The hopping height, respectively the energy niveau, was controlled by the amount of thrust inserted in the legs actuator. This is the most intuitive part of the controller and hereby only discussed briefly. Meanwhile the foot placement was adjusted during the flight phase, the hopping height has to be controlled during the stance phase. Since Raibert used in his work a springy leg, the robot would also hop without insertion of any thrust in the leg. However due to not negligible losses, an additional thrust is necessary to compensate these. Since these losses are monotonic with the hopping height, a fixed thrust could be found which stabilizes the robot at a certain hopping height.

- The balance of the body (attitude) was controlled by applying torques between the leg and the body in stance phase. Since the angular momentum during the flight is conserved there's no possibility to control the attitude during this phase. The torques applied during stance were calculated considering the actual roll and pitch angles and the velocities. The resulting equation can be seen here:

$$\tau_x = k_r(\Theta_{rdes} - \Theta_r) - d_r(\dot{\Theta}_r) \tag{3.2}$$

$$\tau_y = k_p(\Theta_{pdes} - \Theta_p) - d_p(\dot{\Theta}_p) \tag{3.3}$$

Where τ is the torque applied about the x- respectively y-axis, Θ stands for the roll respectively pitch angle and $\dot{\Theta}$ for the angle rate. The parameters k and d had to be found again experimentally.

To adapt the control algorithm used for the one legged robot to a four legged one, Raibert introduced the principle of the virtual leg [9]. The main idea of introducing a virtual leg is to treat two legs as if it were one. In our case treat the two legs FL and RR in stance as if it were only one centered leg VL (Virtual Leg). This approach lets us reducing any quadruped pair gate to better known virtual biped gaits. In our case even a reduction to an one legged gait is possible (Figure 3.5).

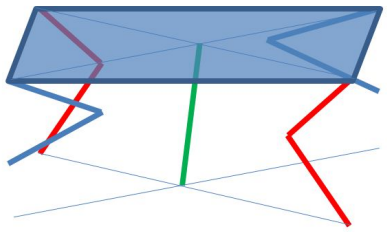

Figure 3.5: The legs FR and RL can be reduced to VL.

To achieve the same behavior with two legs instead of one, some rules for the definition of the virtual leg had to be set:

- The amount of axial thrust exerted by each leg should equal half of the axial thrust exerted by the virtual leg.
- The amount of torque applied between each hip and leg should equal half of the torque around the virtual hip.
- The feet should hit and leave the ground in unison.
- The horizontal position of each foot relative to its hip should equal the position of the virtual foot relative to the virtual hip.

Whereas the first two points were important during the stance phase, the latter two dealt with the foot positioning during the flight phase.

By applying these rules, the three-part locomotion algorithm mentioned above could also be realized for a four legged robot. The only extension necessary is to include a yaw control in the algorithm. Because the yaw angle was also controlled by the foot placement, this enhancement only affected the first part of the controller.

To control the yaw angle, a torque about the z-axis has to be applied. To cause a torque about the z-axis, a force tangential to a circle with center at the virtual hip and through the real hips has to be created (Figure 3.6). For that, the feet have to be misplaced by an angle β. The mechanism of this action is the same as can be seen in Figure 3.4.
Including the adjustments for the yaw control in equation 3.1 the following final solution for the foot placement relative to the virtual hip results:

$$x_f = \frac{\dot{x}T_{st}}{2} + k(\dot{x} - \dot{x}_{des}) + Dcos(\beta + \beta_0) \tag{3.4}$$

$$y_f = \frac{\dot{y}T_{st}}{2} + k(\dot{y} - \dot{y}_{des}) + Dsin(\beta + \beta_0) \tag{3.5}$$

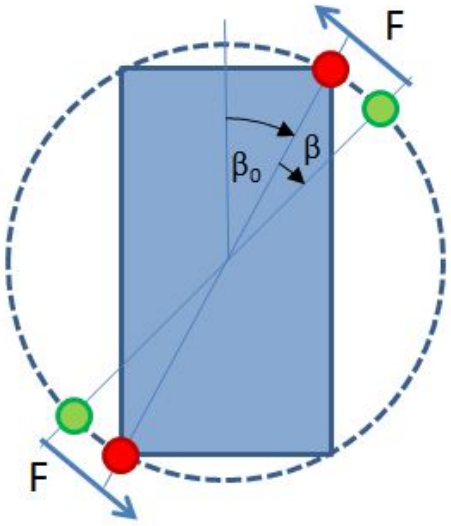

Figure 3.6: Top view of forces created by misplacing the legs. The foot (green) is misplaced by an angle β relative to it's old position (red).

with $\beta_0 = arctan(W/L)$, $D = \sqrt{W^2/2 + L^2/2}$ and L an W are the length and the width of the main body. β was calculated as following:

$$\beta = k_y(\Theta_{ydes} - \Theta_y) - d_y(\dot{\Theta}_y) \tag{3.6}$$

3.1.2 Virtual Model Control

In contrast to the quadruped Raibert simulated, our model doesn't have prismatic legs. In order to apply his method nevertheless, we had to use a virtual modelto convert the wanted axial forces in the legs into actuator torques in our joints [12]. In order to calculate the torques applied in the joint we had to use equation 3.7.

$$\vec{T} = J^T \vec{F} \tag{3.7}$$

With $\vec{F}$ the desired forces and

$$J = \frac{\delta \vec{x}}{\delta \vec{q_L}} \tag{3.8}$$

the Jacobian matrix with $\vec{q_L} = [q_{HAA}, q_{HFE}, q_{KFE}]^T$ the joint angles for hip abduction, hip flection and knee flection and $\vec{x} = [x, y, z]^T$ the vector from the hip to the foot as function of the joint angles $\vec{q_L}$ in the local coordinate system. To transform the position of the foot with respect to the hips from cartesian coordinates to body joint angles, inverse kinematics were used.

3.2 Implementation of the Controller

To implement the algorithm mentioned above it was split in different MATLAB files. While the main control was implemented in one file, some parameters and some important utilities were distributed in several other files. By using the existing utilities a new algorithm can be included more simply in the existing environment. The parameters stored in the external parameters files are the ones used by every controller, such as initial conditions etc.. The parameters used in the control algorithm were stored in the same file as the algorithm itself.

