ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bio-Inspired Robotics Lab
Prof. Dr. Fumiya Iida

Semester-Project

Dynamics of a Free Vibration Based Robot

Autumn Term 2012

Supervised by:
Xiaoxiang Yu

Author:
André Foehr

Abstract

This work presents a novel way of reducing the search space for the optimization of a dynamic system, in this case of the beam of a Curved Beam Hopper (CBH). Note that this method could also be applied to other similar systems. This reduced search space has the enormous advantage, that the computational burden decreases significantly, especially since the simulation of most dynamic systems takes long.

After introducing the CBH, the model thereof is explained in detail. The robot is simulated using a SimMechanics simulation. This simulation discretizes the beam structure of the CBH into different pieces. Most importantly, it relies on an approximation using the Euler Bernoulli beam theory to calculate the discretization parameters of any given beam shape. The ground interaction model used is elastic with damping.

Furthermore the technique to reduce the search space is explained, it uses basic properties from dynamics and geometry to significantly reduce the search space of the optimization. The idea is that rather than just changing one parameter at a time, multiple parameters of the systems are changed using a single iteration parameter.

Then, the performance of the presented technique is discussed and it is shown that even with very few optimization steps, good performance is achieved.

Finally an alternative approximation of the beam structure is outlined in the appendix; this method could be used to improve the approximation of the stiffness of the different discretization elements.

Acknowledgements

I would like to thank Xiaoxiang Yu, for giving me the creative freedom I needed in order to complete this semester project and for the all the time he took to have lengthy discussions.

I would also like to thank Prof. Iida for welcoming me to his laboratory and giving me the opportunity to work on such an interesting system.

Contents

Acronyms, Abbreviations and Symbols

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
CBH	Curved Beam Hopper
GC	Ground Contact (Model)

Symbols

Beam Structure

$a/2$	Depth of the beam shape
a	Width of the ellipse
b	Height of the beam shape / height of the ellipse
t	Thickness
w	Width
l_{elem}	Length of the discretization element
m_{elem}	Mass of the discretization element
I_{elem}	Moment of inertia of the discretization element
k_{elem}	Stiffness coefficient of the discretization element
d_{elem}	Damping coefficient of the discretization element

Top

M_{top}	Mass of Top
m_{rot}	Rotating mass
r_{rot}	Rotation radios of rotating mass
ω_{rot}	Rotational speed

Foot

L_{foot}	Length
W_{foot}	Width
M_{foot}	Mass
I_{foot}	Moment of inertia

GC Model

μ_{slide}	Sliding friction coefficient (dry)
μ_{stick}	Static friction coefficient (dry)
k_i	Stiffness coefficient in i-direction or i-plane
d_i	Damping coefficient in i-direction or i-plane
F_i	Force in i-direction or i-plane
$stickynessVal$	Stickyness coefficient of the GC model

Optimization

ϵ	Iteration parameter
ξ	Second iteration parameter used to circumvent nonlinearities

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

On one hand the inspiration for research in a certain field comes from the research that shows there is great room for innovation in that field. The greatest advantage of legged robots, compared to wheeled robots, is their versatility. Some of the first running robots were introduced by Raibert in **"Legged Robots That Balance"**[1], which later led to BigDog[2]. McGeer achieved robots with very high energy efficiency in **"Passive dynamic walking"**[3]. Alexander explains what are the most important factors in animal locomotion and their origins in his book **"Principles of Animal Locomotion"**[4]. Furthermore Pfeifer gives interesting ideas of what the intelligence of a system is, in **"How the body shapes the way we think"**[5].

On the other hand, motivation for research in a certain field is that in todays world, scientists have to develop systems, which achieve high performance, with high flexibility and high energy efficiency. When comparing the different existing systems it is obvious that the curved beam hopper has similar properties to the passive dynamic walker, i.e. it has a high energy efficiency and virtually no control. Yet, the curved beam hopper has the advantage that multiple gaits or behaviors could be achieved by exciting different eigenmodes of the system.

1.2 The Curved Beam Hopper

The Curved Beam Hopper (CBH) presented in this work consists of a foot, a leg (beam structure), a head and a rotating mass, as can been seen in figure 1.1. The presented CBH is very easy and cheap to build, very light and energy efficient, and requires no complex control structures in order to move.

[1] [Raibert, 1985]
[2] [Raibert et al., 2008]
[3] [McGeer, 1990]
[4] [Alexander, 2003]
[5] [Pfeifer and Bongard, 2006]

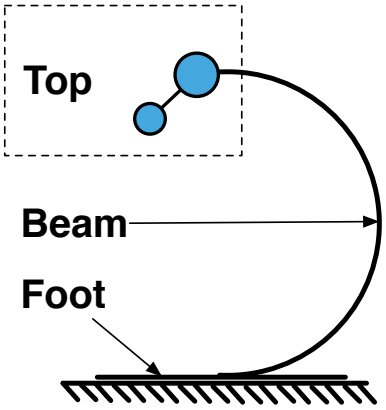

Figure 1.1: The different parts of the CBH

The beam structure itself acts as a 3-dimensional spring between the foot and the head. The head itself is composed of a main mass and a smaller one, which rotates around the main mass, using an actuator. This actuator pumps the energy into the system.

1.3 Goal

The goal chosen for this semester project, is to optimize the parameters of the beam structure for maximal average speed. This goal was chosen because the CBH is inherently energetically efficient due to its low unsprung mass. The fact that we maximize the average speed means that we require the system to be stable for at least some time.

1.4 Previous Work

The CBH is a concept being investigated by the BIRL at ETH Zurich. Different approaches have been followed, such as a C-shaped[6] and a U-shaped[7] robot. The simulation of this semester project is mostly based on a master thesis[8], in which the shape of a given beam was optimized using a genetic algorithm. In a more recent project[9], a control concept was explored using CPG-inspired control mechanisms.

1.5 Outline

The second chapter explains in detail how the simulation works and what main improvements have been done; and explains some of its drawbacks as well. The third chapter outlines the mathematical approach taken to optimize the system. The results are discussed in the fourth chapter and finally a conclusion is given in chapter five.

[6] [Reis and Iida, 2012]
[7] [Reis et al., 2013]
[8] [Näf, 2012]
[9] [Lataniotis, 2013]

Chapter 2

Simulation

2.1 SimMechanics Model

2.1.1 Introduction

The system is modelled using multiple rigid bodies. The beam structure is a deformable structure, this is why it is approximated by multiple small elements with spring-damper elements connecting them. The ground is approximated using spring-damper elements with the same properties as in the previous work[1]. Figure 2.1 shows the general layout of the simulation and how it relates to the physical model. The coordinate system used is a right handed coordinate system and it is defined as follows:

x-axis	Points in forward running direction
y-axis	$\perp$ to x and z-axis
z-axis	Height

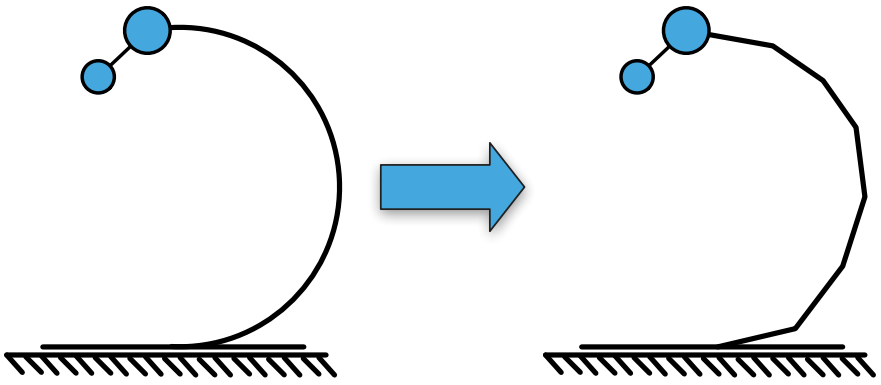

Figure 2.1: Transition from Robot to Simulation

The configuration parameters of the SimMechanics simulation are shown in table 2.1. Note that the ODE solver used is different from the one used in the previous simulation[2], this is possible because of the new ground contact model.

[1] [Näf, 2012]
[2] [Näf, 2012]

Next the individual parts of the simulation will be discussed. The SimMechanics files can be found in appendix C.

Table 2.1: Configuration parameters Simulink

Environment	SimMechanics 1
Analysis mode	Forward dynamics
Step size	Variable
Min step	Auto
Max step	1e-4
Rel. tol.	1e-4
Solver	ODE 23s

2.1.2 Foot

The foot is modeled as one rigid body, its properties can be found in table 2.2. Note that the beam structure is fixed in the middle of the foot. Also the foot is modeled as a thin plate identical to the previous simulation[3].

Table 2.2: Parameters foot

Property	Name	Value	Unit
Length	L_{foot}	0.275	[m]
Width	W_{foot}	0.248	[m]
Mass	M_{foot}	0.073	[kg]
Moment of Inertia	I_{foot}	$\frac{1}{12} \cdot M_{foot} \begin{pmatrix} W_{foot}^2 & 0 & 0 \\ 0 & L_{foot}^2 & 0 \\ 0 & 0 & (W_{foot}^2 + L_{foot}^2) \end{pmatrix}$	[kg·m^2]

2.1.3 Beam Structure

The beam structure is approximated using 16 discretization elements, as shown in figure 2.1. These elements are composed of one rigid body element and one spring-damper element, as shown in figure 2.2. For the beam structure it is assumed that the stiffness and the mass of an element can be calculated using a Euler-Bernouilli beam theory approximation, as detailed in table 2.3, missing values are the parameters that are not fixed during the optimization. The damping in the beam structure is minimal and constant over all optimizations, but substantially improves computational time.

The beam structure is chosen to be an ellipse cut in two along the major or minor axis. However the framework allows for any curve to be used and if done wisely, the approximation of other curves should result in a similar performance.

