Gravity compensation for a bipedal humanoid robot

BSc. H.C. Bouwmans

DC 2012.017

Coach(es): ir. P.W.M. van Zutven

Supervisor: prof. dr. H. Nijmeijer

Technische Universiteit Eindhoven
Department Mechanical Engineering
Dynamics and Control Technology Group

Eindhoven, April, 2012

Abstract

Many research is nowadays done on humanoid robots. They can be very helpful in the future, for example in nursery homes to assist the nurses. The big advantages of humanoid robots is that they can operate in the human environment. At the Eindhoven University of Technology the robot TUlip is used as an experimental platform. TUlip is a bipedal humanoid robot with six actuators in each leg. When the robot is in double support phase the robot loses some degrees of freedom and becomes over actuated. The goal of this project is to design a feedforward gravity compensation algorithm which facilitates the feedback controllers of the robot joints and deals with the problem of over actuation.

To obtain a feedforward algorithm which facilitates the feedback controllers of TUlip, calculations with the manipulator Jacobian are taken as a basis in designing the algorithm. When a certain force is desired on the tip of a robot manipulator the manipulator Jacobian can be used to calculate the required joint torques. In the case of the robot the torso is taken as a static basis with two robot manipulators as its legs. When a foot is in the support phase there are ground contact forces acting on this foot (the tip of the robot manipulator). These ground contact forces are estimated to determine the forces at the feet of the robot and these estimations are than used to calculate the required gravity compensation torques. The obtained algorithm is implemented in a Matlab, SimMechanics simulation model and simulations are performed with the robot in different static positions to verify if the obtained gravity compensation algorithm is helping the robot.

From the simulation results it can be concluded that the obtained gravity compensation algorithm does not calculate satisfactory values. The reason for this is probably that the torso of the robot can not be taken as a static basis. Namely it is assumed that the forces guided through the torso from one leg to the other can be neglected. The forces guided through the torso are probably of significant matter that they can not be neglected. For the future it is recommended to investigate if it is possible to include the forces guided through the torso from one leg to the other. Another approach could be to leave some of the joints unactuated so that the robot is no longer over actuated.

Symbols

Symbol	Meaning	Unit
C	Matrix with the centrifugal and Coriolis terms	-
CoM	Center of mass	-
CoM'	Ground projection of the CoM	-
D	Inertia matrix	-
d	Distance between two points	m
$e_{\%}$	Percentage error	%
f	Vector of forces and moments on a manipulator tip	N / Nm
	Force	N
F	Force	N
g	Gravitational constant	m/s^2
	Gravity compensation vector	-
J	Manipulator Jacobian	m / -
l	Length of a link	m
M	total mass of the robot	kg
m	Mass	kg
n	Moment	Nm
o	Position vector from the basis of a manipulator to the end of a link	m
P	Potential energy	-
q	Joint angle	$degrees$
r	Position vector	m
u	Value of the feedback controller	-
z	Vector which determines the axis of rotation/translation of joint	-
α	Distribution factor for distribution for the ground contact forces	-
θ	Angle	rad
τ	Joint torque	Nm

Subscripts	Meaning	Example
c	Center of mass	r_c
$distance$	Distance between two points	$z_{distance}$
hip	Coordinate of the hip	z_hip
l	Left	F_l
r	Right	F_r
res	Resulting	F_res
v	Linear velocity	J_v
w	Friction	F_w
ω	Rotational velocity	J_ω

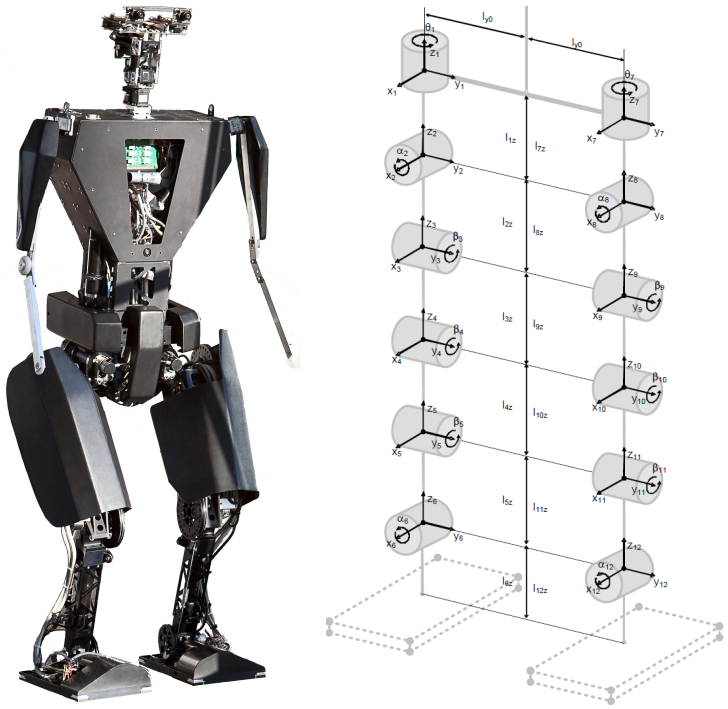
Contents

Chapter 1

Introduction

Humanoid robots are a great topic of research. All over the world universities and institutions are doing research on humanoid robots with the goal to be able to make a robust, autonomous, user friendly and human like robot. One of reasons why human like robots are interesting is that they have the ability to walk. The great advantage of a robot being able to walk is that it can move around in an environment especially designed for humans, such as homes, offices and hospitals. It can for instance go over doorsteps and climb the stairs. The working space of a robot which can not walk but has wheels to move around is limited by for example doorsteps or stairs.

At the Eindhoven University of Technology research is being done on bipedal humanoid robots. The robot which is used as experimental platform is TUlip [1]. TUlip has two legs. Each leg has 6 revolute joint actuators, so each leg has 6 degrees of freedom. There are three actuators in the hip for rotations around the x-, y- and z-axis, one in the knee for rotation around the y-axis and two in the ankle for rotations around the x- and y-axis, see Figure 1.1(b). The motors that are used are Maxon DC motors and all motors have optical encoders to measure the position of each joint. The goal is to let TUlip walk like a human.



(a) (b)

Figure 1.1: Picture of TUlip (a) and a kinematic representation of the lower body of TUlip (b)

At the moment the robot is able to walk statically stable, which means that it always keeps its center of mass above the stance foot. For walking, pre-computed joint reference trajectories are used and the joint references are tracked by local joint feedback controllers. A feedforward gravity compensation algorithm can be used to facilitate the feedback controllers. At the moment feedforward gravity compensation is used during the entire gait. There is, however, a problem when the robot is in double support phase (two feet on the ground). In the double support phase the robot is over actuated. This is because when both feet are on the ground the robot loses some degrees of freedom. With the current gravity compensation of TUlip the joints in the hip are counteracting each other, so too much energy is consumed during the double support phase. *The goal of this project is to design a feedforward algorithm that solves the problem of over actuation in the double support phase. The algorithm should be verified in a simulation model and to implement it on the robot.*

In the literature different approaches for gravity compensation are proposed. Most of the articles describe algorithms to compute the torques to compensate the effect of gravity. In [3] a self learning gravity compensation, in which a lookup table is made for the desired torques needed for the gravity compensation, is explained. An advantage of such approach is that it is not necessary to know the dynamic model of the robot. However, during the self learning phase the robot has to be supported to prevent it from falling and thus from breaking. Another disadvantage is that the self learning part has to be repeated for every new behavior because the configuration of the robot may not be present in the lookup table.

In [4] a gravity compensation is designed using an optimization algorithm. This gravity compensation facilitates the balancing of the robot during walking, but there are some disadvantages. The algorithm is only tested in simulations so possibly the optimization is not fast enough for implementation on a real robot. But the biggest disadvantaged is that the designed algorithm can only be applied to one certain speed of walking, because there is a pre determined constant in the algorithm that has to be changed for different walking speeds.

In the approach of [5] the feedback controller exists of a PD feedback controller and a non-integrated type of feedback of gyroscope signals. The gyroscope is attached to the torso of the robot. The PD feedback part is used to robustly compensate the stationary error. The non-integrated feedback of the gyroscope signals is used for improving the transient response characteristics of the torso of the robot. The total feedback controller compensates for the gravity effects on the torso of the robot. The goal of this project is to design a gravity compensator to facilitate the feedback controllers for the joints in the legs of the robot. The gravity compensator from the research of [5] is only designed to keep the torso in upright position, so it is not suitable for this project.

In [6] a balancing controller is designed. This controller consists of a part for gravity compensation and a part for balancing control. The part for gravity compensation uses the manipulator Jacobian to compute the desired joint torques. In the single support phase the Jacobian from the CoM (Center of Mass) to the ZMP (Zero Moment Point) is used. For the double support case an algorithm is designed to get to the desired joint torques. In this algorithm the ground contact forces, acting on the corners of each foot, are estimated using the desired ZMP. When the contact forces are known the Jacobians from the CoM to the different contact points are used to calculate the desired joint torques. The balancing controller is tested in simulations and on a robot for respectively a step to the front and a step to the side. The conclusion is that the controller balances the robot. However, in the article the control effort is not evaluated and no comparison is given between the controller with and without the gravity compensation.