3.2.1 General Architecture

The control algorithm itself was divided in 6 parts:

- Definition of persistent variables which must be available during the whole simulation.
- Two different flight phases.
- Two different stance phases.
- Calculation of the torques according to the forces calculated earlier.

In the following, the defined persistent variables won't be discussed explicitly.

The definition of four different phases was important since we are dealing with a hybrid system and the control algorithm looks completely different for the two discrete states flight or stance. The stance phase has to be further distinguished between stance FLRR (front left leg and rear right leg are in contact) and stance FRRL (front right leg and rear left leg are in contact). Here the differences in the control algorithm are limited to the annotation of the legs but are still important. The same differentiation has to be made for the flight phase; it's a different behavior for a leg whether it prepares for a stance phase or not. For making this differentiation, a discrete state variable which can have the states 'stanceFLRR', 'stanceFRRL', 'flightFLRR' and 'flightFLRR' (Figure 3.7) was introduced. This variable was named 'ctrl.phase' and was of course a persistent one. Depending on the discrete state, the according control actions were taken as discussed in the previous section.

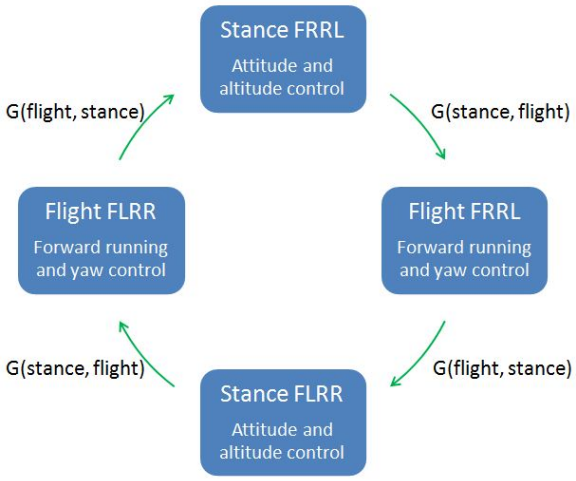

Figure 3.7: Different discrete states during simulation.

The moment of transition from one discrete state to another is defined by the guard conditions G. The transition requirement from stance to flight phase is

$$G(stance, flight) = \Theta_{KFE,stance} < 0.9 \vee F_{z,stance} < 0 \vee isContact_{stance} == 0 \quad . \tag{3.9}$$

In words this means, if either the knee angle of a leg in stance $\Theta_{KFE,stance}$ is smaller than 0.9 rad (i.e. extended to a desired value), the vertical force applied to the leg

is negative or one of the stance legs lost contact with the ground, a transition is taken and the discrete state switches to 'flight'. The guard condition to switch from flight phase to stance phase looks like

$$G(flight, stance) = any(\Theta_{KFE,stance} \geq 1) \wedge any(isContact_{stance} == 1) \quad . \tag{3.10}$$

In words, if any of the legs preparing for stance phase is in contact and one of the legs preparing for stance phase has a knee flection angle greater or equal to 1 rad, the transition is taken. Both guards were treated with an as-soon-as semantic, which means that as soon as a transition was possible it was taken.

3.2.2 Force Calculation

To calculate the forces acting on the legs, one has to differentiate between legs in contact with the ground and legs not in contact with the ground. The actuator torques of the legs which have no interaction with the ground are calculated directly by comparing the desired joint angles and velocities with the actual ones, i.e. a PD-Position control was applied.

$$T = k_q \cdot (q_{Des} - qr) + d_q \cdot (-dqr) \tag{3.11}$$

The parameters k_q and d_q are experimentally found gains. q_{Des} is the desired angle and qr and dqr are the actual joint angles and rotaion rates.
In order to get a spring-mass system similar to the one simulated by Raibert, the axial force exerted by the leg is calculated by assuming a spring between the virtual hip and the virtual foot. Whereas the virtual hip is simply the position of the main body's CoM, the position of the virtual foot had to be calculated first. Since the position of each leg relative to it's hip should be the same in order to fulfill the conditions for the virtual leg, the position of the virtual foot was calculated as the middle between the two legs in stance. The position of the virtual hip at the beginning of the stance phase was used to calculate the initial length of the spring which stayed constant throughout the whole phase, however the actual length was calculated continuously as the distance between the virtual hip and the virtual foot. The resulting force was:

$$F = k \cdot (||\vec{r}_0|| - ||\vec{r}||) \tag{3.12}$$

Where $\vec{r}_0 = VirtualHip_{touchdown} - VirtualFoot$ and $\vec{r} = VirtualHip - VirtualFoot$. Since we're only interested in the length of the spring and not in the direction, the norm of the vectors was taken. To calculate the direction of the force we simply had to point the force in the direction of the spring, since only axial thrusts were desired.

$$\vec{F} = k \cdot (||\vec{r}_0|| - ||\vec{r}||) \cdot \frac{\vec{r}}{||\vec{r}||} \tag{3.13}$$

$k = 12000[N/m]$ was found experimentally by looking for a realistic behavior of the quadruped.

In the controller, the forces were calculated with the function 'getVirtForceAtCoG' for all 4 legs. Because of that, in the next step we had to choose to which legs these forces are assigned according to the discrete state and then calculate the actuator torques with the virtual model introduced in 3.1.2. Since the forces are chosen in a way that they point in the direction of the legs, the choice of the coordinate system was not important yet as long as it was constistent.

3.2.3 Forward Velocity Control

In this subsection the implementation of the correct foot placement is discussed. To place the foot on the right position, first the vector between the hip and the foot is calculated in world coordinates. This position gets transformed in the local coordinate system, out of which it is possible to calculate the joint angles with inverse kinematics.

The foot placement on the one side controls the forward velocity and the yaw angle of the quadruped, on the other side it is also in charge to make sure that the rules of the virtual leg are not violated. In other words, the foot has to be placed in a way that the horizontal displacement relative to the hip of the foot hitting the ground is equal for all the legs and that the legs hit the ground in unison. The equal displacement is achieved by simply calculate the displacement for the virtual foot and then assign it to the individual ones. To hit the ground in unison, the displacement of each foot relative to the virtual hip in vertical direction has to be the same. Very important here is, that the relative displacement in vertical direction is measured and applied in world coordinates (Figure 3.8)! Like that, the z-coordinate of our foot in world coordinates was set.