[3] [Näf, 2012]

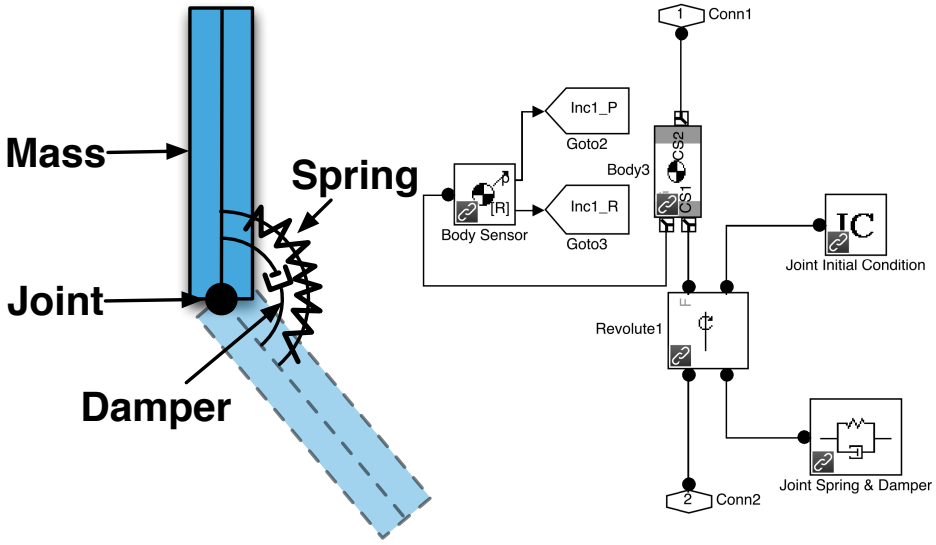

Figure 2.2: Structure of a leg element as well as SimMechanics blocks

Table 2.3: Parameters beam element

Property	Unit	Name	Formula
Length	l_{elem}	-	[m]
Width	w_{elem}	-	[m]
Thickness	t_{elem}	-	[m]
Mass	m_{elem}	$\rho \cdot l_{elem} \cdot w_{elem} \cdot t_{elem}$	[kg]
Moment of Inertia	I_{elem}	$\frac{1}{12} \cdot m_{elem} \begin{pmatrix} 0 & 0 & 0 \\ 0 & l_{elem}^2 & 0 \\ 0 & 0 & l_{elem}^2 \end{pmatrix}$	[kg·m^2]
Stiffness constant	k_{elem}	$\frac{E \cdot w_{elem} \cdot t_{elem}^3/12}{l_{elem}}$	[Nm/rad]
Damping constant	d_{elem}	.003	$[\frac{\text{Nm}}{\text{rad/s}}]$

2.1.4 Head

The top part of the robot is composed of two parts, one being the heavy mass or the head itself, the other one being the light mass which rotates around the head, as shown in figure 2.3. The heavy mass is assumed to distributed on the top element of the beam and there is a torque actuator between the top mass and the rotating mass. The actuation torque is calculated as follows:

$$T = k_{ctrl}(\omega_{des} - \omega(t)) \qquad \text{with: } k_{ctrl} = \text{ control gain}$$

The control gain is chosen to be $k_{ctrl} = 0.1\frac{\text{Nm}}{\text{rad/s}}$. The other parameters for the top part of the robot are in table 2.4, E is Young's modulus and is a material constant; in the case of this simulation aluminum was chosen. Missing values are the parameters that are not fixed during the optimization.

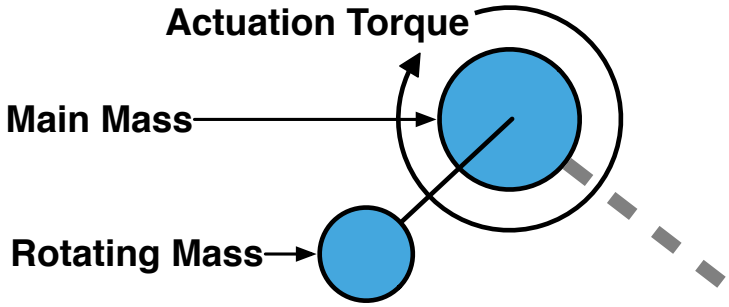

Figure 2.3: Detail of the top

Table 2.4: Parameters of the top

Property	Name	Value	Unit
	Top Mass		
Length	l_{elem}	-	[m]
Width	w_{elem}	-	[m]
Thickness	t_{elem}	-	[m]
Mass	M_{top}	0.088	[kg]
Moment of Inertia	I_{top}	$\frac{1}{12} \cdot M_{top} \begin{pmatrix} 0 & 0 & 0 \\ 0 & l_{elem}^2 & 0 \\ 0 & 0 & l_{elem}^2 \end{pmatrix}$	[kg·m^2]
Stiffness constant	k_{elem}	$\frac{E \cdot w_{elem} \cdot t_{elem}^3/12}{l_{elem}}$	[Nm/rad]
Damping constant	d_{elem}	0.003	$[\frac{\text{Nm}}{\text{rad/s}}]$
	Rotating Mass		
Mass	M_{rot}	.0334	[kg]
Mass link	M_{link}	.008	[kg]
Rotation radius	r_{rot}	.035	[m]
Rotation speed	ω_{rot}	-	[rad/s]

2.1.5 Ground Interaction Model

While the a contact point is in ground contact, the proposed ground contact (GC) model can be in two different states, either sliding or sticking to the ground. In either case the out of plane reaction force is modeled using a spring-damper element. The out of plane force can be calculated as follows:

$$F_z = -k_z \cdot z - d_z \cdot \dot{z} \qquad \text{where: } z = \text{ height}$$

The force F_z is limited such that it can only push away from the ground and not pull into the ground. In order to avoid unwanted switching behavior of the GC, the following conditions have been chosen, if true the state is stance. Note that the variable $stickynessVal$ makes the GC model stay in GC, even if it lifts off the ground a bit; this is common practice and helps to avoid unnecessary switching.

$$(z < stickynessVal \quad \& \quad \text{State}_{old} = \text{"Stance"}) \qquad \text{or} \qquad z < 0$$

Figure 2.4 shows the different parts of the ground contact model.

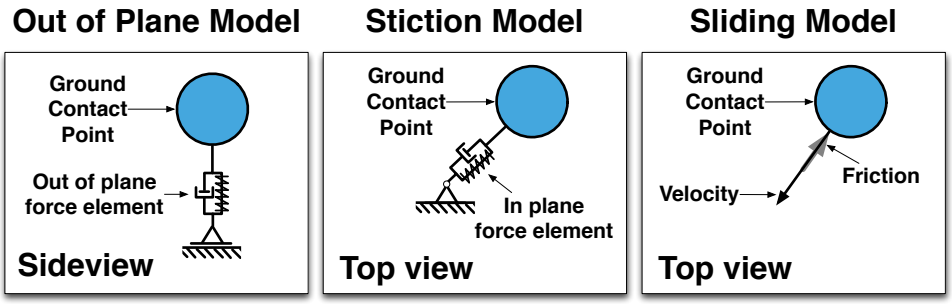

Figure 2.4: Simplified conceptual representation of the different parts of the GC

Sticking model During stiction, the ground reaction force in the plane of the ground is approximated as a spring-damper element, whose position is fixed using a free rotational joint. The force element has a stiffness coefficient k_{xy} and a damping coefficient d_{xy}. Both parts of the force are calculated from the magnitude of deflection respectively the magnitude of the speed, note that the component of each force is projected onto the relevant magnitude vector.

$$F_x = -k_{xy} \cdot \sqrt{dx^2 + dy^2} \cdot \cos(\alpha_{defl}) - d_{xy} \cdot \sqrt{\dot{dx}^2 + \dot{dy}^2} \cdot \cos(\alpha_{speed})$$

$$F_y = -k_{xy} \cdot \sqrt{dx^2 + dy^2} \cdot \sin(\alpha_{defl}) - d_{xy} \cdot \sqrt{\dot{dx}^2 + \dot{dy}^2} \cdot \sin(\alpha_{speed})$$

$$\text{With:} \quad \alpha_{defl} = \text{atan2}(dy, dx) \quad \text{and} \quad \alpha_{speed} = \text{atan2}(\dot{dy}, \dot{dx})$$

If the following statement is true, switch to sliding state.

$$\sqrt{F_x^2 + F_y^2} > \mu_{stick} \cdot F_z$$

Sliding model While the GC is in the sliding state, the ground reaction force in the plane of the ground is approximated using the coulomb sliding law. The magnitude of the force is only dependent on the normal force, while the direction is dependent on the direction the GC point is moving.

$$F_x = -\mu_{slide} \cdot F_z \cdot \cos(\alpha_{speed})$$

$$F_y = -\mu_{slide} \cdot F_z \cdot \sin(\alpha_{speed})$$

If the following statement is true, switch to sticking state.

$$\mu_{slide} \cdot F_z > d_{xz} \cdot \sqrt{\dot{dx}^2 + \dot{dy}^2}$$

Parameters The parameters relevant to the GC model are listed in table 2.5.

Table 2.5: Parameters ground contact model

Name	Value	Unit
μ_{slide}	0.4	[-]
μ_{stick}	0.5	[-]
k_z	140	[Nm/rad]
d_z	15	$[\frac{\text{Nm}}{\text{rad/s}}]$
k_{xy}	20	[Nm/rad]
d_{xy}	8	$[\frac{\text{Nm}}{\text{rad/s}}]$
$stickynessVal$	1e-4	[m]

2.1.6 Discussion

The simulation is computationally expensive, i.e. 10 seconds of simulation time take over 5 minutes to simulate. This obviously is a significant disadvantage, while one might consider the large number of bodies the reason for this, the more important issue is the fact that the mass of each leg element is very small compared to the stiffness of the leg elements.

The initial approach of this semester project was to approximate the beam structure in a massless approximation, as is explained in appendix B; however it was not successful. Using the simulation detailed in this chapter, a new approach was taken; as is elaborated in chapter 3.

The Euler-Bernouilli beam approximation mentioned in subsection 2.1.3, is analyzed in Appendix A.

2.2 Matlab Environment

In order to analyze the system a Matlab framework was developed that varies different parameters, calls the SimMechanics model and finally assesses the performance of the system for the given parameters.

As mentioned in the previous section of this chapter, most of the parameters are fixed. The actual physical and control parameters of the robot which are changed are the width and thickness of the beam as well as it's size and shape and finally the rotational speed of the rotating mass and to a lesser extent the angle between the foot and the beam structure. This in turn affects the approximation of the beam structure, by varying the stiffness, mass and length of each beam element.

The beam structure is described by an arbitrary curve, in this case an ellipse. In order to discretize it an algorithm is used. A set of points on the curve is generated and the points which are the closest to being equally distant to each other, while reaching the end of the curve, are chosen. These points are then used to compute the length of each element as well as the initial angles between the elements. The details of the implementation can be found in appendix D .

As performance criterion the speed of the robot was chosen; it is calculated by dividing the distance traveled in the last 8 seconds of a 10 second simulation by the considered 8 seconds. This method avoids taking initial perturbations into consideration and gives a better estimate of the steady state performance. Note that since the initial parameters of the simulation gave a non-zero performance, this makes it possible to change parameters and observe how it affects the performance, which finally means that the optimization task is simplified.

Chapter 3

Proposed Approach

3.1 Challenges and Possible Approaches

The optimization parameters are the width and thickness of the beam as well as its size and shape and also the rotational speed of the rotating mass and finally, less important, the offset angle of the beam at the base or the foot. This gives us and optimization space, which is at least 6-dimensional.

As mentioned in subsection 2.1.6 the computation of the simulation takes very long, this in turn means that the optimization task is very difficult because of the combination of a high dimensional search space and the computational burden of a single simulation.

In order to enable an optimization in a reasonable amount of time, the total computational burden has to be reduced. Even with only 10 values in each dimension, a total computation time of $5 \cdot 10^6$ min has to be expected, which is completely unrealistic. Two possibilities exist, either the computation time is reduced significantly or the search space is reduced significantly, without loosing much if any performance in the physical system. While these possibilities are not mutually exclusive, they have been investigated separately. The reduction of the computation time was first examined, unfortunately it did not work in a satisfactory manner, it can be found in appendix B.