In [7] a different approach is used. A physical system with springs in the legs of the robot to facilitate the controller for the knee. This designed gravity compensator works only for the knee joint and is a hard-ware solution. In this project the aim is to design a gravity controller which facilitates all the joints. Moreover we do not change the hardware of TUlip. So this approach is not helpful at this time but it may be helpful for the future.

The approach from [6] could be a good approach to design a feedforward algorithm to facilitate the feedback controller of TUlip. The calculations using a Jacobian are taken as a basis for designing a gravity compensating algorithm for the double support case.

This report is organized as follows. How the Jacobian can be used to calculate the desired joint torques is explained in Chapter 2 The design of our algorithm is explained in the Chapter 3. For the completeness of the report also the approach for the single support case is presented, this is done in Section 3.2. After designing the algorithm for the gravity compensation the algorithm is implemented in a 3D dynamical simulation of TUlip model in Matlab [8], SimMechanics [9]. With this simulation model static and dynamic simulations are performed. The implementation and the results of the simulations are presented and discussed in Chapter 3.3. Finally a conclusion is drawn and recommendations are give in Chapter 4.

Chapter 2

Joint torque calculation with manipulator Jacobian

In this chapter it is explained what the manipulator Jacobian is and how it can be used to calculate the required joint torques when a certain forces is desired at the tip of a robot manipulator. After that the approach to design a feedforward controller, which facilitates a feedback controller in the case where a force is applied on the tip of a robot manipulator, is explained. This feedforward controller is also tested for a simple 2 link robot.

2.1 Torque calculations with the manipulator Jacobian

When a certain force is desired at the tip of a robot manipulator, for instance a robot arm, the required joint torques to provide this force can be calculated via the manipulator Jacobian. The manipulator Jacobian is the matrix which relates the velocity of the end effector to the velocities of the joints. The derivation of the manipulator Jacobian can be found in [10] and it is constructed using the following equations, for a robot with n joints:

$$J = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix}, \text{ with} \tag{2.1}$$

$$J_v = \begin{bmatrix} j_{v,1}, & j_{v,2}, & \ldots, & j_{v,n} \end{bmatrix}, \; j_{v_i} = \begin{cases} z_{i_1}^0 \times (o_n^0 - o_{i-1}^0) & \text{for revolute joint } i \\ z_{i-1}^0 & \text{for prismatic joint } i \end{cases} \tag{2.2}$$

$$J_\omega = \begin{bmatrix} j_{\omega,1}, & j_{\omega,2}, & \ldots, & j_{\omega,n} \end{bmatrix}, \; j_{\omega_i} = \begin{cases} z_{i-1}^0 & \text{for revolute joint } i \\ 0_3 & \text{for prismatic joint } i \end{cases} \tag{2.3}$$

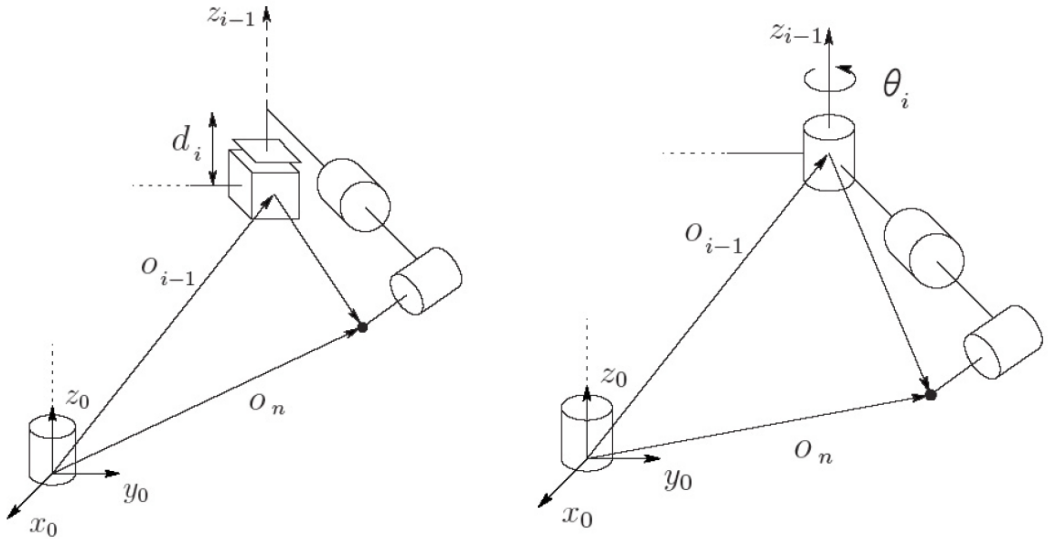
where J_v is the part of the Jacobian related to the linear velocity of the end effector and J_ω to the angular velocity. The vector z_{i-1} determines the axis of rotation or translation for joint i, for example a rotation around the z-axis gives $z_{i-1} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$. The vector o_{i-1} is the position vector from the basis of the manipulator to the end of link i and o_n is the vector from the basis of the manipulator to the end effector. Vectors z_{i-1}, o_{i-1} and o_n are indicated in Figure 2.1.

So when a certain force f is desired at the tip of a robot the required joint torques τ can be calculated by the Jacobian using the following equation [10]:

$$\tau = J^T(q) f, \tag{2.4}$$

where τ is a vector $\tau = \begin{pmatrix} \tau_1, & \ldots, & \tau_n \end{pmatrix}^T$ with τ_i is the required torque in joint i, $J(q)$ is the manipulator Jacobian, depending on joint positions q, and $f = [f_x, f_y, f_z, n_x, n_y, n_z]^T$, with f_j and n_j respectively the forces and torques in direction j.

For the calculations of the joint torques in a algorithm the Jacobian is first calculated parametrically depending on the joint angles. The joint angles are the only varying parameter in the Jacobian. With this parametrical Jacobian only the parametrical joint angles have to be substituted by numerical



(a) Prismatic joint

(b) revolute joint

Figure 2.1: Schematic representation of a robot manipulator with z_{i-1}, o_{i-1} and o_n indicated in the figure

values to obtain a numerical Jacobian for different positions of the robot manipulator. In this way the parametrical Jacobian only has to be calculated once, which speeds up the algorithm. With the Jacobian and the desired forces on the end effector f the required joint torques τ can be calculated using (2.4).

2.2 Approach to calculate the required joint torques with the manipulator Jacobian

When it is desired for a robot manipulator to stay in a desired position a simple stabilizing PD-controller can be applied. If a certain force is desired at the tip of this robot manipulator, the manipulator Jacobian can be used to calculate the required joint torques to provide this force. When the forces on the end effector are known it is possible to design a feedforward algorithm, with the theory from Section 2.1. Such a feedforward controller can reduce the controller effort from the feedback controller to zero if no disturbances are present. To design the required feedforward controller the following steps have to be taken,

1. The Jacobian has to be constructed using (2.1) to (2.3) and should be depended on the joint positions q, so that it can be calculated fast for different configurations of the robot arm.
2. When the desired position is known this can be substituted into the parametric Jacobian. When also the desired forces at the tip of the robot arm are know, the required joint torques can be calculated using (2.4). The force vector f should be of opposite sign to the forces acting on the end effector, because to keep the robot in the desired position the robot arm should counteract the forces that are applied to it.

2.3 Implementation on a 2 link robot arm

To show that the approach from the previous section can be used to design a feedforward controller, which facilitates the feedback controller, the approach is implemented for a two link robot arm. The robot arm consist of two revolute joints, the first rotating around the z-axis and the second rotating around the x-axis, see Figure 2.2.

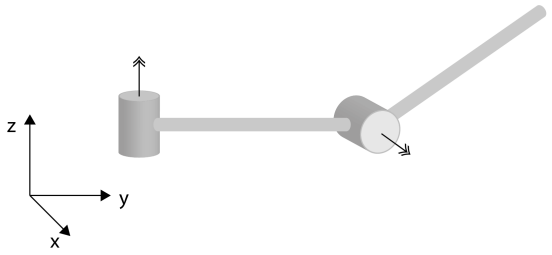


Figure 2.2: Drawing of the 2 link robot used for the simulation

With SimMechanics a simulation model is constructed of the 2 link robot stated above. A feedback controller is designed to keep the robot arm in a desired position. The simulation model and the m-file for the feedforward controller are included in Appendix A. With the simulation model two simulations are performed, one without and one with the feedforward controller, to show the effect of the feedforward controller. The initial and reference positions for the two joint angles and the forces on the robots tip are stated in Table 2.1. Arbitrary tip forces are used and no gravity is included, to simplify the analysis.