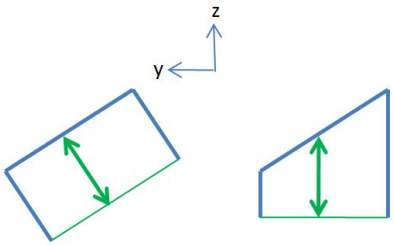

Figure 3.8: Same displacement in z-direction in local (left) and world coordinate frame (right).

For the forward velocity control, the desired foot position was again adjusted in world coordinates. If we would have done that in local coordinates, the positioning in horizontal direction would as well affect the vertical position in the world frame. For the displacement relative to the hip we had to consider the actual forward velocity in main body coordinates, the desired forward velocity and the duration of the stance phase. While the forward velocity was handed over from the model and just had to be transformed into main body coordinates and the desired forward velocity was given, the duration of the stance phase had to be identified. On that account the time between the discrete transitions from 'flight' to 'stance' and from 'stance' to 'flight' was measured each time the simulation performed these transitions. Since like that, the first stance phase couldn't rely on any measured data, an educated initial guess had to be made.

3.2.4 Roll and Pitch Control

For the roll and pitch control a slightly different approach was chosen than the one introduced by Raibert, the main principle allthough stayed the same: In order to balance the roll respectively pitch angle a momentum about the according axis had to be applied during the stance phase. This momentum was caused by additional

vertical forces applied in the hips of the stance legs. In case of pitch control, this force was depending on the actual pitch angle and the pitch angle rate in main body coordinates. Why the body fixed coordinate system was chosen becomes clear when you revolve the quadruped around the z-axis for $\pi/2$. In that case the roll angle in the world coordinates describes the pitch angle in body fixed coordinates and vice versa. The desired force calculated itself as:

$$F_{pitch} = k_{pitch} \cdot \phi_{pitch} + d_{pitch} \cdot \dot{\phi}_{pitch} \tag{3.14}$$

$$F_{roll} = k_{roll} \cdot \phi_{roll} + d_{roll} \cdot \dot{\phi}_{roll} \tag{3.15}$$

The forces found were added to the forces calculated in Section 3.2.2.

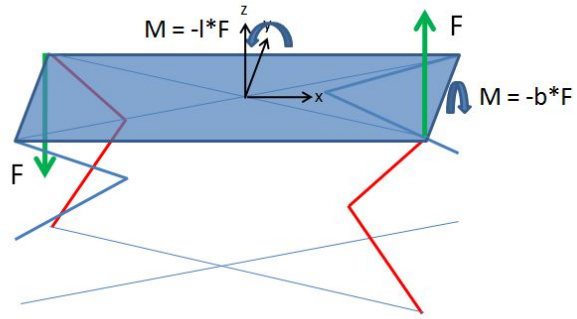

Figure 3.9: An added force creates a momentum about both axis.

Unfortunately, if you just add vertical forces you will also cause a momentum about the axis you don't want to influence. In Figure 3.9 an example is shown: We assume that we want to correct an existing positive pitch angle by adding respectively subtracting forces to the stance legs. We succeed in creating a momentum about the y-axis and will probably be able to correct the pitch angle. Unfortunately also a momentum about the x-axis arises which causes an unwanted roll movement.

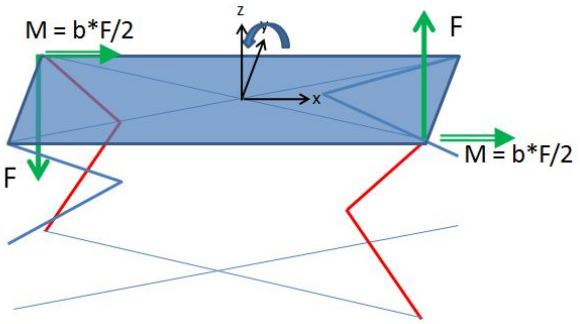

Figure 3.10: An added torque equalizes the created momentum about the roll axis.

To counteract this unwanted momentum, an additional torque was added to the hips which equalized the unintended angular acceleration (Figure 3.10). The quantity could easily be calculated:

$$M_{pitch} = b_{mb} \cdot F/2 \tag{3.16}$$

$$M_{roll} = l_{mb} \cdot F/2 \tag{3.17}$$

If a Force F created to act against a pitch movement is applied, a Torque M_{pitch} must be applied about the x-axis to equalize the angular momentum. The same holds for compensating the momentum caused by a force acting against a roll movement and the y-axis, see equation 3.17. The torques are applied between the leg and the hip and were directly added to the hip abduction respectively hip flection actuator torques calculated earlier.

To spot the right parameter was crucial for this method to work. Only a very small range of values provided a satisfying result. Since at the beginning no satisfying results were found a different attempt with a quadratic approach for the angular velocity was tried. With the quadratic approach it was easier to find values which kept the quadruped trotting for a few secondes, but no good values which stabilized the robot could be found; so we went back to the linear approach (see more in the Section 4). The values found to be the best fitted are $k_{pitch} = 20$, $k_{roll} = 15$, $d_{pitch} = 13$ and $d_{roll} = 7$.

3.2.5 Energy Shaping

Since the system only contains of a spring-mass like model trotting on a ground modeled as a spring damper system, the system constantly loses energy. To act against this phenomena an additional force in vertical direction was added, which was based on a simplified energy shaping approach. The energy shaping method assures that always the same amount of energy is in the system. To do so, the potential and the kinetic energy always were calculated at touchdown. The rotational energy and the kinetic energy of the legs were hereby neglected, since they're very small compared to the total energy.

$$E = m \cdot g \cdot z + \frac{m}{2} v^2 \tag{3.18}$$

The first calculation served as an initial value for the system. Later measurements got compared with the initial one and the difference is calculated, to determine the fortitude of the forces added.

$$F_E = \Delta E \cdot (||\vec{r}_0|| - ||\vec{r}||) \cdot c \quad , \tag{3.19}$$

with ΔE the calculated energy difference and c an experimentally determined constant. The force was added to the hips in stance phase in vertical direction.