3.2 Reducing Search Space

Because the initial values give a performance which is strictly greater then 0, it is possible to optimize using a gradient ascent algorithm, this can approximately give local maxima. However this approach causes problems such as choosing a step size and the fact that we would have to compute the gradient of the performance in a 6-dimensional space. In order to circumvent this, a slightly different approach has been chosen; a set of simulations is run, where only one single parameter is changed, then the parameter that gave the best results is chosen. This works similarly to the gradient ascent algorithm, but the convergence should generally be slower. However it gives more of an insight into the system and does not require the computation of derivatives.

The convergence of this optimization while being run sequentially for the physical parameters of the system, is very slow. This is due to the dependence of different parameters. For instance if one changes the thickness of the beam structure, the

optimal rotational frequency changes as well. The next step is to change multiple physical parameters at once, which in turn is like changing one parameter that changes the characteristic behavior of the system in a more sensible way. These will be called tasks from now on.

There are 5 different optimization tasks, of which 4 will be discussed in this section. Most of these tasks change multiple parameters at once in a dependent manner.

- **Rotational speed** of the top
- **Stiffness** of the structure
- **Size** of the structure
- **Shape** of the structure
- Offset angle at the base of the robot

As mentioned in subsection 2.1.6 the reason for the long simulation time is the fact that the stiffness of the beam elements is very high compared to their weight, this is why, for all the tasks, we chose the ratio of the mass of a leg element and its stiffness constant. In order to verify this assumption a simulation was made where the mass of the leg elements was changed while other parameters were kept constant. As you can see in figure 3.1, the performance increases as the mass of the leg elements decreases, but abruptly decreases at a certain point, which is most likely due to numerical problems. Physically it also makes sense that a spring structure with a lower mass is more efficient then the opposite. So it can be concluded that this assumption is correct. Note that throughout the remainder of this section, the variable ϵ is the iteration parameter used to vary the physical parameters of the robot.

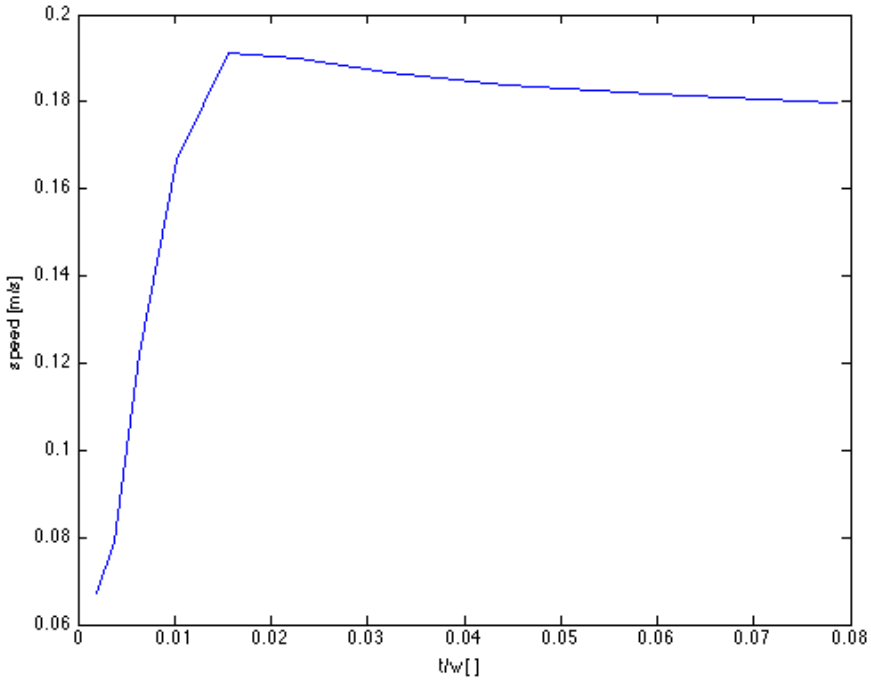

Figure 3.1: Influence of the stiffness to weight ratio of the leg elements on the performance of the system. Note: t = thickness, w = width

3.2.1 Rotational speed

The current optimal rotational speed of the top is related to the speed of the current the eigenfrequency of the beam structure. However rather then computing said eigenfrequency, we simply choose the frequency which maximizes the speed of the robot. While the optimal frequency for the robot is sometimes different from eigenfrequency of the beam structure, one might argue that; loosely speaking it is comparable to an eigenfrequency of the whole structure. According to this theory, using the optimized frequency would then mean that the optimal eigenmode of the whole system is excited. The then optimized rotational speed of the top is however optimal for the current configuration and might change as the configuration changes. In the following tasks, we try to stay close to this optimal eigenmode and eigenfrequency.

3.2.2 Stiffness

The optimization of the stiffness optimizes the force for a given deflection, this leads to a shift in the optimal rotational speed. This can be explained by the example of a simple spring mass system, as shown in figure 3.2, if one changes the stiffness of the spring then the eigenfrequency changes as well. Similarly for our system the optimal rotational speed changes as we change the stiffness of the structure. Below you can see the formulae that are used to vary the structure, using the iteration parameter ϵ.

$$w = w_0 \cdot \epsilon$$

$$t = t_0$$

$$\Rightarrow m_{\text{elem}} = m_{\text{elem }0} \cdot \epsilon$$

$$\Rightarrow k_{\text{elem}} = k_{\text{elem }0} \cdot \epsilon$$

$$\omega_{rot} = \omega_{rot} \cdot \sqrt{\epsilon}$$

Notice that as mentioned before the ratio k/m is constant. Also the only physical characteristic that is changed is the width of the beam.

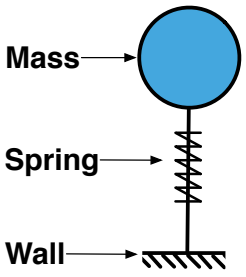

Figure 3.2: Spring mass oscillator

3.2.3 Size

The optimization of the size is somewhat unintuitive at first, rather then just changing the size of the structure, the thickness is changed as well. For an unchanged size the change in thickness would lead to $k_{\text{elem}} = k_{\text{elem }0} \cdot \epsilon^3$ and $m_{\text{elem}} = m_{\text{elem }0} \cdot \epsilon$, however since we also have a change of length by scaling the beam shape (for $l \backsim \epsilon$, $k \backsim 1/\epsilon$ because $k = \frac{E \cdot w_{elem} \cdot t_{elem}^3/12}{l_{elem}}$), which leads to the following formulae:

$$a = a_0 \cdot \epsilon$$
$$b = b_0 \cdot \epsilon$$
$$w = w_0$$
$$t = t_0 \cdot \epsilon$$
$$\Rightarrow m_{\text{elem}} = m_{\text{elem }0} \cdot \epsilon^2$$
$$\Rightarrow k_{\text{elem}} = k_{\text{elem }0} \cdot \epsilon^2$$
$$\omega_{rot} = \omega_{rot}$$

Where: a and b are the width and height of the ellipse defining the beam shape

One reason to change the size of the beam in this fashion, is that the force for small deflections stays identical and we only translate the point where we have 0 deflection, thus it does not vary the apparent stiffness of the structure a lot, which again means that we do not influence the previous task a lot. Another reason is that the ratio k/m is constant.

Using the simple model shown in figure 3.3 and the various formulae, it can be shown that the force does not change for small deflections Δh even though the size (in this case the length) is different. It is straightforward to see that this approximation extends to the case of the whole beam structure.

The derivation is as follows:

$$\text{Define: } \epsilon = l_1/l_2$$
$$k_i = \frac{E \cdot w_i \cdot t_i^3/12}{l_i}$$

The force for a given deflection can be approximated as follows:

$$F_i = k_i \cdot \Delta\alpha_i/l_i \simeq k_i \cdot \Delta\text{h}/l_i^2$$
$$F_1 = F_2 \iff k_1/l_1^2 = k_2/l_2^2$$
$$\iff \frac{E \cdot w_1 \cdot t_1^3/12}{l_1^3} = \frac{E \cdot w_2 \cdot t_2^3/12}{l_2^3}$$

Assume $w_1 = w_2$ and use ϵ:

$$\Longrightarrow t_1^3 = \epsilon^3 \cdot t_2^3 \Longrightarrow t_1 = \epsilon \cdot t_2$$

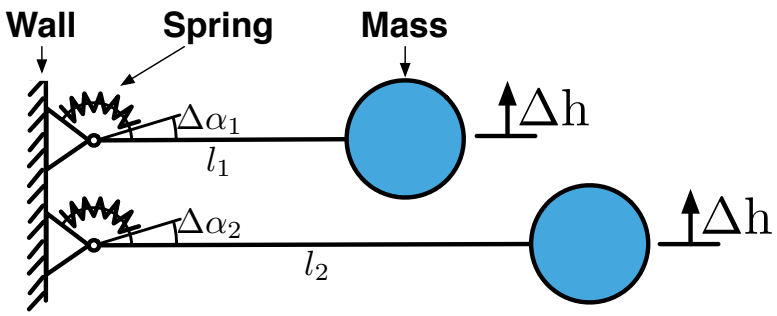

Figure 3.3: Derivation of the size optimization formulae

3.2.4 Shape

The optimization of the shape is similar to the size optimization. The big difference is that, because the change in length is a nonlinear function; we have to introduce a new variable ξ, which can then be used to calculate the new parameters. The new formulae are introduced below, for further explanations refer to the previous subsection:

$$a = a_0 \cdot \epsilon$$

$$b = b_0$$

$$w = w_0$$

$$\xi = \frac{l_{\text{elem}}(\epsilon)}{l_{\text{elem }0}}$$

$$t = t_0 \cdot \xi$$

$$\Rightarrow m_{\text{elem}} = m_{\text{elem }0} \cdot \xi^2$$

$$\Rightarrow k_{\text{elem}} = k_{\text{elem }0} \cdot \xi^2$$

$$\omega_{rot} = \omega_{rot}$$

Where: a and b are the width and height of the ellipse defining the beam shape

Because of the nonlinear effects while changing the shape, it is not possible to say that the force stays identical for small deflections, however one can argue that for small changes in size the forces stay locally similar. This is a statement which is a lot less strong, i.e. the range of values for which it is going to be a good approximation is a lot smaller. Thus it is preferable to repeat the other optimization tasks after optimizing the shape. Note that here the ratio k/m is also constant.

Chapter 4

Results

4.1 Simulation

Using the methods derived in chapter 3 the system was optimized.[1] The initial system was configured as follows:

Table 4.1: Initial parameters of optimization

Property	Name	Value	Unit
Depth of the beam shape	$a/2$	0.15	[m]
Height of the beam shape	b	0.3	[m]
Thickness	t	1.1 e-3	[m]
Width	w	30 e-3	[m]
Rotation speed	ω_{rot}	19.45	[rad/s]

Rotational speed First the rotational speed of the top was optimized. Figure 4.1 shows the performance for different frequencies. Note that most of the optimizations are run more then once (not shown) in order to achieve a more precise result for the optimized parameter, while having an idea of the general layout of the performance. This un-automated process amongst other avoids choosing outliers. Since the optimal rotation frequency is related to the eigenfrequency of the whole structure.