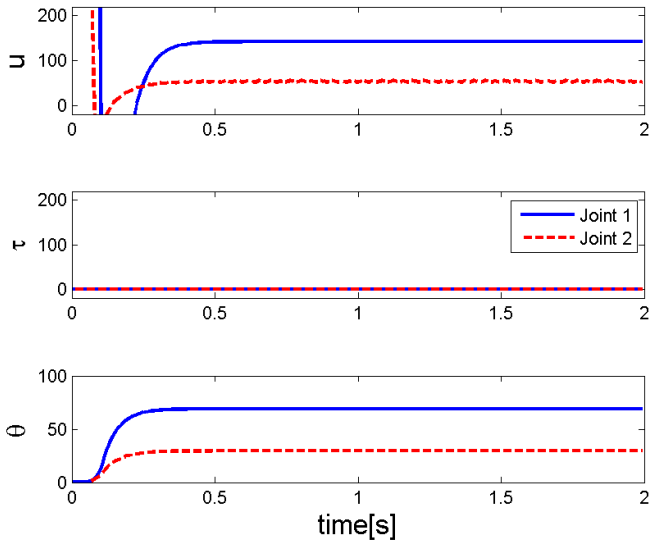
Table 2.1: Initial and reference joint positions for the simulations with the 2 link robot

Angle	**Joint 1 [degrees]**	**Joint 2 [degrees]**	
Initial position	0	0	
Reference position	70	30	
Tip forces	**x-direction [N]**	**y-direction [N]**	**z-direction [N]**
	102	53,5	10

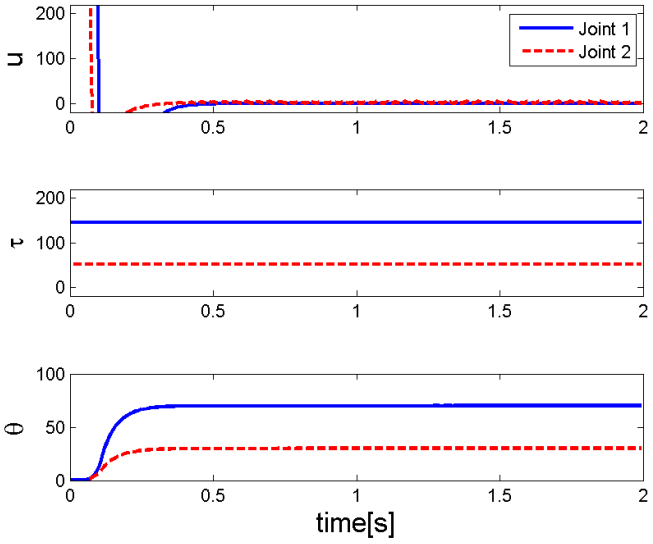
The results of the simulations without and with the feedforward controller are plotted in Figure 2.3. Both figures contain a plot with the feedback controller values u, the feedforward value τ and the joint angles θ. After 0.3 seconds the robot manipulator reaches steady state. It is clear to see that in the simulation with the feedforward controller the value for the feedback controller becomes almost zero in steady state. There is only a small controller value needed to keep the robot arm in the reference position. From these simulations it can be concluded that the proposed algorithm for the feedforward controller facilitates the feedback controller. This approach can be used for designing a feedforward controller for TUlip.

2.4 Summary

In this chapter it is explained how the manipulator Jacobian can be used to calculate the desired forces when a certain force is desired at the tip of a robot manipulator. First it was explained how the manipulator Jacobian can be constructed and how it can be used to calculate the desired joint torques. With the manipulator Jacobian a feedforward algorithm is obtained to calculate the required joint torques. Finally the obtained feedforward controller is implemented in a Simulink model to verify that it facilitates the feedback controller. From the simulation results it can be concluded that the obtained feedforward controller can be used to obtain a feedforward controller for TUlip.



(a) Simulation results without feedforward controller. The first plot shows feedback controller value u, the second plot shows the feedforward value τ and the third plot shows the joint angle θ



(b) Simulation results with feedforward controller. The first plot shows feedback controller value u, the second plot shows the feedforward value τ and the third plot shows the joint angle θ

Figure 2.3: Results from the simulations with the 2 link robot

Chapter 3

Gravity compensation a for bipedal robot

With the approach from the previous chapter a feedforward controller can be designed for the double support phase (two legs of the robot on the ground) of a walking gait. In this chapter the approach to design such a feedforward controller is explained. First the approach to design this feedforward controller is explained in the the 2D for the ease of understanding. After that the 2D approach is extended to 3D. At last the approach to design a feedforward controller for the single support phase (one foot on the ground) is explained.

3.1 Approach for double support

3.1.1 2D approach

For the calculations of the required joint torques, the base frame is put in the robot's torso. The two legs can be seen as two manipulators. When a foot of the robot touches the ground, ground contact forces are acting on that foot. The foot can be seen as the manipulator tip. These ground contact forces are a result of gravitational effects acting on the robot. With the forces acting on the feet and the manipulator Jacobian known, the joint torques required to keep the robot in a certain position can be calculated using (2.4).

In Figure 3.1 a 2D schematic representation of a robot is drawn. Because the robot is in the double support phase both feet are on the ground and there are forces acting on each foot. On each foot there are two forces acting; a force due to the gravity (F_l and F_r) in normal direction and a friction force ($F_{w,l}$ and $F_{w,r}$) in the tangential direction. The point where the forces act is the point under the ankle. This point is chosen because the robots feet are assumed to stay flat on the ground so that the sum of the forces acting on the feet are guided through the feet onto the ankles.

Ground contact forces due to the gravity

The distribution of the two normal forces, due to the gravity, between both feet can be calculated using the following equations. This is a simple static force/moment balance which comes from [10]:

$$F_r = \alpha M g, \tag{3.1}$$

$$F_l = (1-\alpha) M g, \tag{3.2}$$

$$\alpha = \frac{d_l}{d_l + d_r}. \tag{3.3}$$

In these equations M is the total mass of the robot, g the gravitational constant, d_l and d_r respectively the distances between CoM', the projection of the center of mass on to the ground, and the left and right foot and α the factor which distributes the ground contact force between the two feet. So when $\alpha = 1$ the robot is totally supported by its right foot and when $\alpha = 0$ the robot is totally supported by its left foot.

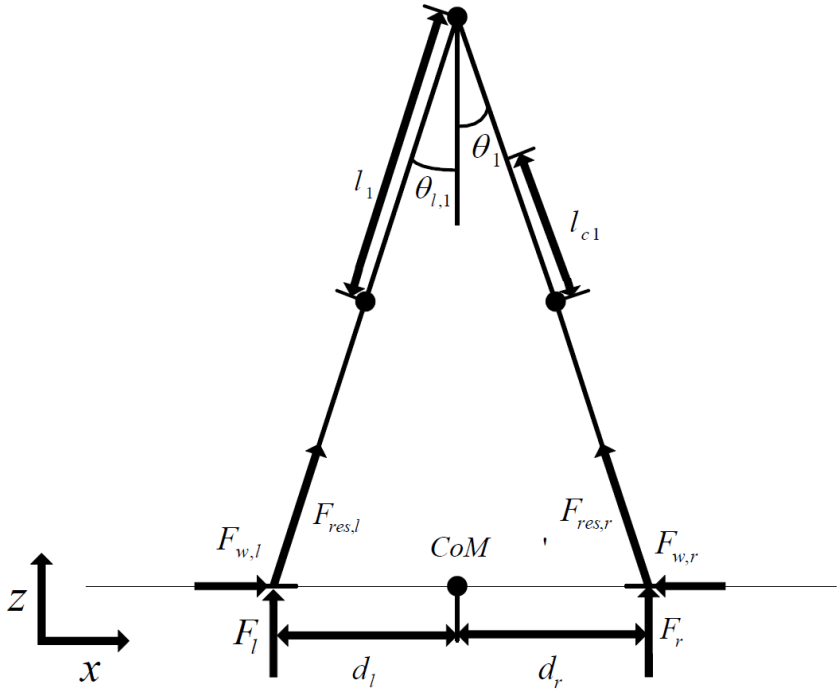


Figure 3.1: Schematic representation of the robot with the forces acting on it in 2D

The distances d_r and d_l should be known to calculate α. To calculate d_r and d_l the following formulas should be used:

$$d_{tot} = \sin(\theta_{r,1})l_1 + \sin(\theta_{r,1} + \theta_{r,2})l_2 - \sin(\theta_{l,1})l_1 - \sin(\theta_{l,1} + \theta_{l,2})l_2, \tag{3.4}$$

$$d_r = \frac{1}{M}\sum m_{i,j} r_{x,i,j}, \tag{3.5}$$

$$d_l = d_{tot} - d_r, \tag{3.6}$$

where $d_{tot} = d_r + d_l$ is the distance between the two feet. Angle $\theta_{i,j}$ is the joint angle, with i the leg and j the joint number. Angles are taken positive from x to z, joint 1 is the hip joint and joint 2 is the knee joint. Length l_i is the length of the two links, with link 1 the upper leg and link 2 the lower leg. Mass M is the total mass of the robot, $m_{i,j}$ the mass of link i, j and $r_{x,i,j}$ the x-component of the position vector from the right foot of the center of mass of link i, j. For example $r_{x,r,1} = \sin(\theta_{r,1})lc_1 + \sin(\theta_{r,1} + \theta_{r,2})l_2$, with lc_1 is the length from joint 2 to the CoM of link 1.