Chapter 4

Results

4.1 General Results

The results presented in this section are mainly about stabilizing the robot around the equilibrium point $\dot{x} = \dot{y} = \Theta_{pitch} = \Theta_{roll} = 0$. Therefore the reaction of the controller on several different disturbances was observed. To optimize the controller and especially the parameters used, two different approaches were used. First qualitative judging was done: By looking at the simulation you could easily tell whether it was stable or not. This approach was used in the early phase of the optimization, when the scales of the parameters weren't known yet and quite often the model was unstable and crashed. The hereby found parameters already delivered satisfying results, it was possible to trot without forward velocity and also disturbances were successfully corrected. In the Figures 4.1, 4.2, 4.3 and 4.4 you can see how the model reacts on disturbances in certain directions. Figure 4.1 shows the roll angle over time, with an initial disturbance of 0.1 rads. You can see that the system stabilizes pretty well. In the Figure 4.2 the pitch angle is shown with a disturbance of 0.1 rads, in 4.3 the velocity in x-direction and in 4.4 the velocity in y-direction, always with a disturbance of 0.1 m/s. The parameters used here were:

Parameter	Empirical Results
d_{pitch}	13
k_{pitch}	20
d_{roll}	7
k_{roll}	15
k_{foot}	0.044

For the complete results with all the graphs, please consult appendix C.

Once satisfying values were found with this method an Eigenvalue Analysis was done to optimize specific parameters further.

4.2 Eigenvalue Optimization

We decided to look at the parameters for the pitch and the roll control, k_{pitch}, d_{pitch}, k_{roll} and d_{roll}. The system was linearized around the equilibrium point $\dot{x} = \dot{y} = \Theta_{pitch} = \Theta_{roll} = \dot{\Theta}_{pitch} = \dot{\Theta}_{roll} = 0$. To optimize the stability, the six state variables mentioned above were considered, the yaw angle and the position of the main body, as well as the vertical velocity were not considered here. The two discrete states regarded were defined as the initial position and the after next apex height

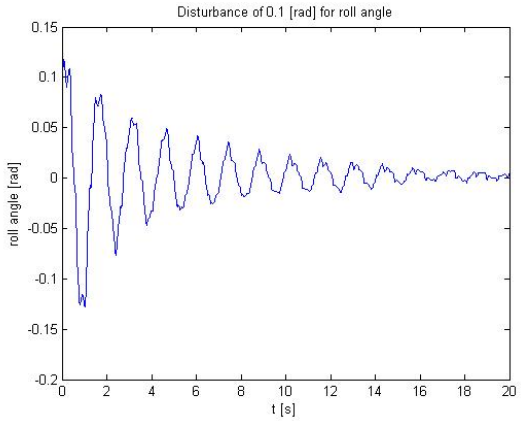

Figure 4.1: Disturbance in the roll angle of 0.1 rad.

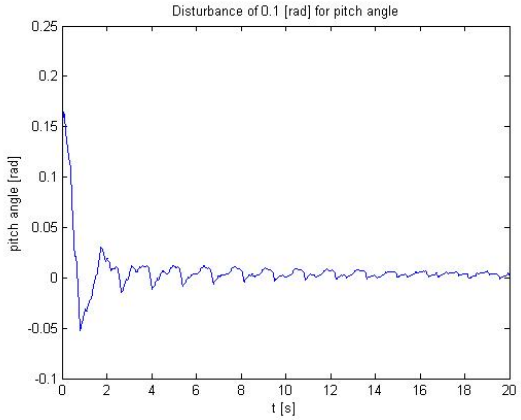

Figure 4.2: Disturbance in the pitch angle of 0.1 rad.

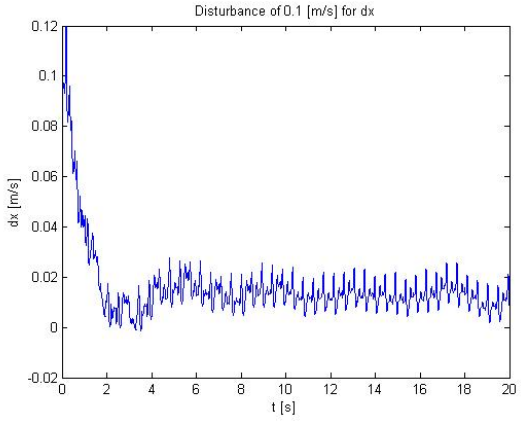

Figure 4.3: Disturbance in the forward velocity of 0.1 m/s.

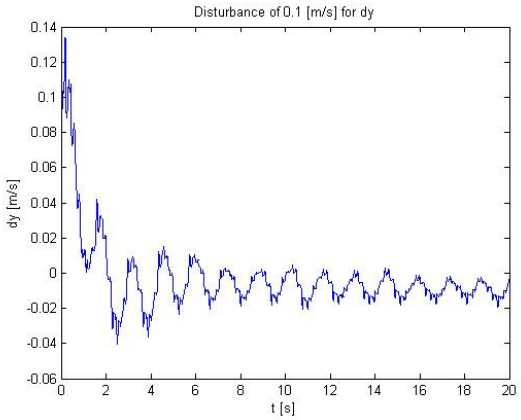

Figure 4.4: Disturbance in the sidewards velocity of 0.1 m/s.

(Figure 4.5). Like that it was assured that the model was in the same discrete state again and the two positions are comparable.

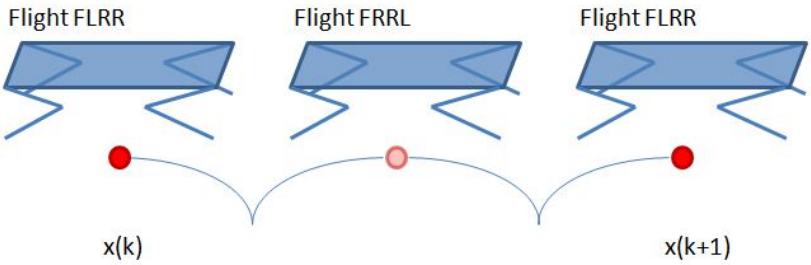

Figure 4.5: Only every other apex point is considered.