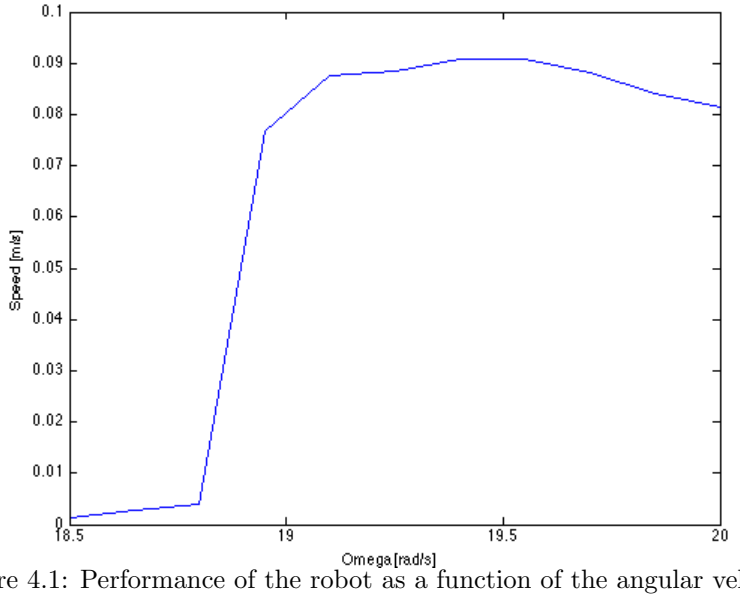

Figure 4.1: Performance of the robot as a function of the angular velocity

[1] Note: running the simulation on different computers leads to slightly different results

Stiffness Using the then calculated optimal rotational frequency, the stiffness is optimized, while changing the frequency itself as specified in subsection 3.2.2. This is achieved by changing the physical width of the beam. An interesting feature of the performance curve is that after a very fast increase it only decreases slowly for an increase in stiffness, i.e. in the real robot it might be interesting to choose a stiffness which is slightly higher then the optimized one. It is important however that if the stiffness is increased, the frequency has to be increased accordingly.

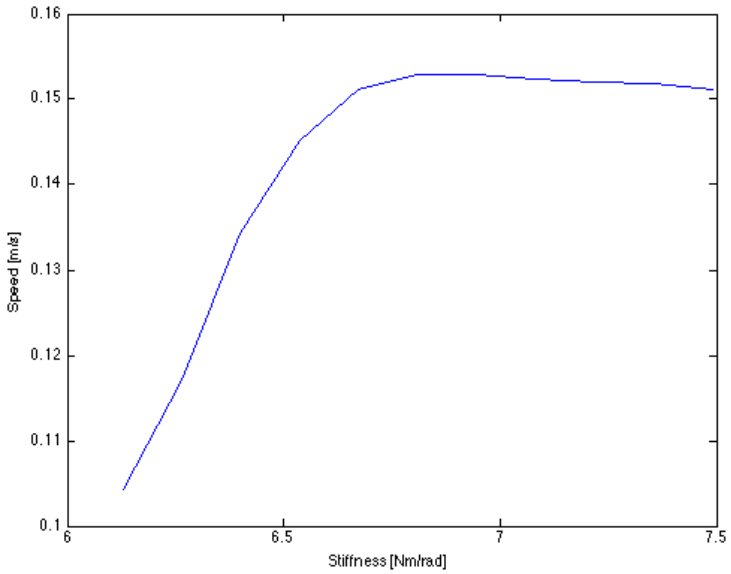

Figure 4.2: Performance of the robot as a function of the stiffness

Size In the next task, using the new optimized parameters, the size of the robot is optimized. Starting from a size of .15 m depth and .3 m height, we can see in figure 4.3 that the initial size is far from optimal. The frequency at which the motor is rotating is constant for this task, the basic idea is to shift the point of 0 deflection in the vertical direction without changing the rest of the behavior of the system too much.

Shape Finally we have the shape optimization, which changes the depth of the beam structure while keeping the height constant. Here the idea is that similarly to the size optimization, the shape is changed which means that the general layout of the forces changes. The optimization favored a more slender shape as can be seen in figure 4.4, it can also be seen that if the shape is too slender the system becomes unstable. A higher performance can be achieved by doing another iteration of the optimization tasks (frequency, stiffness, size).

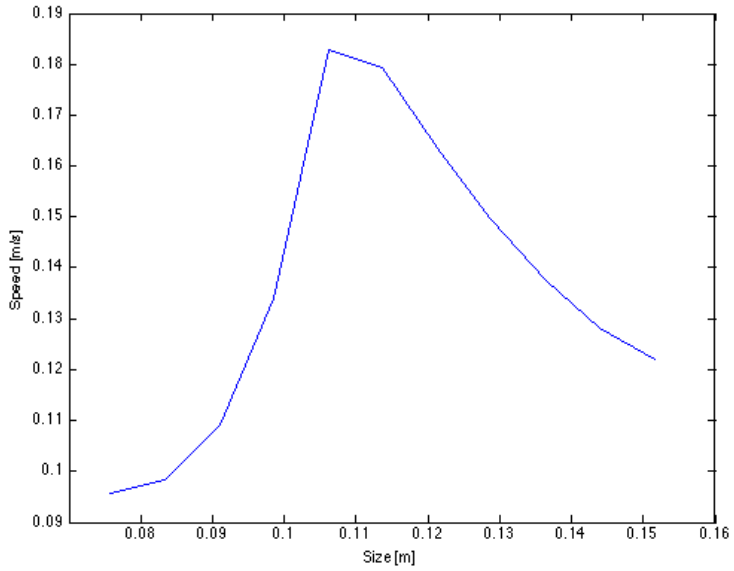

Figure 4.3: Performance of the robot as a function of the depth of the structure

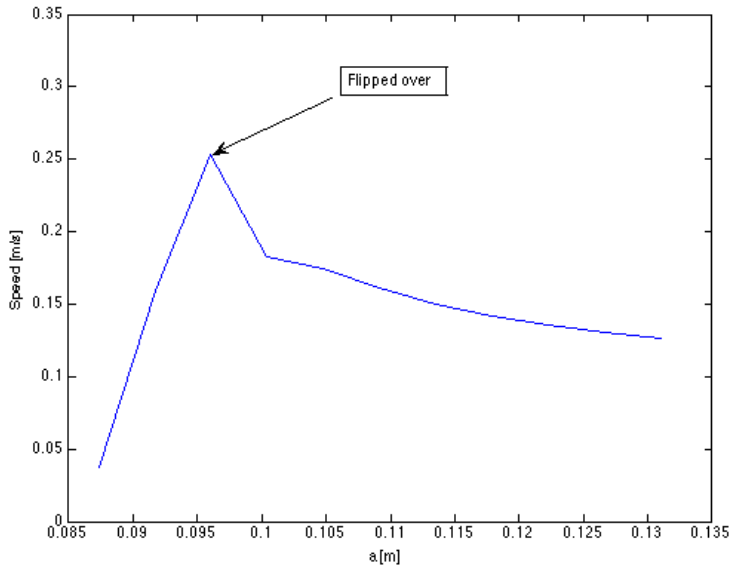

Figure 4.4: Performance of the robot as a function of the depth of the beam shape

The best performance achieved was 0.2009 m/s for the parameters given below, compared to the initial performance of 0.0936 m/s the performance more then doubled through just one optimization run after optimizing the shape. It also fares well compared to the previous work[2], especially when considering the simplicity of the new approach.

[2] [Näf, 2012]

Table 4.2: Final parameters of optimization

Property	Name	Value	Unit
Depth of the beam shape	$a/2$	0.0943	[m]
Height of the beam shape	b	0.2096	[m]
Thickness	t	7.6032 e-04	[m]
Width	w	44.88 e-3	[m]
Rotation speed	ω_{rot}	25.60	[rad/s]

4.2 Verification

Dominik Näf has shown that the model given in his thesis[3], gives qualitative results. There are only two significant differences between the simulation used in this project and the one used in the previous thesis.

The first one being the GC model. However except for it's increased stability it behaves virtually identical to the previous GC model; which is due to the fact that it is a different implementation of the same model.

The second difference is the estimation of the parameters of the discretization elements. In this case we have shown in appendix A, that even though this approximation might not be very precise, it should still be fairly easy to find the right beam dimensions because all the other parameters, as well as the dynamic response, are known.

We can conclude that if the actual beam stiffness is chosen to be identical to the one predicted, this simulation yields qualitative results. While experiments, in this case, may not be necessary to verify the results found in this project; they certainly would be beneficial. Unfortunately, due to time constraints as well as technical limitations, no experiments using a robot close to the one simulated could be performed.

4.3 Discussion

The results from section 4.1 have shown that the proposed approach works and gives an easy way to provide results in this multivariable search space. During the simulations it could also be observed that for the optimal solution the time to take one step t_{step} and the time of one rotation converge to the same value. This was also found in the thesis of Näf[4], using a completely different optimization method. This in turn might indicate that when the eigenfrequency of the beam structure and the eigenfrequency of the whole system, as introduced in subsection 3.2.1, converge the solution is at least optimal for some of the parameters. Although in this situation this might also come from the fact that the rotating mass has a lot of weight compared to the mass of the head.

On the other hand section 4.2 shows that the results we have found could easily be implemented in a real robot. Furthermore, the technique derived can be used for other similar structures.

[3] [Näf, 2012]
[4] [Näf, 2012]

Chapter 5

Conclusion

This work has presented a novel approach toward optimizing the parameters of the beam structure of a CBH or similar structures. How the different optimizations have been run has been explained in chapter 2 and their results have been discussed in chapter 3. The most important aspect of this technique is that the optimization tasks have to be chosen such that they only influence one characteristic of the whole system. The order in which these different optimization tasks are run has been chosen as follows: First the speed due to excitation of the rotational mass was maximized, i.e. the rotational frequency and the stiffness of the beam structure. Second the position of the 0-deflection point and the way the forces act was optimized, i.e. the details of the structure were optimized. The influence of the different sequences for these optimizations was not analyzed, however this could be done in the future.

Furthermore the GC model of the previous simulation has been revised and the new model performs consistently better in matters of stability.

Finally a massless approximation of the beam structure has been introduced in appendix B. While it might not be precise enough to approximate the whole beam structure because it does not consider the mass, it could be used to approximate sections of the beam structure and thus deliver a more precise approximation of the stiffness of the discretization elements. Another option might be to rely on some kind of tabulated data or a completely new approximation of the beam structure.

There are various directions in which further investigations could go. One possibility would be to use the current simulation to research new shapes, such as more asymmetrical shapes; likes ellipses, which are not cut along the major or the minor axis, but rather along an arbitrary one. It could also be interesting to investigate how a change of the foot or the head would change the optimal solution of the beam structure. A more precise way to approximate the stiffness of the discretization elements would certainly help to increase the accuracy of the simulation. It might also be helpful to include a model which checks for plastic deformations in the beam structure.