Ground contact forces due to friction

In the current gravity compensator the torques in the hip are counteracting each other, so the resulting torque on the torso is zero. It is preferred that these torques become zero so that no energy is dissipated in the joints counteracting each other. When the resulting ground forces $F_{res,l}$ and $F_{res,r}$ point into the hip there is no resulting moment around the hip induced by the ground contact forces. Thus the torques in the hips are zero when the robot is in upright position. This gives a way to estimate the friction force acting on the feet of the robot. The friction force which leads to a resulting ground forces which is pointing into the hip can be calculated using,

$$F_{w,i} = \tan(\theta_i)F_i \tag{3.7}$$

Where $F_{w,i}$ is the coulomb friction force on a foot, θ is the angle between the line from the hip to the point where the forces act on the foot and F_i is the ground contact force due to the gravity.

The angle θ can be calculated using,

$$z_{hip,i} = \cos(\theta_{i,1})l_1 + \cos(\theta_{i,1} + \theta_{i,2})l_2 \tag{3.8}$$

$$x_{hip,i} = \sin(\theta_{i,1})l_1 + \sin(\theta_{i,1} + \theta_{i,2})l_2 \tag{3.9}$$

$$\theta_i = \arctan\left(\frac{x_{hip,i}}{z_{hip,i}}\right) \tag{3.10}$$

where $z_{hip,i}$ is the vertical distance from the hip to foot i and $x_{hip,i}$ the horizontal distance from the hip to foot i.

When we apply the theory from [10], the Jacobian of the legs of the robot can be derived. With this Jacobian and the forces acting on the robots feet, (2.4) can be used to calculate the required joint torques to keep the robot in the given position. In the 2D case the values of f_y and n_y from (2.4) will be zero because no force and moment are acting is this direction.

Algorithm for calculating the joint torques

With the given equations from above the required torques in the joints can be calculated as follows:

1. The vertical distance between the feet has to be calculated. This is required to determine if the robot is in double or single support, if the vertical distance between the two feet is zero the robot is in double support. The vertical distance between the feet and the hip can be calculated using (3.8) and with (3.11) the distance between the two feet can be calculated.

$$z_{distance} = z_{hip,r} - z_{hip,l} \tag{3.11}$$

2. The distances d_r and d_l have to be calculated using (3.5) and (3.6). These values are required to calculate α using equation (3.3), required to calculate the ground contact forces acting on the feet in normal direction. This calculation can be done when the position and the dimensions of the robot are known using trigonometry.

3. Next the angle θ has to be calculated using (3.8) to (3.10). This angle is required to estimate the friction forces in such a way that the resulting force acting on each foot is pointing into the hip.

4. With α and θ known first (3.1) and (3.2) are used to calculate the ground contact forces in normal direction. Next (3.7) can be used to estimate the friction force. With the Jacobian and all the ground contact forces known, the required joint torques can be calculated using (2.4).

3.1.2 3D approach

In the 2D approach there are ground forces acting on each foot in two directions, namely one normal (z-direction) and one tangential (x-direction) to the leg of the robot. When the problem is extended to 3D another tangential friction force (y-direction) is added. So the friction forces act in the xy-plane and the normal force in the z-direction. When the model is extended to 3D the basic algorithm stated in Section 3.1.1 stays the same. The calculations in step 1, 2 and 3 become more complicated because the trigonometric calculations now have to be applied in 3D instead of in 2D. The calculations for the ground projection of the CoM becomes different in the 3D. The CoM does not have to be above the line between the two ankles of the robot. When the CoM is not above the line between ankles the CoM' is projected onto this line for the calculations of the factor α, see Figure 3.2.

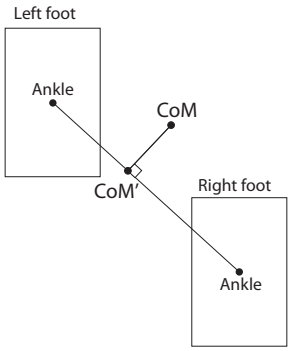


Figure 3.2: Schematic representation the projected center of mass for the 3D case

3.2 Approach for single support

When the robot is in single support, hence one foot on the ground and one in the air, the robot can be seen as a normal robot manipulator with its base in the foot that is on the ground. The torso, head and arms of the robot are in this case modeled as one link. For such a robot manipulator the theory from [10] can be used. Here the equation of motion are derived using the Euler-Lagrange equations and this leads to the following equation,

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau, \tag{3.12}$$

where $D(q)$ is the inertia matrix, $C(q,\dot{q})$ is the matrix which contains the centrifugal and Coriolis terms, $g(q)$ is the gravity vector, τ is the input torque and q are the positions of the joints. The vector $g(q)$ is the only part from (3.12) which remains when the robot is in steady state, because $\dot{q}$ and $\ddot{q}$ are zero. In (3.12) $g(q)$ is the gravity vector and this is the vector which contains the required torques for gravity compensation. This vector is derived from the potential energy P, of the robot,

$$P = \sum_{i=1}^{n} m_i g^T r_{ci}, \tag{3.13}$$

with m_i the mass of link i, g the gravity vector giving the direction of the gravity in the inertial frame and r_{ci} the position vector of the center of mass of link i. The gravity vector is simply the Jacobian of the potential energy function (3.13),

$$g_k(q) = \frac{\partial P}{\partial q_k} \tag{3.14}$$

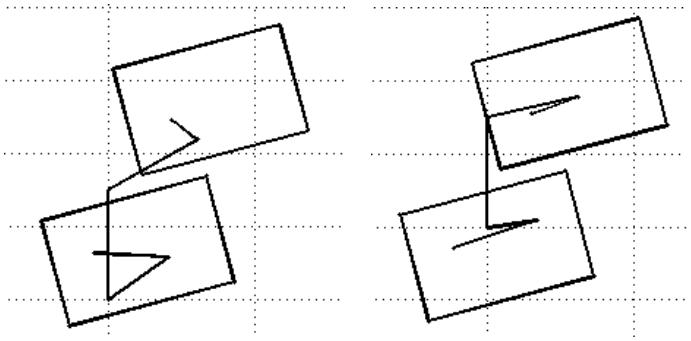
$$g(q) = [g_1(q), ..., g_2(q)]^T \tag{3.15}$$

here k is the k^{th} joint of the robot manipulator.

3.3 Implementation of the gravity compensation algorithm

The designed algorithm for the double support case from Section 3.1.1 is implemented in a simulation model from TUlip in Matlab [2], SimMechanics [3]. The simulation model and the m-files with the algorithm for the feedforward gravity compensation are presented in Appendix B. The results from these simulations are presented in this chapter.

For the static simulations, simulations in which the robot stays in one positions and does not walk, eight different positions are evaluated. These positions are positions from the double support phase of the walking gait for TUlip, designed in [9]. The first position is at the beginning of the double support phase and the last position is at the end, with the other six in between. A schematic top view of the first and eight position are shown in Figure 3.3. All the eighth position are shown in Appendix C along with a table with the joint angles for the different positions.



(a) Position 1, begin of the double support phase

(b) Position 8, end of the double support phase

Figure 3.3: Top view of the schematic representation of the first and eight position for the simulations

3.3.1 Results from simulations with estimated ground contact forces

Table 3.1 shows results from the simulations with the eight different positions. This table contains the controller value needed to keep the robot in a certain position without the gravity compensation, the values calculated by the obtained gravity compensation algorithm and the percentage error between these two values, calculated using the following equation:

$$e_{\%} = \frac{u - \tau}{u} 100\%, \tag{3.16}$$

with u the value from the feedback controller without gravity compensation and τ the value from the gravity compensation algorithm. Also graphical representations of the the controller and gravity compensation values are shown in Appendix D.

In Table 3.1 joints 1 to 6 are the joints from the right leg, from hip to ankle, and joints 7 to 12 are the joints from the left leg, from hip to ankle. It is clear that almost all the values for the percentage error are far from zero. For the first and the seventh joint the percentage error is very large. This is due to the fact that the actual values are small and a small deviation in the gravity compensation value results in a large percentage error. If the feedforward gravity compensation algorithm would calculate satisfactory values the percentage error would be close to zero. It is clear that this is not the case so the obtained algorithm is not suitable.

3.3.2 Results from simulations with measured ground contact forces

A reason why the obtained gravity compensating algorithm is not suitable could be that the estimation of ground contact forces are not correct. Another set of simulation is performed with the same simulation model, but instead of estimating the ground contact forces they are measured and used in the algorithm for the gravity compensation.

In Table 3.2 the simulation results from the simulations with the measured ground contact forces are stated. In Appendix E the simulation are shown graphically. It is clear that again the values for the percentage error are far from zero. For joints 5, 6, 7, 11 and 12 the values of the percentage error are close to 100, this is because the values from the gravity compensation algorithm are much smaller than the values of the feedback controller. Again it can be concluded that the gravity compensation does not calculate satisfactorily values. So the mistake in the algorithm is not only that the estimation of the ground contact forces is not correct.