Since we linearized around the zero point, the problem could be stated as:

$$\Delta x_{k+1} = A \cdot \Delta x_k \tag{4.1}$$

A is hereby the linearization matrix and Δx is a vector containing the difference of our state variables to our equilibrium point. If the eigenvalues of A have a magnitude smaller than 1, the system is considered to be stable. The smaller the eigenvalues the better, so the goal of the optimization was to get the eigenvalues as small as possible. We were looking for the configuration where the spectral radius is the smallest.

To be able to test a big amount of different parameters a little script was written which ran the simulation under different condition and calculated the eigenvalues independently. The best configuration found after several loops was with the parameters:

Parameter	Empirically found parameters	After EV - optimization
d_{pitch}	13	3
k_{pitch}	20	20
d_{roll}	7	2
k_{roll}	15	8
$\|\lambda\|\infty$	2.5760	0.9349

The biggest Eigenvalue with these parameters was $\lambda_{max} = 0.7719 + 0.5274i$. As can be seen from the results above, since the biggest Eigenvalue has a magnitude smaller than one, the system is considered to be stable. Running the simulation with these values with a disturbance of 0.1 rad in the pitch angle, delivers the results shown in the Figures 4.6 and 4.7.

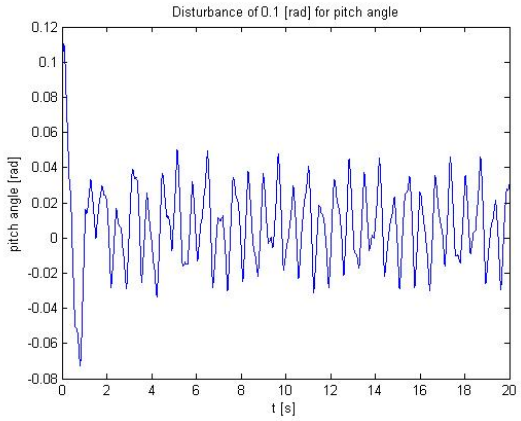

Figure 4.6: Disturbance in the pitch angle of 0.1 rad.

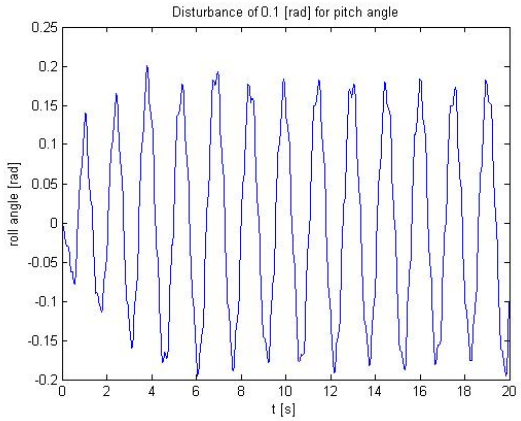

Figure 4.7: Disturbance in the pitch angle of 0.1 rad.

As can be seen in the graphs, the angles still tend to oscillate heavily. To fix this issue, we also tried to optimize the parameters for the forward velocity control k_{foot} (see Section 3.2.3). Since also with the optimization of the parameters for the forward velocity no better values could be found, we went back to using the parameters empirically found, which deliver by far the best results.

There might be several reason why the eigenvalue analysis didn't work out. First of all the equilibrium point is not validated: Even though the system stays stable when starting at this point, a slight oscillation can still be observed. Other reasons could be the insufficient accuracy of the solvers and time steps used or the too big interval of linearization.

4.3 Forward Trotting

With the parameters found in the section 4.1 also the behaviour while forward trotting was examined. The fastest stable velocity in forward direction was found to be around 0.2 m/s. At this speed the control of the quadruped was still possible, eventhought with a certain bias. In Figure 4.8 a graph is illustrated where you can see the forward velocity plotted over time. The controller was set to stay at zero velocity for 2 seconds, then walk forward with a speed of 0.2 m/s for 10 seconds and the go back to zero velocity. Since the yaw angle was not controlled, the velocity in y-direction was controlled in a way that it keeped on walking in a straight line with respect to the world coordinate system.

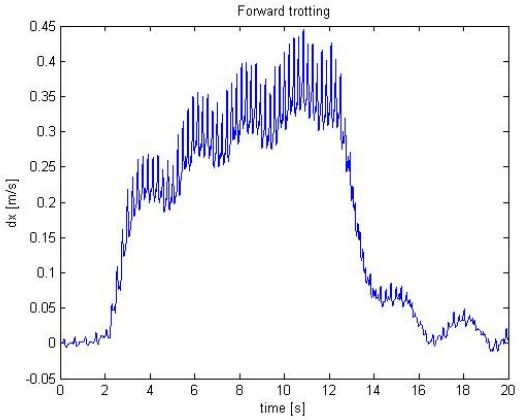

Figure 4.8: Forward trotting for 10 seconds.

A big oscillation of the forward velocity can be observed, but it can be explained as the natural behaviour of a spring mass model. Unfortunately, on top of this fluctuation also a quite big drift is observable, the longer the quadruped trots forward, the higher the velocity.

Chapter 5

Interpretation

5.1 Model

The model worked reliable during the simulations, and the simulation time was reasonable. It was made simple to implement several different controllers to the model, so the overall judgment was positive. Some points that might be improved can be seen in the next subsection.

5.1.1 To do

Due to the fact that a lot of simulations with different parameters were done, the simulations tolerance was held pretty high in order to keep the simulation time short. To make more diagnostically conclusive simulations, the tolerance has to be adjusted more precicly, which would on the other hand encrease the simulation time drastically.

Another point of the model which might still be improved is the ground contact model. Since a simple approach for the nonlinear spring damper system was chosen here, the discontinuities of the ground contact forces at impact are still existent. With a different approach, as it was introduced in Section 2.4 this inaccuracy might be inhibited.

5.2 Controller

The performance of the controller was satisfying, specially because it is at the same time a very simple, but still robust controller. The implementation of the Raibert three part controller was possible with a reasonable amount of work. The biggest problem that occured was the determination of the best fitted parameters for the control algorithm.