Finally a very interesting subject would be to analyze the possibility of using different eigenmodes of the system in order to achieve different tasks, like hopping higher for some of the time and then faster when it is not necessary to hop high anymore, etc..

Bibliography

[Alexander, 2003] Alexander, R. (2003). *Principles of animal locomotion*. Princeton Univ Pr.

[Chudnovsk et al., 2006] Chudnovsk, V. et al. (2006). Modeling flexible bodies in simmechanics and simulink. *MatLab Digest, May*.

[Lataniotis, 2013] Lataniotis, C. (2013). Cpg-inspired control of curved beam hoppers. Master's thesis.

[McGeer, 1990] McGeer, T. (1990). Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62.

[Näf, 2012] Näf, D. (2012). Shape optimization of a curved beam hopping robot. Master's thesis, BIRL, ETH Zurich.

[Pfeifer and Bongard, 2006] Pfeifer, R. and Bongard, J. C. (2006). *How the body shapes the way we think: a new view of intelligence*. MIT press.

[Raibert, 1985] Raibert, M. (1985). Legged robots that balance.

[Raibert et al., 2008] Raibert, M., Blankespoor, K., Nelson, G., Playter, R., et al. (2008). Bigdog, the rough–terrain quadruped robot. In *Proceedings of the 17th World Congress*, pages 10823–10825.

[Reis and Iida, 2012] Reis, M. and Iida, F. (2012). An energy-efficient hopping robot based on free vibration of a curved beam.

[Reis et al., 2013] Reis, M., Yu, X., Maheshwari, N., and Iida, F. (2013). Morphological computation of multi-gaited robot locomotion based on free vibration. *Artificial Life*, 19(1):97–114.

Appendix A

Euler Bernouilli Beam Approximation

As noted previously, the Euler Bernouilli beam theory gives us the approximation of the stiffness of a beam element as noted below. This formula is only an approximation and it has been shown to be only qualitatively correct[1].

$$k_{\text{elem}} = E \cdot \frac{w \cdot t^3/12}{l_{\text{elem}}}$$

In the master thesis, preceding this work, the stiffness of the beam element has been estimated[2], this estimate has been used as the reference value in table A.1. Both estimates are for a beam width of 10 mm, a beam thickness of 1.25 mm and an element which is 38.4 mm long.

Table A.1: Comparison stiffness: Approx. vs. Reference

Euler Bernouilli	Reference	Unit
8.5	9.5	[Nm/rad]

There is a considerable discrepancy between the estimate, which is based on the Euler Bernouilli beam theory, and the reference value, which was extrapolated using a motion capture system. Since the reference value is optimized to fit the dynamic response of the physical system, it is assumed to be the physical truth. This in turn means that for the true physical system we do not know the optimal dimensions of the beam, however we know all the other parameters such as shape, size and the optimal rotational frequency, as well as the dynamic characteristics of the systems. This in turn means that it should be possible to find the right beam parameters starting from the optimized parameters.

[1] [Chudnovsk et al., 2006]
[2] [Näf, 2012]

Appendix B

Massless Approximation

In order to reduce the computational load, the idea of a massless approximation of the whole structure was followed. The process of the approximation is fairly simple. The beam is completely fixed on one side like it would be in the case of the robot and the position of the other end is fixed as well, it can however freely rotate on that end as shown in figure B.1. The beam is then let go, starting from an initial state, which is not statically stable and then converges to the static deflection of the beam. Finally the reaction forces are measured in the freely rotating joint. The approximation is completely massless since it is run in a zero-gravity environment.

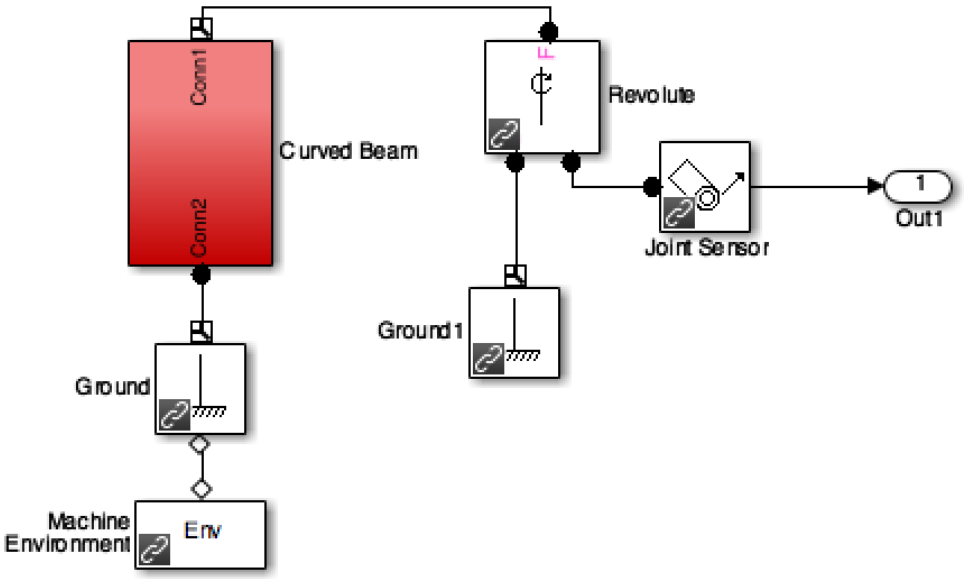

Figure B.1: Top view of the SimMechanics model

As shown in figure B.2 there are however significant differences between this approximation and the simulation, described in chapter 2, it approximates. However

it might be possible to use this simulation to make an approximation with less elements which is not completely massless. It could also be used to calculate better approximations of the stiffness of the different elements of the discretization.

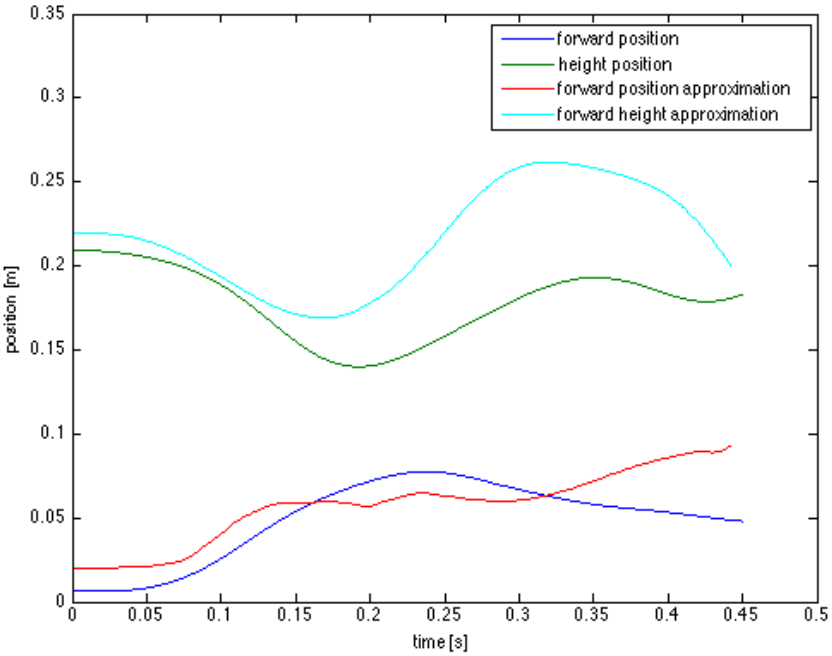

Figure B.2: Comparison between the massless approximation and the original simulation for identical parameters

Note that the Matlab files for this part of the semester project can only be found on the CD.

Appendix C

SimMechanics Model

C.1 Overview

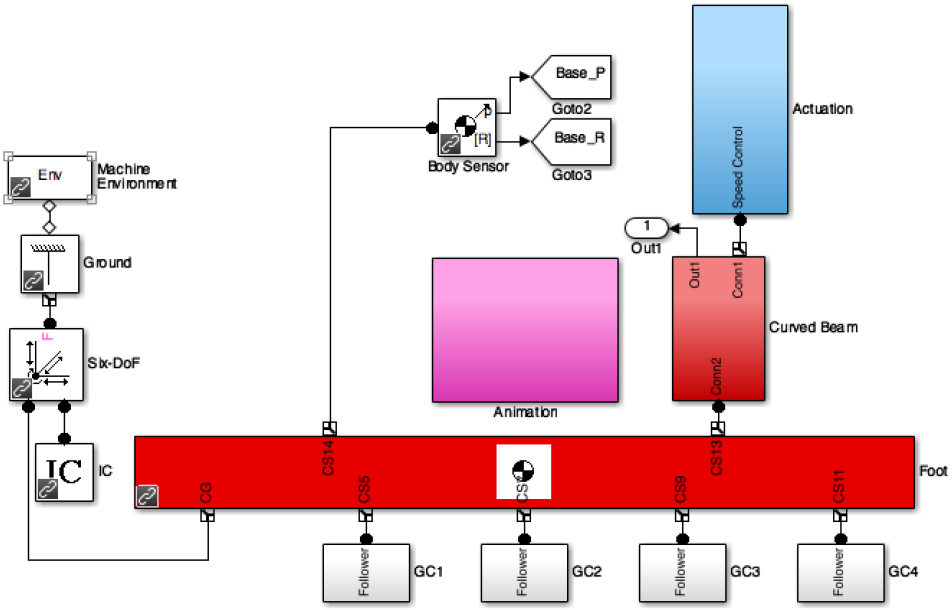

Figure C.1: Top view of the SimMechanics model

C.2 GC Model (GC1-GC4)