3.3.3 Conclusion on the gravity compensation algorithm

In the previous two sections the result from the simulations are presented. From the results of Section 3.3.1 it can be concluded that the obtained gravity compensation algorithm, with estimation of the ground contact forces, does not give satisfactory estimations of the desired joint torques. From the results of Section 3.3.2 it can be concluded that when the ground contact forces are measured the

Table 3.1: Simulation results from simulation with estimated forces: the gravity compensation value GC, the controller value and the error difference between them for the different joints and the different positions

Position	**Joint 1**			**Joint 2**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	8,45E-03	4,58E-05	18322,88	-0,2482	-0,2033	22,11
2	6,00E-03	4,08E-05	14607,31	-0,2339	-0,0998	134,31
3	3,27E-03	3,12E-05	10357,17	-0,1576	-0,0676	133,08
4	2,07E-03	1,25E-05	16465,48	-0,0630	-0,0304	107,32
5	1,56E-03	7,37E-06	21116,59	-0,0415	-0,0213	95,39
6	4,31E-04	-1,05E-06	41239,49	-0,0092	-0,0070	32,35
7	-1,96E-03	-1,27E-05	15356,87	0,0312	0,0124	151,59
8	-4,81E-03	1,36E-04	3628,47	0,0606	0,0852	28,84
Position	**Joint 3**			**Joint 4**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	0,0670	0,0326	105,62	-0,1578	-0,0931	69,48
2	0,0630	0,0057	1006,53	-0,1619	-0,1008	60,58
3	0,0424	-0,0003	14467,25	-0,1650	-0,1019	61,91
4	0,0170	-0,0116	246,33	-0,1559	-0,0977	59,52
5	0,0112	-0,0149	174,80	-0,1518	-0,0965	57,37
6	0,0025	-0,0204	112,16	-0,1436	-0,0937	53,23
7	-0,0084	-0,0282	70,30	-0,1282	-0,0881	45,49
8	-0,0161	-0,0369	56,42	-0,1111	-0,0477	133,18
Position	**Joint 5**			**Joint 6**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	0,0063	0,0368	82,87	-0,0074	0,0482	115,32
2	0,0053	0,0089	40,54	-0,0071	0,0930	107,63
3	0,0038	0,0052	27,54	-0,0047	0,0639	107,42
4	0,0034	0,0062	44,87	-0,0015	0,0335	104,49
5	0,0035	0,0065	47,18	-0,0008	0,0274	102,74
6	0,0035	0,0071	50,50	0,0004	0,0183	97,87
7	0,0036	0,0079	53,92	0,0018	0,0067	72,89
8	0,0036	-0,0188	119,41	0,0029	0,0776	96,33
Position	**Joint 7**			**Joint 8**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	-1,50E-02	2,38E-06	628042,38	-0,1032	-0,0478	115,75
2	-1,54E-02	1,79E-05	86042,68	-0,1004	-0,0532	88,57
3	-1,62E-02	2,31E-05	70130,60	-0,1022	-0,0579	76,52
4	-1,41E-02	2,41E-05	58501,71	-0,0964	-0,0539	78,83
5	-1,29E-02	2,44E-05	53073,04	-0,0916	-0,0523	74,99
6	-1,06E-02	2,43E-05	43615,52	-0,0801	-0,0479	67,18
7	-6,22E-03	2,24E-05	27868,06	-0,0536	-0,0365	46,69
8	-1,48E-03	-1,74E-04	754,54	-0,0149	0,1424	110,49
Position	**Joint 9**			**Joint 10**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	0,0256	-0,0085	402,14	-0,0315	-0,0217	45,41
2	0,0249	0,0090	178,16	-0,0323	-0,0375	13,83
3	0,0259	0,0123	110,10	-0,0456	-0,0447	2,04
4	0,0251	0,0197	27,60	-0,0726	-0,0514	41,13
5	0,0240	0,0220	9,15	-0,0808	-0,0523	54,31
6	0,0212	0,0256	17,35	-0,0953	-0,0540	76,58
7	0,0143	0,0304	52,88	-0,1193	-0,0568	109,97
8	0,0040	0,0149	72,97	-0,1448	-0,1476	1,88
Position	**Joint 11**			**Joint 12**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	-0,0028	-0,0152	81,86	-0,0042	-0,0226	81,54
2	-0,0031	-0,0241	87,30	-0,0041	0,0153	126,93
3	-0,0042	-0,0272	84,48	-0,0044	0,0028	259,20
4	-0,0056	-0,0244	76,92	-0,0045	-0,0100	55,05
5	-0,0060	-0,0225	73,52	-0,0044	-0,0145	69,70
6	-0,0064	-0,0188	65,95	-0,0041	-0,0216	81,05
7	-0,0069	-0,0123	43,95	-0,0033	-0,0315	89,64
8	-0,0071	-0,0491	85,45	-0,0020	0,1077	101,83

Table 3.2: Simulation results from simulation with measured forces: the gravity compensation value GC, the controller value and the error difference between them for the different joints and the different positions

Position	**Joint 1**			**Joint 2**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	5,77E-07	4,58E-05	98,74	-0,0610	-0,2033	70,01
2	-4,36E-08	4,08E-05	100,11	-0,0475	-0,0998	52,40
3	-2,42E-07	3,13E-05	100,77	-0,0327	-0,0676	51,71
4	-1,43E-08	1,25E-05	100,11	-0,0161	-0,0304	47,08
5	-1,27E-08	7,36E-06	100,17	-0,0123	-0,0213	42,10
6	-9,45E-10	-1,05E-06	99,91	-0,0064	-0,0070	7,55
7	-2,03E-08	-1,27E-05	99,84	0,0014	0,0124	88,46
8	-5,17E-08	1,36E-04	100,04	0,0030	0,0852	96,45
Position	**Joint 3**			**Joint 4**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	-0,0030	0,0326	109,09	-0,0891	-0,0931	4,37
2	0,0005	0,0057	90,76	-0,0741	-0,1008	26,53
3	-0,0006	-0,0003	97,95	-0,0723	-0,1019	29,00
4	-0,0092	-0,0116	20,50	-0,0710	-0,0977	27,39
5	-0,0118	-0,0149	20,76	-0,0705	-0,0965	26,89
6	-0,0162	-0,0204	20,38	-0,0694	-0,0937	25,89
7	-0,0227	-0,0282	19,48	-0,0668	-0,0881	24,12
8	-0,0104	-0,0369	71,86	-0,0235	-0,0477	50,58
Position	**Joint 5**			**Joint 6**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	5,0924E-06	0,0368	99,99	9,8079E-07	0,0482	100,00
2	-3,7555E-08	0,0089	100,00	-2,9793E-09	0,0930	100,00
3	-2,8838E-07	0,0052	100,01	-2,5841E-08	0,0639	100,00
4	-3,4150E-08	0,0062	100,00	-3,3150E-09	0,0335	100,00
5	-4,0611E-08	0,0065	100,00	-3,8905E-09	0,0274	100,00
6	-7,7403E-09	0,0071	100,00	-8,6259E-10	0,0183	100,00
7	1,4922E-07	0,0079	100,00	1,3627E-08	0,0067	100,00
8	2,8951E-06	-0,0188	100,02	1,3588E-06	0,0776	100,00
Position	**Joint 7**			**Joint 8**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	4,52E-09	2,38E-06	99,81	-0,0113	-0,0478	76,39
2	-5,71E-10	1,79E-05	100,00	-0,0205	-0,0532	61,57
3	1,17E-09	2,31E-05	99,99	-0,0174	-0,0579	69,99
4	-8,23E-10	2,41E-05	100,00	-0,0120	-0,0539	77,68
5	-1,09E-09	2,44E-05	100,00	-0,0103	-0,0523	80,36
6	-3,78E-10	2,43E-05	100,00	-0,0071	-0,0479	85,26
7	1,92E-08	2,24E-05	99,91	-0,0013	-0,0365	96,35
8	3,36E-06	-1,74E-04	101,94	0,0091	0,1424	93,59
Position	**Joint 9**			**Joint 10**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	0,0170	-0,0085	301,13	-0,0042	-0,0217	80,47
2	0,0343	0,0090	282,03	-0,0075	-0,0375	80,00
3	0,0385	0,0123	212,64	-0,0100	-0,0447	77,62
4	0,0402	0,0197	104,77	-0,0168	-0,0514	67,25
5	0,0400	0,0220	81,99	-0,0190	-0,0523	63,69
6	0,0391	0,0256	52,46	-0,0230	-0,0540	57,44
7	0,0364	0,0304	19,74	-0,0299	-0,0568	47,42
8	0,0516	0,0149	246,71	-0,0600	-0,1476	59,32
Position	**Joint 11**			**Joint 12**		
	GC	**Controller**	**Difference**	**GC**	**Controller**	**Difference**
1	7,9775E-07	-0,0152	100,01	1,6195E-07	-0,0226	100,00
2	-4,3620E-10	-0,0241	100,00	-7,8019E-12	0,0153	100,00
3	1,6359E-09	-0,0272	100,00	-1,3411E-10	0,0028	100,00
4	-4,1146E-10	-0,0244	100,00	-2,4525E-10	-0,0100	100,00
5	-8,8701E-10	-0,0225	100,00	-2,9594E-10	-0,0145	100,00
6	1,0526E-12	-0,0188	100,00	-1,7764E-10	-0,0216	100,00
7	7,8620E-08	-0,0123	100,00	8,4977E-09	-0,0315	100,00
8	1,3103E-05	-0,0491	100,03	5,4992E-06	0,1077	99,99

algorithm does not give satisfactory estimations of the desired joint torques either. So there must be a mistake in the derivation of the algorithm. The assumption that the torso of the robot can be taken as a static base with two robot manipulators as legs is probably wrong. When the robots torso is assumed to be a static base no forces are guided from one leg to the other via the torso. These forces are probably of significant matter so they cannot be neglected.