The parameters found empirically stabilize the model very good in case of the roll, pitch and altitude control. The model can be held in a stable position for an arbitrary long time and it reacts appropriate on disturbances and brings the model back to the equilibrium point within only a few steps.

The forward velocity control worked satisfying. The model can be kept in the same position for a long time and only a small drift of the position can be observed. Unfortunately the control of the velocity when trotting forward might still be improved. There a visible difference between the desired and the actual velocity can

be observed.

The yaw control was not implemented during this work.

With the eigenvalue analysis, satisfying parameters could not be found. Possible reasons for this failure are listed in section 4.2.

5.2.1 To do

As a first step I would suggest to implement the yaw control. The implementation of the yaw control is already done to a certain point and a completion of this algorithm seems promising, since it's based on the same method as the forward velocity control. Again, enough time should be calculated for the determination of the right parameters.

The completion of the control for a stable forward trotting might also be done in a reasonable amount of time and also seems very promising. The biggest problem one might face there, are the occurring additional momentums related with the foot placement, which might make an additional control component inevitable.

Chapter 6

Acknowledgment

With the completion of this project, I like to express my sincere thanks to all persons who contributed to this work. First of all I'd like to thank my supervisors Marco Hutter and David Remy for their competent support at all times throughout the whole work. I specially appreciate their flexibility in rescheduling my work due to my temporary disability. I also like to thank Roland Siegwart and the Autonomous System Lab for making this project possible and allowing me to get new insights and knowledge in the field of robotics.

Appendix A

Files

The final simulation consited of the following files:

- **ScarpETH.mdl:** In this file the complete model is defined, inclusive ground contact forces and wrapper for the control algorithm.
- **ctrl_VM_trot_dn.m:** In this file the control algorithm is defined, inclusive all the parameters needed for the controller, such as initial guess for the stance time duration, several gains for the pitch, roll and forward velocity control, constants for the energy shaping method, desired foot positions during the flight phase, etc.. The explaination to the variables and function executed are written as comments in the file.
- **param_all_dn.m:** This is the file to call the parameter files 'param_mech_dn.m' and 'param_ctrl_dn.m'.
- **param_mech_dn.m:** In this file the physical properties of the quadruped (as introduced in section 2.2) are described. Furthermore also the parameters for the ground contact model and the physical enviroment are defined here.
- **param_ctrl_dn:** In this file the initial condition for the simulation are defined. Further the sampling frequenz of the controller is defined here.
- **The utilities folder:** In this folder external function for coordinate transformations and inverse kinematics are stored. These utilities were implemented by M. Hutter and were only slightly adapted.

Appendix B

How to use

To run the simulation with the implemented controller these steps have to be followed:

1. Open the file 'ScarpETH.mdl'.
2. Open the file 'param_all_dn.m' and press execute.
3. Press the 'Start simulation' button in the model 'ScarpETH.mdl'.

To implement your own controller in the model follow these steps:

1. Open the file 'ScarpETH.mdl'.
2. Open the block 'ctrlWrapper'. (colored in magenta)
3. Follow the instruction in line 5 and 30.
4. Open the file 'param_all_dn.m' and press execute.
5. Press the 'Start simulation' button in the model 'ScarpETH.mdl'.

Appendix C

Results

For reasons of completness, here are the graphs for the roll and pitch angle, as well as forward and sidewards velocity reacting to different disturbances with the parameters introduced in Section 4.1.

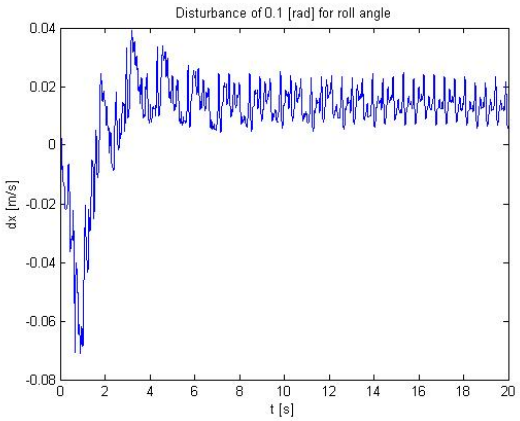

Figure C.1: Disturbance in the roll angle of 0.1 rad.

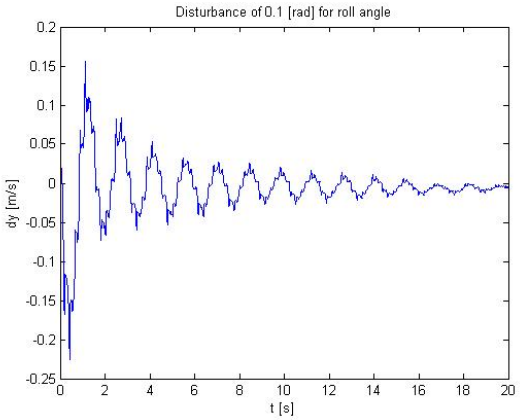

Figure C.2: Disturbance in the roll angle of 0.1 rad.

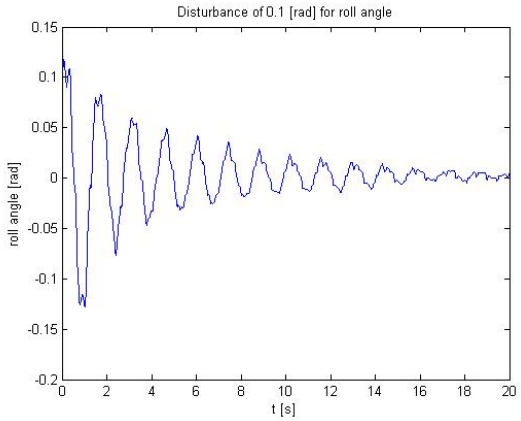

Figure C.3: Disturbance in the roll angle of 0.1 rad.

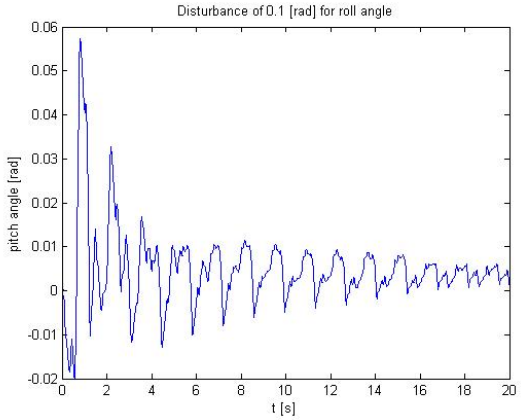

Figure C.4: Disturbance in the roll angle of 0.1 rad.