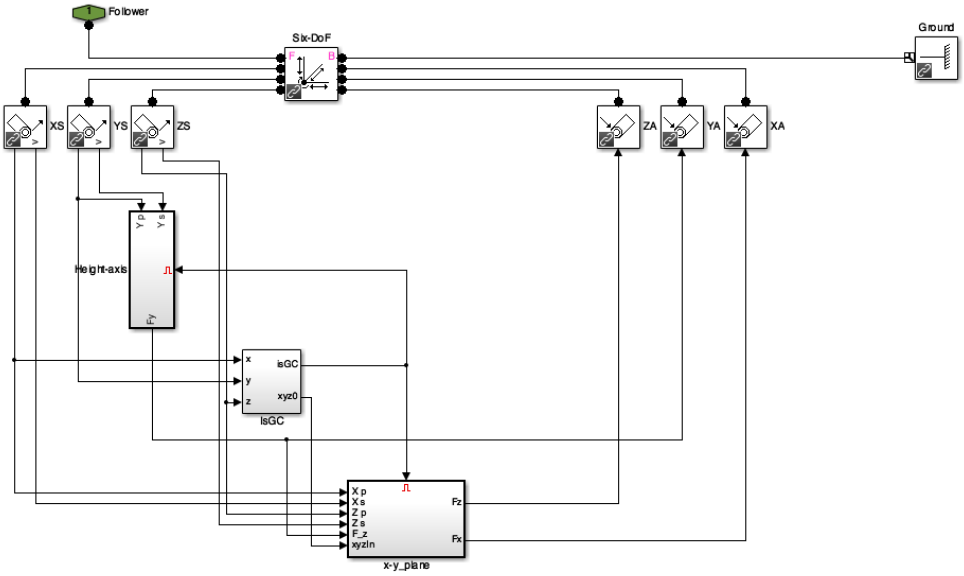

Figure C.2: Top view of the GC model

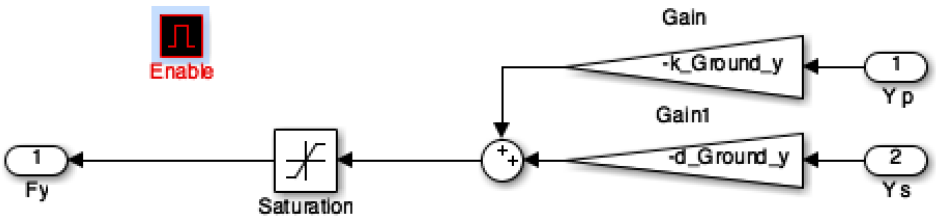

Figure C.3: Height-axis block in the GC model

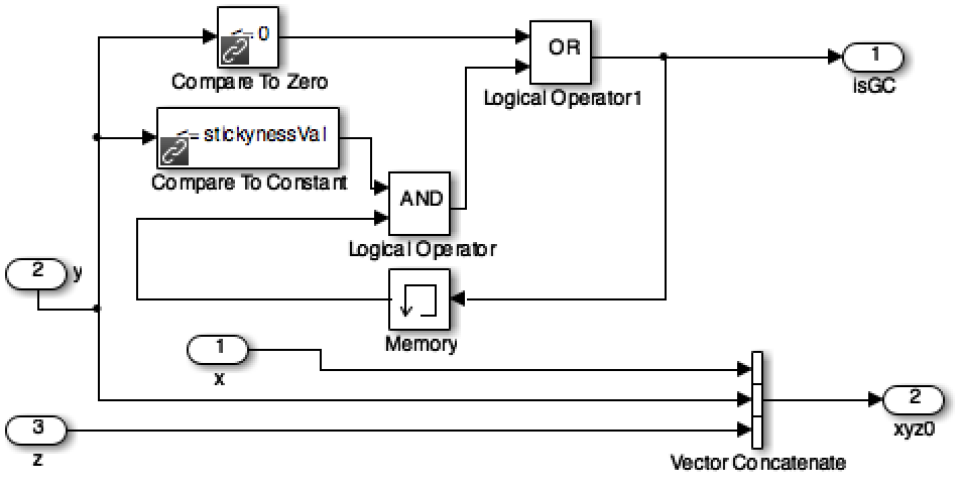

Figure C.4: isGC block in the GC model

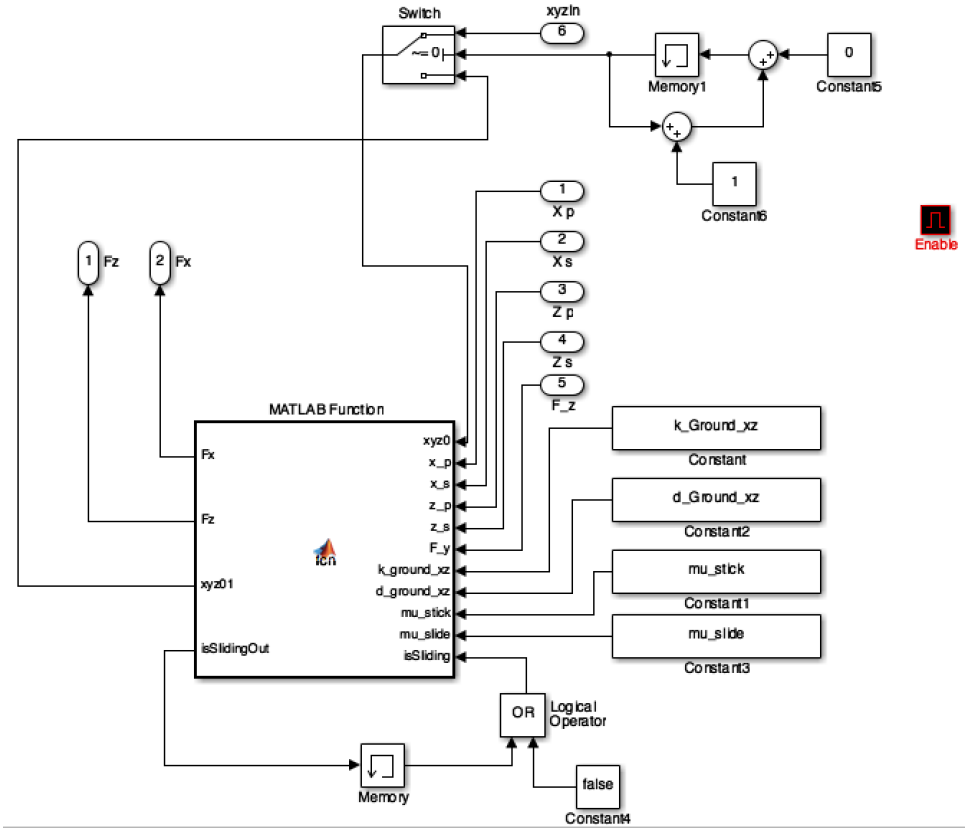

Figure C.5: x-y plane block in the GC model

MATLAB Function in the x-y plane block

```
function [Fx,Fz,xyz01,isSlidingOut] = ...
    fcn(xyz0,x_p,x_s,z_p,z_s,F_y,k_ground_xz,d_ground_xz,mu_stick,...
        mu_slide,isSliding)
    isSlidingOut = isSliding;
    xyz01 = xyz0;
    x0 = xyz0(1);
    y0 = xyz0(2);
    z0 = xyz0(3);

    dx = x_p-x0;
    dz = z_p-z0;

    dangle = atan2(dz,dx);
    sangle = atan2(z_s,x_s);
    dmag = sqrt(dx^2+dz^2);
    smag = sqrt(x_s^2+z_s^2);

    if isSliding
        Fmag = mu_slide*F_y;
        if Fmag > d_ground_xz*smag
            isSlidingOut = false;
            Fmag = d_ground_xz*smag;
            xyz01 = [x_p;y0;z_p];
        end
        Fx = -Fmag*cos(sangle);
        Fz = -Fmag*sin(sangle);
    else
        Fx = -dmag*k_ground_xz*cos(dangle) - ...
            smag*d_ground_xz*cos(sangle);
        Fz = -dmag*k_ground_xz*sin(dangle) - ...
            smag*d_ground_xz*sin(sangle);
        Fmag = sqrt(Fx^2 + Fz^2);
        if Fmag > mu_stick*F_y
            isSlidingOut = true;
            Fmag = abs(mu_slide*F_y);
            Fx = -Fmag*cos(sangle);
            Fz = -Fmag*sin(sangle);
        end

    end
end
```