3.4 Summary

In this chapter the feedforward gravity compensation algorithm is obtained and the results from the simulations are presented. First the approach to get to the feedforward algorithm is explained in 2D for the ease of understanding. It is explained how the ground contact forces due to gravity en due to friction can be estimated. Next the obtained algorithm is extended for the 3D case and this algorithm is implemented in a Simulink model. With this model several static simulations are performed, with the ground contact forces estimated and measured, and the results from these simulations are presented. From these results it is concluded that the obtained algorithm does not give satisfactory estimations of the desired joint torques.

Chapter 4

Conclusion and recommendations

The goal of this project was to design a feedforward gravity compensation algorithm which can facilitate the feedback controller for the humanoid robot TUlip. First some literature was reviewed. In one of the articles the manipulator Jacobian is used for torque calculations which stabilize the robot. The calculations with the manipulator Jacobian are taken as a basis in designing the gravity compensation algorithm. In Chapter 2 it is explained how the manipulator Jacobian can be constructed. It is also shown how it can be used to calculate the desired joint torques when a certain force is desired on the tip of a robot manipulator. In Chapter 3 the feedforward algorithm for the gravity compensation is derived in 2D, for the ease of understanding, and this is extended to 3D. The gravity compensation algorithm is implemented in Matlab and static simulations are performed with a Matlab, SimMechanics simulation model. The results from these simulations are presented in Chapter 3.3.

From the simulation results in Chapter 3.3 it can be concluded that the designed feedforward gravity compensation algorithm does not calculate satisfactory values. Even when the estimation of the ground contact forces is left out and when the ground contact forces are measured the algorithm does not calculate satisfactory values. The mistake is probably in the assumption that the torso of the robot can be seen as a static base with two robot manipulators as separate legs. When this assumption is made the forces guided through the torso from one leg to the other are neglected. These forces are probably of significant matter so they can not be neglected.

For future work it might be recommended to look if it is possible to include the guidance of the forces through the torso in the algorithm. If this is not possible it might be an option to look at the possibility to leave some joints unactuated so that the robot loses some actuated degrees of freedom and the robot is not over actuated anymore.

Bibliography

[1] P.W.M. van Zutven, P. Mironchyk, C. Çilli, E. Steur, E. Ilhan, J. Caarls, R.S. Pieters, A. Filiz, P. Heijkoop, T. de Boer, A. Smorenberg, E. van Breda, G. Liqui Lung, F. Wilbers, C Plooij, G. van der Hoorn, S. Kiemel, R. Reinink, E. Dertien, G. van Oort, D. Kostic, H. Nijmeijer, M. Wisse, P.P. Jonker, R. Carloni, S. Stramigioli and T.P.H. Warmerdam, *Dutch Robotics 2011 adult-size team description*, in RoboCup 2011 Symposium papers and Team Description papers, Istanbul, Turkey, 7, 2011.

[2] G. Pratt and M. Williamson, *Series elastic actuators*, In Proc., Int. Conf. on Intelligent Robots and Systems, Pittsburgh, PA, 1995.

[3] M. Xie, W. Zhong, L. Zhang, H.J. Yang, C. S. Song, J. Li, L.B. Xian and L. Wang, *Self Learning of Gravity Compensation by LOCH Humanoid Robot*, IEEE ∼ RAS International Conference on Humanoid Robots, 2008, pp 320-325.

[4] O. Ayhan and K. Erbatur, *Biped Walking Robot Hybrid Control with Gravity Compensation*, IEEE International Conference on Industrial Electronics, 2005, pp 1797-1802.

[5] T. Sugihara and Y. Nakamura, *Gravity Compensation on Humanoid Robot control with Robust Joint Servo and Non-integrated Rate-gyroscope*, IEEE International Conference on Humanoid Robots, 2006, pp 194-199.

[6] S.H. Hyon and G. Cheng, *Gravity Compensation and Full-Body Balancing for Humanoid Robots*, IEEE International Conference on Humanoid Robots, 2006, pp 214-221.

[7] S. Hirata, A. Konno and M. Uchiyama, *Design and Evaluation of a Gravity Compensation Mechanisme of a Humaniod Robot*, IEEE/RSJ International Conference on Intelegent Robots and Systems, 2007, pp 3635-3640.

[8] P. van Zutven, *Modeling, identification and stability of humanoid robots*, MSc thesis, DCT 2009.100, Dynamics and control group, Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven University of Technology, 2009.

[9] S. van Dalen, *Walking Strategies for Bipedal Humanoid Robots*, MSc thesis (work in progress), Dynamics and control group, Department of Mechanical Engineering, Eindhoven University of Technology, 2011.

[10] M.W. Spong, S.Hutchinson and M. Vidyasagar, *Robot modeling and control*, John wiley & Sons, Inc, 2006, pp 129-153 & pp 255-257.

[11] S. Kajita, M. Morisawa, K. Miura, S. Nakaoka, K. Harada, K. Kaneko, F. Kanehiro and K. Yokoi, *Biped Walking Stabilization Based on Linear INverted Pendulum Tracking* IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp 4489-4496

Appendix A

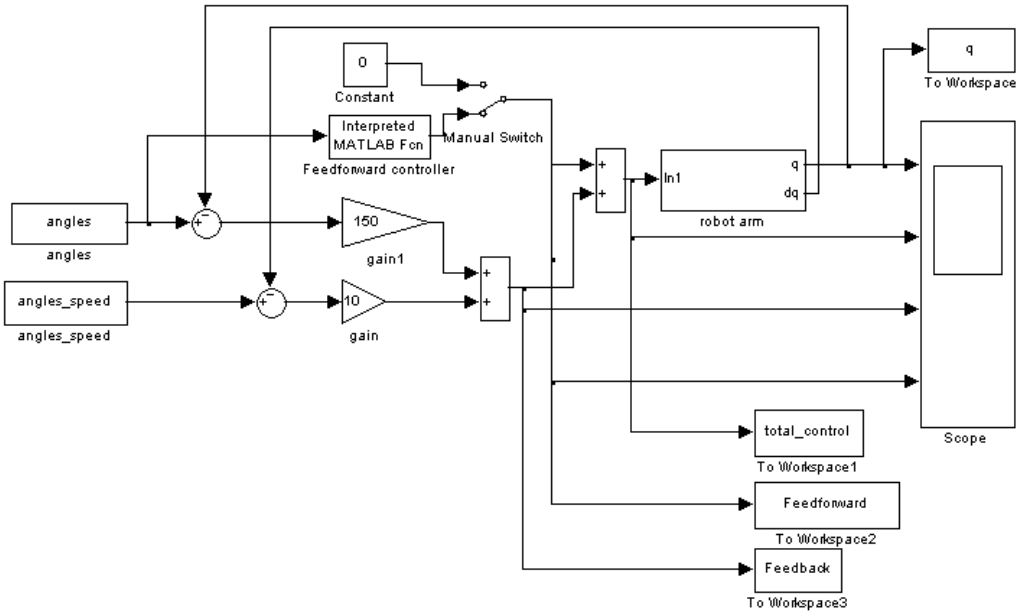
2 link robot simulation model

In figure A.1 the simulation model of the 2 link robot, used in for the simulations in Matlab, SimMechanics, is shown. In figure A.1(a) the main simulation model is shown. In this model a feedback controller is implemented to keep the robot in a desired position. Also the feedforward algorithm is implemented. In figure A.1(b) the subblock robot arm is shown. This is the part of the simulation model which contains the SimMechanics parts. The 2 link robot arm is build up with two links. The first link is connected to the base via joint 1 which can rotating around the z-axis. The second link is connected to the first link via a joint which can rotate around the y-axis. At the end of the second link a force is applied using a link actuator. Both joints are actuated using a joint actuator and their positions and velocities are measured using a joint sensor.

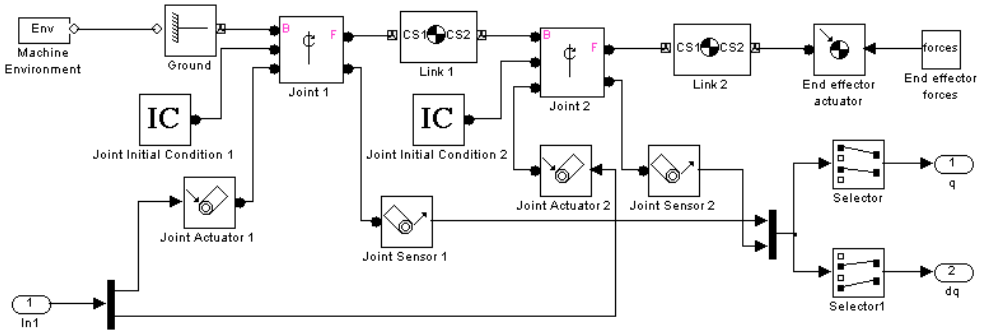
The algorithm for the feedforward controller is shown in A.1. The manipulator Jacobian J is constructed parametrical on forehand using (2.1) to (2.3). The input for the feedforward controller are the reference angles for the joints in degrees and they are converted to radians for the calculation of the required joint torques. The values for the dimensions and position of the robot arm are inserted in J and with the known end effector forces f the required joint torques τ are calculated.