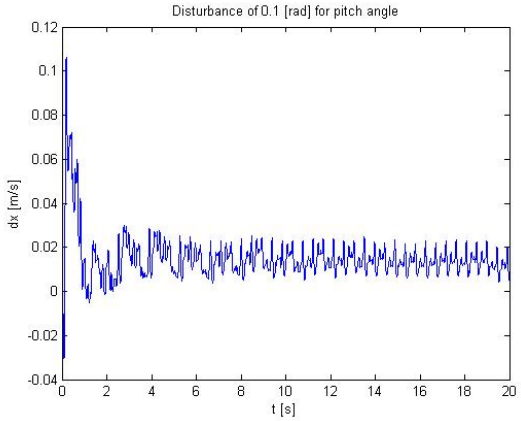

Figure C.5: Disturbance in the pitch angle of 0.1 rad.

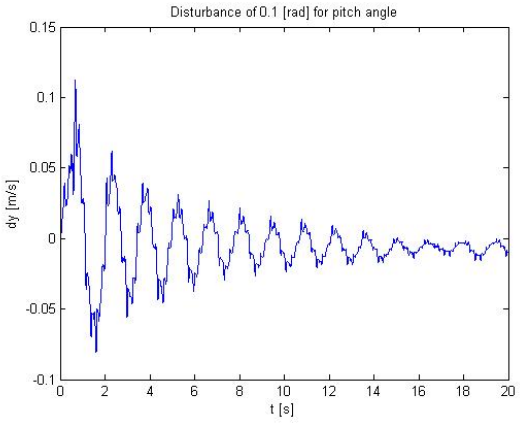

Figure C.6: Disturbance in the pitch angle of 0.1 rad.

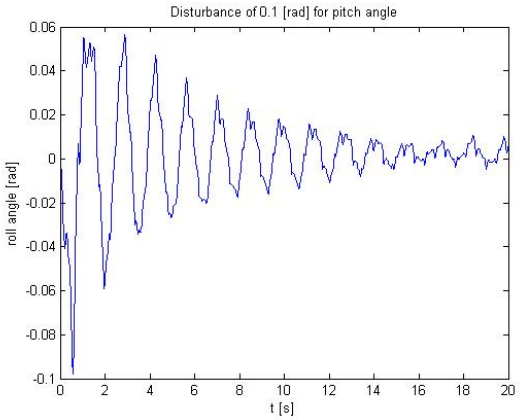

Figure C.7: Disturbance in the pitch angle of 0.1 rad.

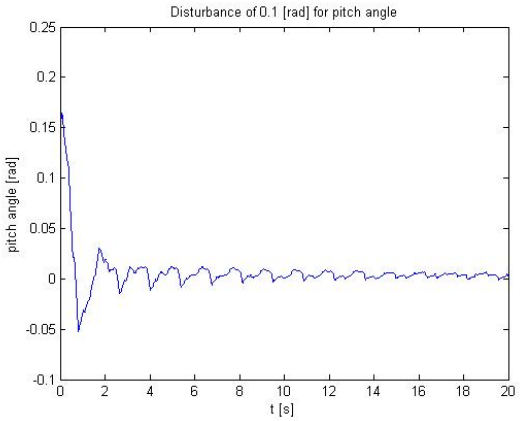

Figure C.8: Disturbance in the pitch angle of 0.1 rad.

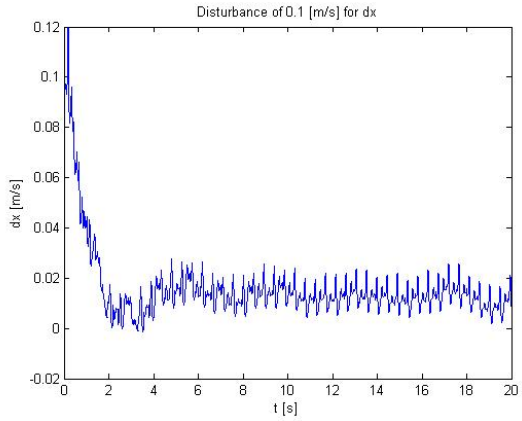

Figure C.9: Disturbance in the forward velocity of 0.1 m/s.

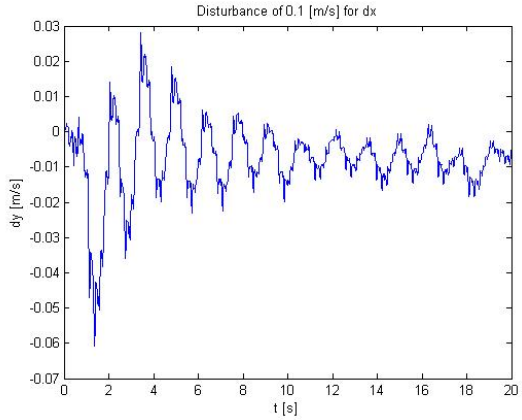

Figure C.10: Disturbance in the forward velocity of 0.1 m/s.

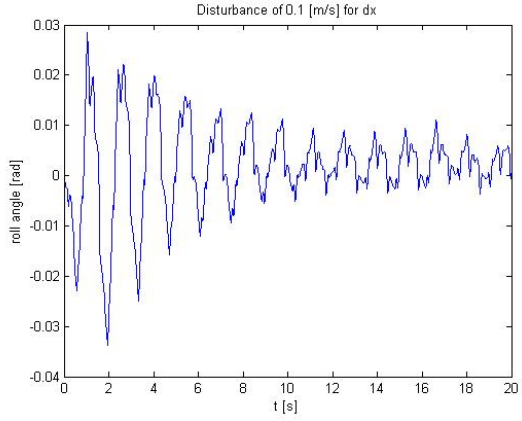

Figure C.11: Disturbance in the forward velocity of 0.1 m/s.