C.3 Curved Beam

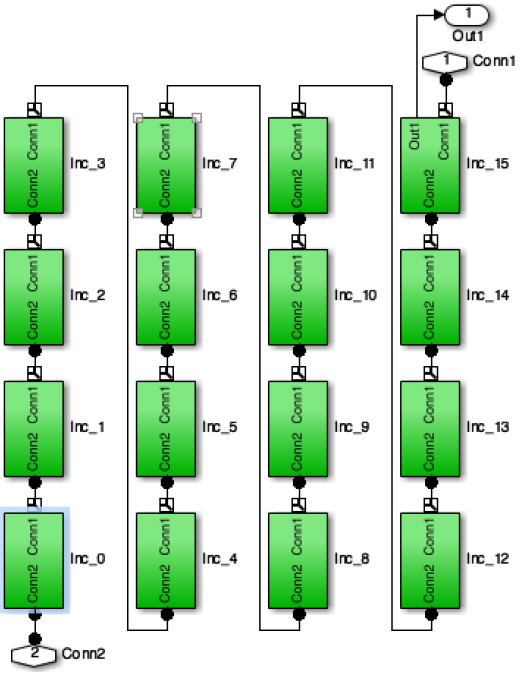

Figure C.6: Top view of the Curved Beam model

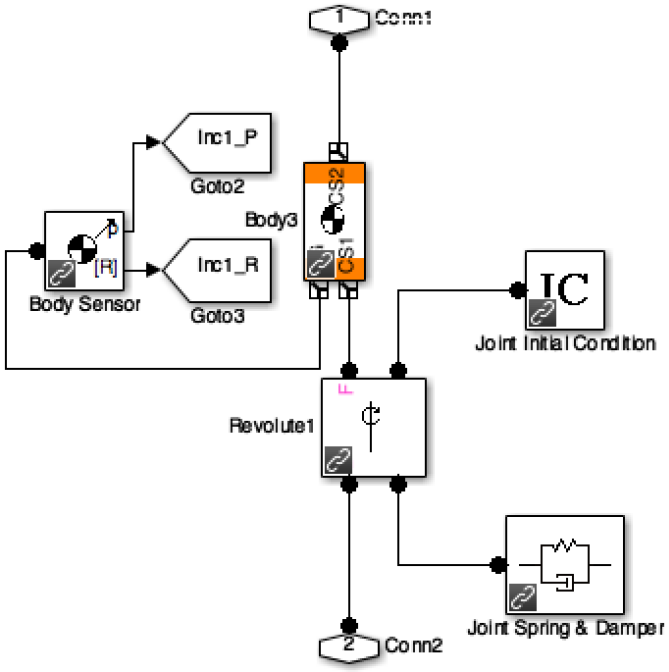

Figure C.7: Inc_i in the Curved Beam model

C.4 Actuation

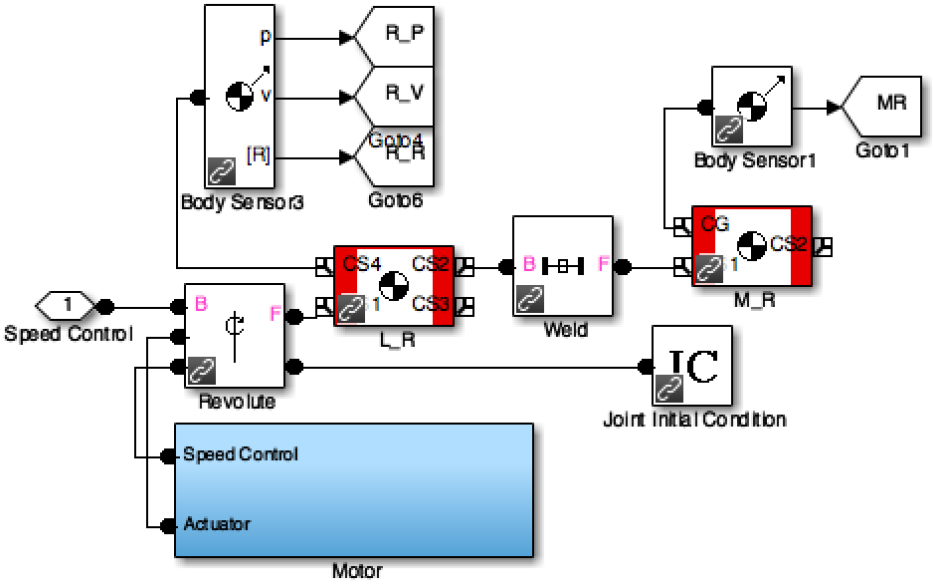

Figure C.8: Top view of the Actuation model

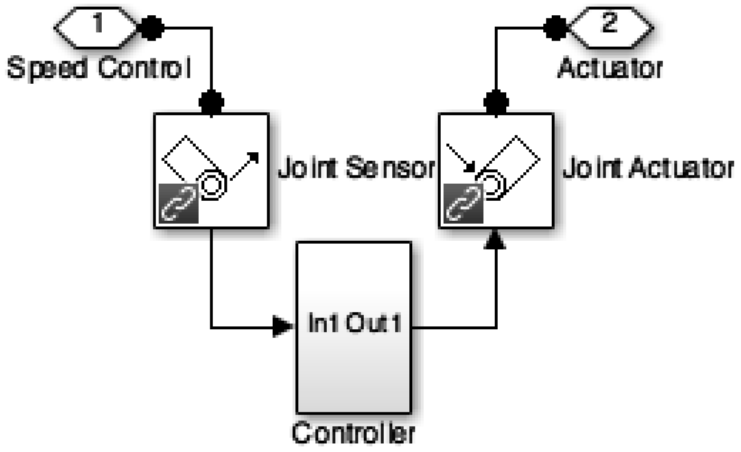

Figure C.9: Motor in the Actuation model

Appendix D

Matlab Framework

Note: Notation/variables are sometimes different from the ones presented in the text.

D.1 Curve Generation

D.1.1 curve.m

```
function [L_Leg_n, Alpha] = curve(params,N)
    % Convert circle to equivalent ellipse
    if strcmp(params.shape,'circle')
        params.shapeparams.a = params.shapeparams.R;
        params.shapeparams.b = params.shapeparams.R;
        params.shape = 'ellipse';
    end
    if strcmp(params.shape,'ellipse')
        if mod(N,2)
            disp('Discretize using pair amount of elements')
            return
        end
        t = pi/2:.0004:pi+pi/2;
        x = params.shapeparams.a*cos(-t);
        y = params.shapeparams.b*sin(-t);

        xx = repmat(x,length(x),1);
        yy = repmat(y,length(y),1);
        dx = xx-xx';
        dy = yy-yy';
        l = sqrt(dx.^2+dy.^2);
        for j = 1:length(x)
            l(j,1:j) = inf*ones(1,j);
        end
        way = zeros(N,length(x));
        costw = zeros(N,length(x));
        way(1,:) = 1:length(x);
        costw(1,:) = l(1,:);
        relevantways = length(x)-1;
        for i=2:N-1
            [way(i,:),costw(i,:),relevantways] = ...
                findway(way(i-1,:),costw(1,:),l,relevantways);
        end
        way(N,:) = repmat(length(x),1,length(x));
        for k=1:length(x)
            costw(N,k) = l(way(N-1,k),way(N,k));
        end
        [~, optwayidx] = min(var(costw));
        fprintf('Variance: %d \n',var(costw(:,optwayidx)));
        optway = way(:,optwayidx);
        xy = [x(1) x(optway);y(1) y(optway)];
```

```
41          plot(x,y,'b')
42          axis equal
43          hold on
44          plot(xy(1,:),xy(2,:),'rx')
45          plot(xy(1,:),xy(2,:),'g')
46          L_Leg_n = mean(costw(:,optwayidx));
47          vecs =[];
48          for l=2:length(xy(1,:))
49              vecs = [vecs,xy(:,l)-xy(:,l-1)];
50          end
51          vecs = [[-L_Leg_n; 0],vecs,[L_Leg_n; 0]];
52          Alpha = [];
53          for m=2:length(vecs(1,:))
54              temp = getVecAngle(vecs(:,m-1),vecs(:,m));
55              Alpha = [Alpha, temp];
56          end
57          Alpha = Alpha(1:16)';
58      end
59      if strcmp(params.shape,'triangle')
60          if mod(N,2)
61              disp('Discretize using pair amount of elements')
62              return
63          end
64
65          Alpha  = zeros(N,1);
66          angle  = atan2(params.y_trim,params.x_trim);
67          len = sqrt(params.x_trim^2+params.y_trim^2);
68
69          Alpha(1)   = acos(len/(params.L_Leg_n*N));
70          Alpha(N/2+1) = -2*Alpha(1);
71
72          Alpha(1) = Alpha(1)+angle-pi;
73          L_Leg_n  = params.L_Leg_n;
74      end
75      if strcmp(params.shape,'ellipse_trim')
76          if mod(N,2)
77              disp('Discretize using pair amount of elements')
78              return
79          end
80
81          Alpha  = zeros(N,1);
82          len = sqrt(params.x_trim^2+params.y_trim^2);
83          Alpha_med = linspace(0,2*pi/N,100000)';
84          vec_res = zeros(100000,2);
85
86          for k = 1:N
87              vec_res = vec_res + params.L_Leg_n*[cos(-k*Alpha_med), ...
                    sin(-k*Alpha_med)];
88          end
89          len_res   = sqrt(vec_res(:,1).^2+vec_res(:,2).^2);
90          [~, idx]  = min(abs(len_res-len));
91          Alpha(2:16) = -repmat(Alpha_med(idx),15,1);
92          final_pos = zeros(100000,2);
93          d_angle = linspace(0,2*pi,100000)';
94
95          for k = 1:N
96              final_pos = final_pos + ...
                    params.L_Leg_n*[cos(d_angle+sum(Alpha(1:k))), ...
                    sin(d_angle+sum(Alpha(1:k)))];
97          end
98          [~,idx2] = ...
                min((final_pos(:,1)-params.x_trim).^2+(final_pos(:,2)-params.y_trim).^2);
99          Alpha(1) = Alpha(1)+d_angle(idx2)-pi;
100
101         L_Leg_n  = params.L_Leg_n;
102     end
103 end
```

D.1.2 findway.m

```
function [way,costw,relevantways] = ...
    findway(oldway,initcost,l,relevantways)
    way = 0*initcost;
    costw = 0*initcost;
    for i=1:relevantways
 %      abs(l(oldway(i),:)-initcost(i))
        [~,way(i)] = min(abs(l(oldway(i),:)-initcost(i)));
        costw(i) = l(oldway(i),way(i));
%        way(i)
        if way(i)==length(way)
            relevantways = i-1;
            way(i:end)=0;
            costw(i:end)=inf;
            break;
        end
    end
    way(relevantways+1:end)=1;
    costw(relevantways+1:end)=inf;
end
```

D.1.3 getVecAngle.m

```
function angle = getVecAngle(v1,v2)
    v1 = [v1;0];
    v2 = [v2;0];
    angle = 2*sign(cross(v1,v2))'*[0 0 ...
        1]'*atan(norm(v1-v2)/norm(v1+v2));
%    norm(v1-v2)
end
```

D.2 Optimization

D.2.1 Frequency.m

```
params.iterparam = linspace(18.5,20,params.nsims);%*omega;
params.iterparam_name = 'omega';
success = zeros(length(params.iterparam),1);
performance = zeros(length(params.iterparam),1);

tfinal = params.simTime;

tic
for i = 1:length(params.iterparam)
    fprintf(['Simulation %g out of %g, Value of optimization ...
        parameter ' params.iterparam_name ': ...
        %g\n'],i,length(params.iterparam),params.iterparam(i))
    omega = params.iterparam(i);
    try
        [t1,~,y1] = sim(simName,tfinal,[]);
        success(i)=~any(y1(:,2)<0);
        idx2 = find(t1>2,1);
        if success(i)
            performance(i) = y1(end,1)-y1(idx2,1);
        else
            idx = find(y1(:,2)<0,1,'first');
            performance(i) = y1(idx,1)-y1(idx2,1);
```

```
        end
        fprintf('Simulation success, performance: %g\n',performance(i))
    catch
        success(i)=-1;
        successi = success(i);
        performance(i) = NaN;
        disp('failure')
    end
    toc
end

saveallinf
```

D.2.2 Stiffness.m

```
%% Change stiffness while keeping eigenmode

params.iterparamspace = linspace(1.1,1.5,params.nsims);
params.iterparam_name = 'space;k_Leg_n;m_Leg_n;omega';
params.iterparam = [params.iterparamspace;...
                    params.iterparamspace.^1*k_Leg_n;...
                    params.iterparamspace.^1*m_Leg_n;...
                    params.iterparamspace.^.5*omega];

success = zeros(length(params.iterparamspace),1);
performance = zeros(length(params.iterparamspace),1);

tfinal = params.simTime;

tic
for i = 1:length(params.iterparamspace)
    fprintf(['Simulation %g out of %g, Value of optimization ...
        parameter ' params.iterparam_name ': ...
        %g\n'],i,length(params.iterparamspace),params.iterparam(2,i))
    k_Leg_n = params.iterparam(2,i);
    m_Leg_n = params.iterparam(3,i);
    omega   = params.iterparam(4,i);
    try
        [t1,~,y1] = sim(simName,tfinal,[]);
        success(i)=~any(y1(:,2)<0);
        idx2 = find(t1>2,1);
        if success(i)
            performance(i) = y1(end,1)-y1(idx2,1);
        else
            idx = find(y1(:,2)<0,1,'first');
            performance(i) = y1(idx,1)-y1(idx2,1);
        end
        fprintf('Simulation success%g, performance: ...
            %g\n',success(i),performance(i))
    catch
        success(i)=-1;
        performance(i) = NaN;
        disp('failure')
    end
    toc
end

saveallinf
```

D.2.3 SizeStiffFreqNew.m

```
%% Keep Omega const and change size
params.iterparamspace = linspace(.8,1.2,params.nsims);%.690
params.iterparam_name = 'space;sizeA;sizeB;params.b;params.h;omega';
params.iterparam = [params.iterparamspace;...
                    params.iterparamspace*params.shapeparams.a;...
                    params.iterparamspace*params.shapeparams.b;...
                    params.iterparamspace.^0*params.b;...
                    params.iterparamspace.^1*params.h;...
                    params.iterparamspace.^0*omega];

success = zeros(length(params.iterparamspace),1);
performance = zeros(length(params.iterparamspace),1);

tfinal = params.simTime;

tic
for i = 1:length(params.iterparamspace)
    fprintf(['Simulation %g out of %g, Value of optimization ...
        parameter ' params.iterparam_name ': ...
        %g\n'],i,length(params.iterparamspace),params.iterparam(1,i))
    params.shapeparams.a = params.iterparam(2,i);
    params.shapeparams.b = params.iterparam(3,i);
    params.b = params.iterparam(4,i);
    params.h = params.iterparam(5,i);
    params.I = params.b*params.h^3/12;
    omega    = params.iterparam(6,i);
    recalc_curve

    try
        [t1,~,y1] = sim(simName,tfinal,[]);
        success(i)=~any(y1(:,2)<0);
        idx2 = find(t1>2,1);
        if success(i)
            performance(i) = y1(end,1)-y1(idx2,1);
        else
            idx = find(y1(:,2)<0,1,'first');
            performance(i) = y1(idx,1)-y1(idx2,1);
        end
        fprintf('Simulation success %g, performance: ...
            %g\n',success(i),performance(i))
    catch
        success(i)=-1;
        performance(i) = NaN;
        disp('failure')
    end
    toc
end

saveallinf
```