Table A.1: Matlab code for the feedforward controller for the 2 link robot

```
function [tau] = calc_torques(angles)
global l1 l2 forces     % global parameters: -l1 being the length of link 1
                        %                    -l2 begin the length of link 2
                        %                    -forces being the forces on
                        %                         the tip of the robot

th1=angles(1)*2*pi/360; % desired angle for joint 1
th2=angles(2)*2*pi/360; % desired angle for joint 2

J=[ -sin(th1)*(l1 + l2*cos(th2)), -l2*cos(th1)*sin(th2);    % manipulator Jacobian
     cos(th1)*(l1 + l2*cos(th2)), -l2*sin(th1)*sin(th2);
                                0,          -l2*cos(th2);
                                0,             -sin(th1);
                                0,              cos(th1);
                                1,                    0];

f=[forces;0;0;0];   % force vector needed for calculating the joint torques

tau=-J'*f;      % Joint torque calculation
end
```



(a) Main simulink model



(b) Subblock robot arm

Figure A.1: Simulation model for the simulations of th2 2 link robot arm

Appendix B

Implementation of the gravity compensation for simulations in 3D

In this appendix the simulation model and the different m-files are presented which are used to calculate the desired joint torques for the 3D case. Figure B.1 is the main simulation file. Sub-block *TULIPV3* is shown in Figure B.2 and in this sub-block TUlip is simulated with SimMechanics. Figure B.3 shows the simulation model of the left leg which consists of six links and six joints.

In Table B.1 and B.2 the main m-file is shown, which is implemented in the Matlab, SimMechanics model from Figure B.1. The input of this function-file is the joint angles and the output is the calculated joint torques for gravity compensation. The m-files in Tables B.3 to B.10 are called in the main file. These m-files calculate sub-parts of the algorithm stated in Section 3.1.1. The m-file in Table B.3 calculates the vertical distance between the two feet to determine whether the robot is in single or double support. The m-file in Tables B.4 to B.6 calculates the distribution factor α and position of CoM' with respect to the right foot. The m-file in Table B.7 calculates the gravity compensation torques for a leg when it is a swing leg. The m-file from Tables B.8 and B.9 calculates the gravity compensation torques when a leg is a support leg. Finally the m-file from Table B.10 calculates the numerical Jacobian. In this m-file the parametric jacobian is left out because this is to big to include in the report.

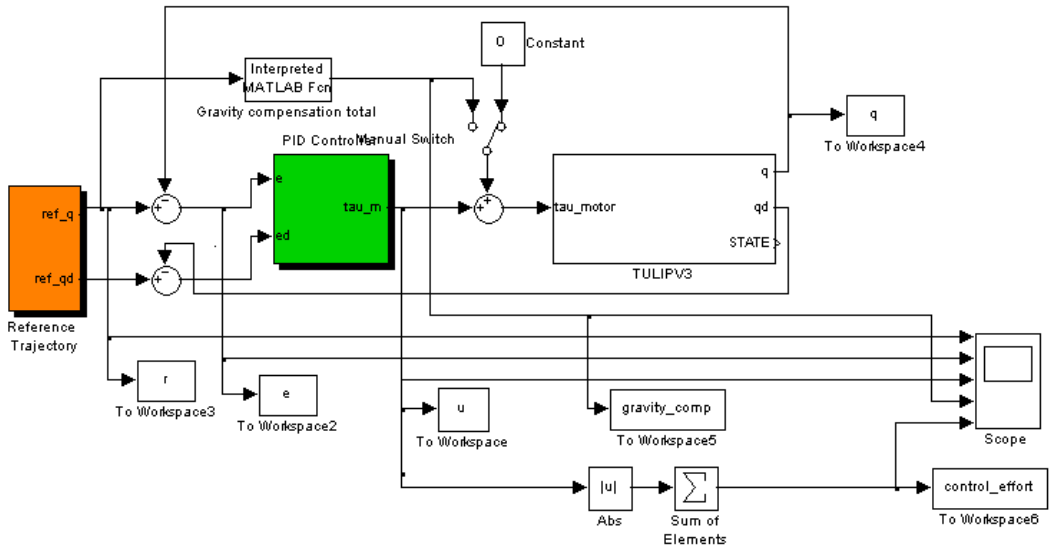


Figure B.1: The main simulation model

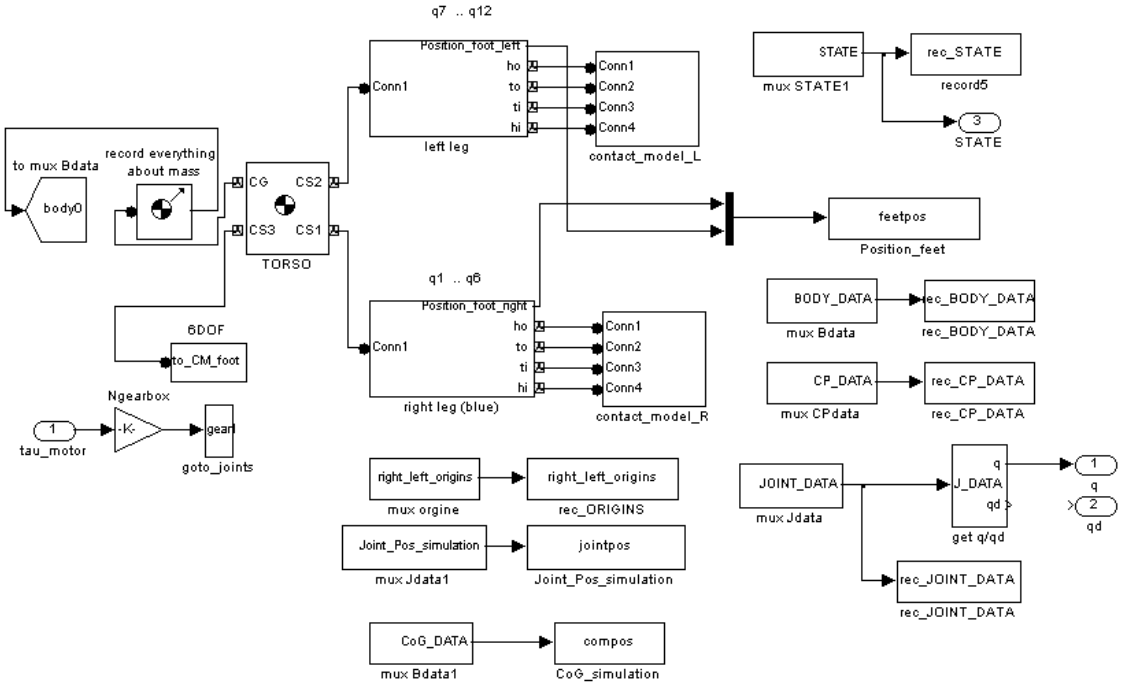


Figure B.2: The sub-simulation model *TULIPV3*, the part which simulates the robot

Table B.1: Main matlab file part 1

```
function [torques]=calc_torques_3D(angles)
%% values
%%% robot parameters %%%
par.a1=150e-3/2;  par.d1=0;       par.lc1=par.a1/2;   par.m1=14.3857;
par.a2=0;         par.d2=182e-3;  par.lc2=par.d2/2;   par.m2=1.0278;
par.a3=0;         par.d3=0;       par.lc3_x=0;        par.m3=0;
                                  par.lc3_z=0;
par.a4=320e-3;    par.d4=0;       par.lc4=par.a4/2;   par.m4=3.5839;
par.a5=276e-3;                    par.lc5=par.a5/2;   par.m5=0.5273;
par.a6=13e-3;     par.d6=0;       par.lc6=par.a6/2;   par.m6=0.9190;
par.a7=0;         par.d7=0;       par.lc7_x=par.a7/2; par.m7=0;
                                  par.lc7_y=par.d7/2;
par.a8=0;         par.d8=40e-3;   par.lc8=par.d8/2;   par.m8=0;

par.foot_t  = 175.5e-3;   %dx_ankle_to_inside_too
par.foot_h  = 59.5e-3;    %dx_ankle_to_inside_heel
par.foot_i  = 62.5e-3;    %dx_ankle_to_inside_foot
par.foot_o  = 87.5e-3;    %dx_ankle_to_outside_foot

par.M=26.5015;

%%% rest %%%
par.g=9.81;

%% angles
[angles_right, angles_left]=ref_angles;

%% d_height, CoM, alpha
d_height=calc_d_height(angles_right, angles_left, par);
[CoM, alpha, d_feet, pos_right, pos_left]=calc_alpha_CoM(angles_right,...
                                          angles_left, par);
```