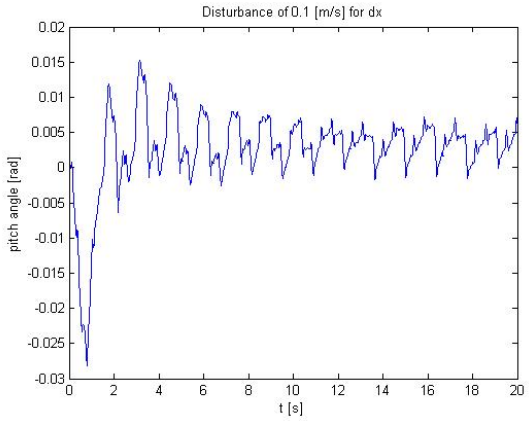

Figure C.12: Disturbance in the forward velocity of 0.1 m/s.

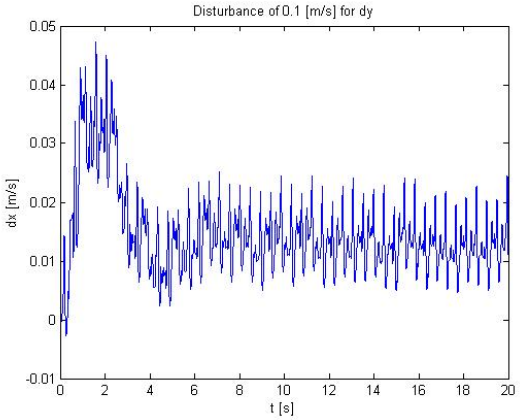

Figure C.13: Disturbance in the sidewards velocity of 0.1 m/s.

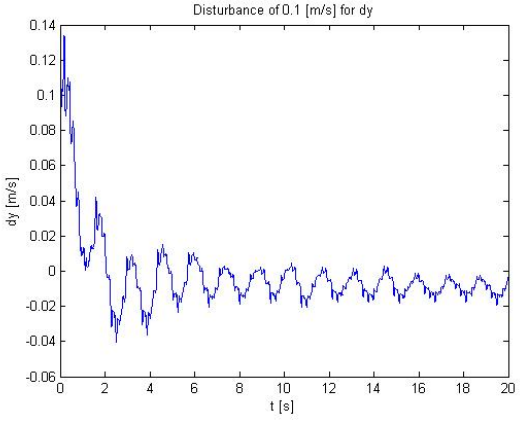

Figure C.14: Disturbance in the sidewards velocity of 0.1 m/s.

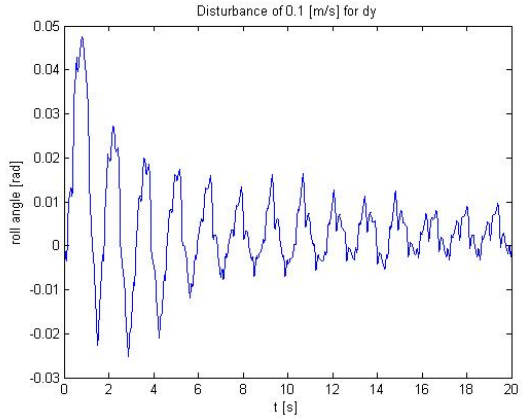

Figure C.15: Disturbance in the sidewards velocity of 0.1 m/s.

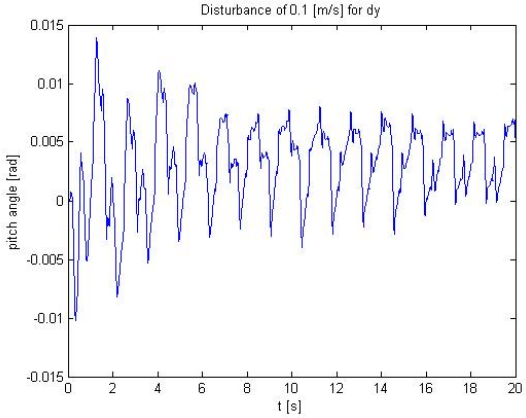

Figure C.16: Disturbance in the sidewards velocity of 0.1 m/s.

Bibliography

[1] MathWorks Official Homepage: *http://www.mathworks.com/products/simmechanics/*

[2] P. Omlin, C. D. Remy, M. Hutter: *Implementation of Optimal Control Strategies for a Hopping Leg*, 2010.

[3] M. Silva, J. A. Tenreiro Machado, I. S. Jesus: *Modeling and simulation of walking robots with 3 DOF legs*, Institute of Engineering Porto, 2006.

[4] M. Silva, J. A. Tenreiro Machado, A. Lopes: *Modelling and simulation of artificial locomotion systems*, Cambridge University Press, 2005.

[5] D. W. Marhefka, D. E. Orin: *Simulation of Contact Using a Nonlinear Damping Model*, International Conference on Robotics and Automation, 1996.

[6] M. Sobotka: *Hybrid Dynamical System Methods for Legged Robot Locomotion with Variable Ground Contact*, Lehrstuhl für Steuerungs- und Regelungstechnik Technische Universität München, 2007.

[7] J. K. Mills, C. V. Nguyen: *Robotic Manipulator COLLISIONS: modelling and Simulation*, ASME Journal of Dynamic Systems, Measurement and Control, 1992.

[8] M. H. Raibert, H. B. Brown, Jr. M. Chepponis, E. Hastings, J. Koechling, K. N. Murphy, S. S. Murthy, A. J. Stentz: *Dynamically Stable Legged Locomotion*, The Robotics Institute and Department of Computer Science Carnegie-Mellon University Pittsburg, 1983.

[9] M. H. Raibert: *Trotting, Pacing and Bounding by a Quadruped Robot*, Artificial Intelligence Laboratory, Massachusetts Institute if Techology, Cambridge, 1990.

[10] H. Geyer: *Simple Models of Legged Locomotion based on Compliant Limb Behavior*, Fakultät für Sozial- und Verhaltenswissenschaften, Friedrich-Schiller-Universität, Jena, 2005.

[11] L. R. Palmer III: *Intelligent Control and Force Redistribution for a high-speed Quadruped Trot*, The Ohio state university, 2007.

[12] A. Lauber: *Virtual Model Control on ALoF*, ETH Zurich, 2011.