D.2.4 LegWeight.m

```
params.iterparamspace = linspace(.9,1.1,params.nsims);%.690
params.iterparam_name = 'space;params.b;params.h';
params.iterparam = [params.iterparamspace;...
                    params.iterparamspace.^-3*params.b;...
                    params.iterparamspace.^1*params.h];


success = zeros(length(params.iterparam),1);
```

```
performance = zeros(length(params.iterparam),1);

tfinal = params.simTime;

tic
for i = 1:length(params.iterparamspace)
    fprintf(['Simulation %g out of %g, Value of optimization ...
        parameter ' params.iterparam_name ': ...
        %g\n'],i,length(params.iterparamspace),params.iterparam(1,i))
    params.b = params.iterparam(2,i);
    params.h = params.iterparam(3,i);
    params.I = params.b*params.h^3/12;

    m_Leg_n = params.density*params.b*params.h*L_Leg_n;
    I_Leg_n = 1/12 * m_Leg_n * [0.000001 0 0;0 L_Leg_n^2 0;0 0 ...
        L_Leg_n^2];

    k_Leg_n = stiffness_k(L_Leg_n,params);

    try
        [t1,¬,y1] = sim(simName,tfinal,[]);
        success(i)=¬any(y1(:,2)<0);
        idx2 = find(t1>2,1);
        if success(i)
            performance(i) = y1(end,1)-y1(idx2,1);
        else
            idx = find(y1(:,2)<0,1,'first');
            performance(i) = y1(idx,1)-y1(idx2,1);
        end
        fprintf('Simulation success %g, performance: ...
            %g\n',success(i),performance(i))
    catch
        success(i)=-1;
        performance(i) = NaN;
        disp('failure')
    end
    toc
end

saveallinf
```

D.2.5 OffsAngle.m

```
params.iterparam = linspace(0,pi/32,params.nsims);
params.iterparam_name = 'Alpha_offs';
success = zeros(length(params.iterparam),1);
performance = zeros(length(params.iterparam),1);

tfinal = params.simTime;

tic
for i = 1:length(params.iterparam)
    fprintf(['Simulation %g out of %g, Value of optimization ...
        parameter ' params.iterparam_name ': ...
        %g\n'],i,length(params.iterparam),params.iterparam(i))
    Alpha_offs    = params.iterparam(i);

    try
        [t1,¬,y1] = sim(simName,tfinal,[]);
        success(i)=¬any(y1(:,2)<0);
        idx2 = find(t1>2,1);
        if success(i)
            performance(i) = y1(end,1)-y1(idx2,1);
        else
```

```
20              idx = find(y1(:,2)<0,1,'first');
21              performance(i) = y1(idx,1)-y1(idx2,1);
22          end
23          fprintf('Simulation success %g, performance: ...
                %g\n',success(i),performance(i))
24      catch
25          success(i)=-1;
26          performance(i) = NaN;
27          disp('failure')
28      end
29      toc
30  end
31
32  saveallinf
```

D.3 Miscellaneous

D.3.1 stiffness_k.m

```
1  function k = stiffness_k(length,params)
2      k = params.E_modulus*params.I./length;
3  end
```

D.3.2 curve.m

```
1  [L_Leg_n, Alpha] = curve(params,N_Leg_n);
2
3  Alpha_0 = Alpha;
4  Alpha_init = Alpha;
5
6  % Spring constant and damping rate between leg incrementals
7  k_Leg_n = stiffness_k(L_Leg_n,params);
8
9  % Mass and inertia of leg incremental
10 m_Leg_n = params.density*params.b*params.h*L_Leg_n;
11 I_Leg_n = 1/12 * m_Leg_n * [0.000001 0 0;0 L_Leg_n^2 0;0 0 L_Leg_n^2];
12
13 % The mass of point on the top
14 I_T = 1/12 * M_T * [0.000001 0 0;0 L_Leg_n^2 0;0 0 L_Leg_n^2];
```

D.3.3 saveallinf.m

```
1  performance = performance/(params.simTime-2);
2
3  pause(1)
4  load('allInformation.mat','allinf');
5  outpt = [params.iterparam',success,performance];
6  saveidx = length(allinf)+1;
7  allinf(saveidx).outpt   = outpt;
8  allinf(saveidx).omega   = omega;
9  allinf(saveidx).M_B     = M_B;
10 allinf(saveidx).L_B     = L_B;
11 allinf(saveidx).B_B     = B_B;
12 allinf(saveidx).I_B     = I_B;
13 allinf(saveidx).params  = params;
14 allinf(saveidx).N_Leg_n = N_Leg_n;
```

```
15 allinf(saveidx).L_Leg_n = L_Leg_n;
16 allinf(saveidx).Alpha   = Alpha;
17 allinf(saveidx).material= material;
18 allinf(saveidx).k_Leg_n = k_Leg_n;
19 allinf(saveidx).d_Leg_n = d_Leg_n;
20 allinf(saveidx).m_Leg_n = m_Leg_n;
21 allinf(saveidx).I_Leg_n = I_Leg_n;
22 allinf(saveidx).M_T     = M_T;
23 allinf(saveidx).I_T     = I_T;
24 allinf(saveidx).M_R_arm = M_R_arm;
25 allinf(saveidx).M_R     = M_R;
26 allinf(saveidx).L_R     = L_R;
27 allinf(saveidx).mu_slide= mu_slide;
28 allinf(saveidx).mu_stick= mu_stick;
29 allinf(saveidx).v_th    = v_th;
30 allinf(saveidx).k_Ground_y    = k_Ground_y;
31 allinf(saveidx).d_Ground_y    = d_Ground_y;
32 allinf(saveidx).k_Ground_xz   = k_Ground_xz;
33 allinf(saveidx).d_Ground_xz   = d_Ground_xz;
34 allinf(saveidx).stickynessVal = stickynessVal;
35 allinf(saveidx).g             = g;
36 allinf(saveidx).Task          = Task;
37 allinf(saveidx).L_Orig        = L_Orig;
38 allinf(saveidx).Alpha_offs    = Alpha_offs;
39 save('allInformation.mat', 'allinf');
```

D.4 Execution Files

Note that only the most general file to call the simulations is shown.

D.4.1 run_shape.m

```
1  clear all
2  clc
3  close all
4  warning('off','all')
5  addpath('GenerateCurve')
6  addpath('Sims')
7  addpath('Optimisation')
8  addpath('Misc')
9
10 %% Optimisation parameters
11 % Task
12 params.nsims = 11;
13 params.simTime = 10;
14
15 params.chooseshape = 0;
16 chooseshape = params.chooseshape;
17 Task = 'Shape';
18
19 param_choose_shape
20
21 %% Start
22 if strcmp(Task,'LegWeight')
23     LegWeight
24 elseif strcmp(Task,'Frequency')
25     Frequency
26 elseif strcmp(Task,'Stiffness')
27     Stiffness
28 elseif strcmp(Task,'SizeStiffFreq')
29     SizeStiffFreqNew
30 elseif strcmp(Task,'OffsAngle')
```

```
31      OffsAngle
32  elseif strcmp(Task,'Shape')
33      ShapeOpt
34  end
```

D.5 Parameter Files

Note that only the simplest parameter file is shown, while the optimal solution is in param_choose_shape.m

D.5.1 current_param.m

```
1   clear all
2   clc
3   close all
4   warning('off','all')
5   addpath('GenerateCurve')
6   addpath('Sims')
7   addpath('Optimisation')
8   addpath('Misc')
9
10  simName = 'Current_noanim';
11
12
13  % The rotational speed:
14  Ctrl_Gain = 1e-1;
15  omega = 19.45*sqrt(1.7);
16
17  %% ------ Design parameter of robot---------------------------
18  % The mass, length, width and Inertia of the link below (foot)
19  M_B = 0.058+0.015;
20  L_B = 0.275;
21  B_B = 0.248;
22  I_B = 1/12 * M_B * [B_B^2 0 0;0 L_B^2 0;0 0 (B_B^2 + L_B^2)];
23
24  L_Orig = 0;
25
26  %% Parameters Beam
27  % Beam Shape
28  params.shape = 'ellipse';
29  params.shapeparams.a = .3032/2*.72;
30  params.shapeparams.b = .3032/2*.72;
31  N_Leg_n =16;
32
33  % Chosen material
34  material= 'aluminium';
35  % Material properties
36  if strcmp(material,'steel')
37      % Steel
38      params.density = 7.8*10^3;
39      params.E_modulus = 200e9;
40  elseif strcmp(material,'aluminium')
41      % Aluminum
42      params.density = 2.7*10^3;
43      params.E_modulus = 69e9;
44  end
45
46  % Dimensions beam
47  params.b = 30*10^-3*1.7;
48  params.h = 1.1*10^-3*.72;
49  params.I = params.b*params.h^3/12;
50
```

```
51 [L_Leg_n, Alpha] = curve(params,N_Leg_n);
52 Alpha_offs = .04;
53
54 Alpha_0 = Alpha;
55 Alpha_init = Alpha;
56
57 % Spring constant and damping rate between leg incrementals
58 k_Leg_n = stiffness_k(L_Leg_n,params);
59 d_Leg_n = 0.003;
60
61 % Mass and inertia of leg incremental
62 m_Leg_n = params.density*params.b*params.h*L_Leg_n;
63 I_Leg_n = 1/12 * m_Leg_n * [0 0 0;0 L_Leg_n^2 0;0 0 L_Leg_n^2];
64
65 % The mass of point on the top
66 M_T = 0.088;
67 I_T = 1/12 * M_T * [0 0 0;0 L_Leg_n^2 0;0 0 L_Leg_n^2];
68
69 % The mass and length of the rotating arm
70 M_R_arm = 0.008;
71 M_R = 0.0334;
72 L_R = 0.035;
73
74 %% ------ Ground Contact Model: ...
       ----------------------------------------------
75 % Friction constants and velocity threshold
76 mu_slide = 0.4;
77 mu_stick = 0.5;
78 v_th = 1e-2;
79
80 % ground reaction parameters (y-Axis)
81 k_Ground_y = 140;
82 d_Ground_y = 15;
83 k_Ground_xz = 20;
84 d_Ground_xz = 8;
85 stickynessVal = 1e-4;
86
87 % Gravity
88 g = 9.81;
```