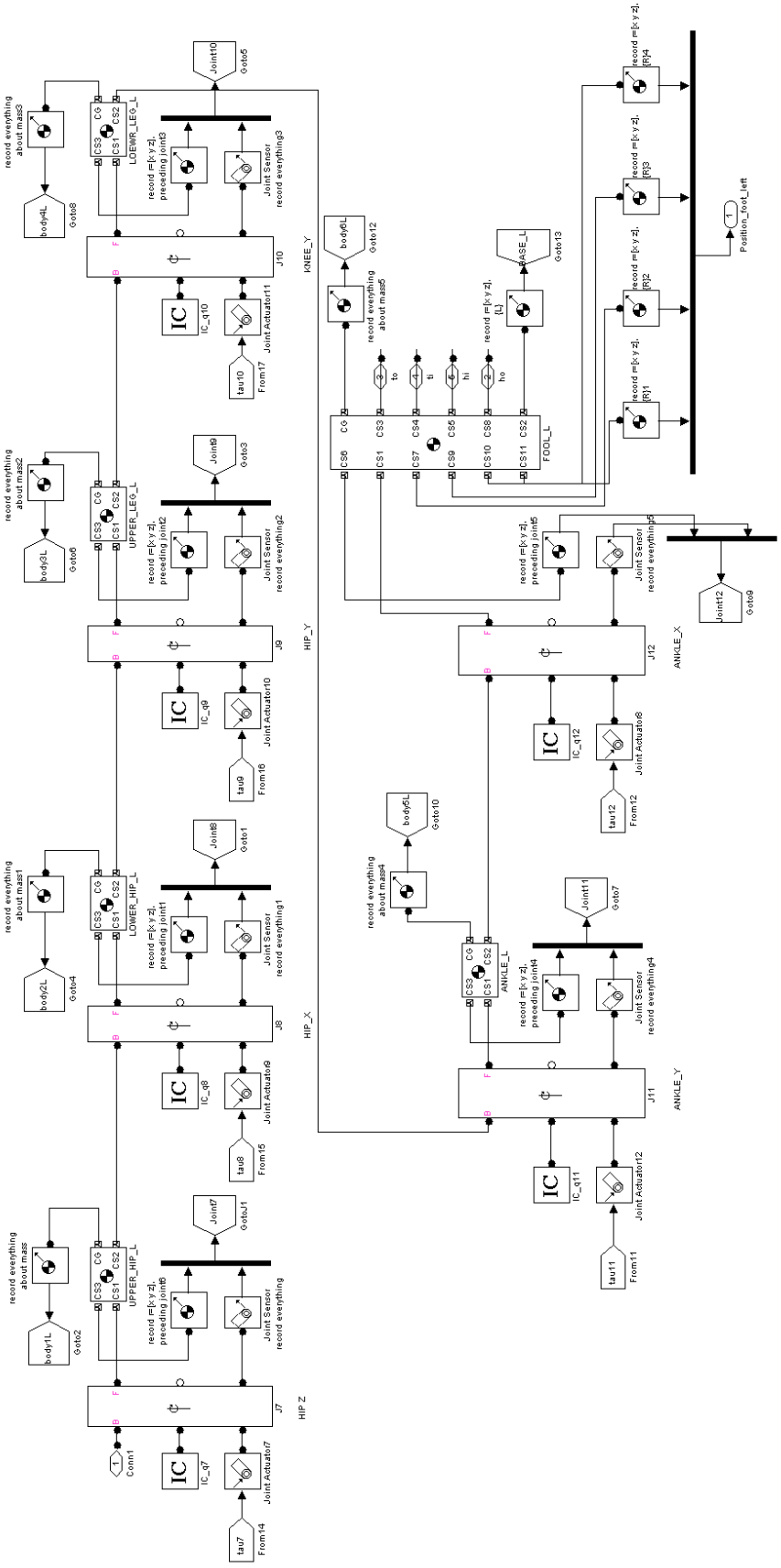


Figure B.3: The simulation model of the left leg

Table B.2: Main matlab file part 2

```
%% calc torques

if alpha == 0   %right is stance left is swing
    index=1;
    foot =1;    %right foot is stance

    %%% alpha and point of groundforce %%%
    alpha_right=1;

    %%%%%%%%%%%%%%%%%%
    %%% calc torques %%%
    calc_torques=calc_torques_swing(angles,index)/1e2;

    %%% torques right %%%
    torques_right=-calc_torques_stance(angles_right, alpha_right, par, CoM,...
                  foot, pos_right, pos_left)/1e3;

    %%% torques left %%%
    torques_left=calc_torques(7:12);

elseif alpha == 1         %right is swing left is stance
    index=2;
    foot =2;    %left foot is stance

    %%% alpha and point of groundforce %%%
    alpha_left=1;

    %%%%%%%%%%%%%%%%%%
    %%% calc torques %%%
    calc_torques=calc_torques_swing(angles,index)/1e2;

    %%% torques right %%%
    torques_right=calc_torques(1:6);

    %%% torques left %%%
    if d_height<1e-3
        par.a1=-par.a1;  %this is the only value that differce from the left and
                         %right leg and so the same functions for the calcualation
                         %of the torques can be used
        torques_left=-calc_torques_stance(angles_left, alpha_left, par, CoM,...
                     foot, pos_right, pos_left)/1e3;
    else
        torques_left=calc_torques(7:12);
    end

else                     %%dubbel support
    index=3;
    alpha_right=1-alpha;
    alpha_left=alpha;

    %%%%%%%%%%%%%%%%%%
    %%% calc torques %%%
    %%% right %%%
    foot=1;   %right foot is stance
    torques_right=-calc_torques_stance(angles_right, alpha_right, par, CoM,...
                  foot, pos_right, pos_left)/1e3;

    %%% left %%%
    par.a1=-par.a1;  %this is the only value that differce from the left and
                     %right leg and so the same functions for the calcualation
                     %of the torques can be used
    foot=2;   %left foot is stance
    torques_left=-calc_torques_stance(angles_left, alpha_left, par, CoM,...
                 foot, pos_right, pos_left)/1e3;
end

%% output
torques=[torques_right ;torques_left];
```

Table B.3: Matlab function for calculating the distance in height between the two feet

```
function [d_height]=calc_d_height(angles_right, angles_left, par)
%% values
%%% robot parameters %%%
a1=par.a1;
a2=par.a2;   d2=par.d2;
a3=par.a3;
a4=par.a4;   d4=par.d4;
a5=par.a5;
a6=par.a6;   d6=par.d6;
a7=par.a7;   d7=par.d7;
a8=par.a8;   d8=par.d8;

%%% rest %%%
g=9.81;

%% angles
%%% right %%%               %%% left %%%
th1_r=angles_right(1);      th1_l=angles_left(1);
th2_r=angles_right(2);      th2_l=angles_left(2);
th3_r=angles_right(3);      th3_l=angles_left(3);
th4_r=angles_right(4);      th4_l=angles_left(4);
th5_r=angles_right(5);      th5_l=angles_left(5);
th6_r=angles_right(6);      th6_l=angles_left(6);

%% difference height
%%% right %%%
height1_r=sin(th6_r+th2_r)*a1;
height2_r=cos(th6_r+th2_r)*cos(th5_r+th4_r+th3_r)*d2;
height4_r=cos(th6_r)*cos(th5_r+th4_r)*a4;
height5_r=cos(th6_r)*cos(th5_r)*a5;
height6_r=cos(th6_r)*a6;
height7_r=a7;
height8_r=d8;

height_r=height1_r+height2_r+height4_r+height5_r+height6_r+height7_r+height8_r;

%%% left %%%
height1_l=sin(th6_l+th2_l)*a1;
height2_l=cos(th6_l+th2_l)*cos(th5_l+th4_l+th3_l)*d2;
height4_l=cos(th6_l)*cos(th5_l+th4_l)*a4;
height5_l=cos(th6_l)*cos(th5_l)*a5;
height6_l=cos(th6_l)*a6;
height7_l=a7;
height8_l=d8;

height_l=-height1_l+height2_l+height4_l+height5_l+height6_l+height7_l+height8_l;

%%% difference %%%
d_height=height_r-height_l;
```

Table B.4: Matlab function for calculating the factor α and CoM' part 1

```
function [CoM, alpha,d_feet, pos_right, pos_left]=calc_alpha_CoM(angles_right,...
                                                    angles_left, par)
%% values
%%% robot parameters %%%
a1=par.a1;   d1=par.d1;     lc1=par.lc1;        m1=par.m1;
a2=par.a2;   d2=par.d2;     lc2=par.lc2;        m2=par.m2;
a3=par.a3;   d3=par.d3;     lc3_x=par.lc3_x;    m3=par.m3;
                            lc3_z=par.lc3_z;
a4=par.a4;   d4=par.d4;     lc4=par.lc4;        m4=par.m4;
a5=par.a5;                  lc5=par.lc5;        m5=par.m5;
a6=par.a6;   d6=par.d6;     lc6=par.lc6;        m6=par.m6;
a7=par.a7;   d7=par.d7;     lc7_x=par.lc7_x;    m7=par.m7;
                            lc7_y=par.lc7_y;
a8=par.a8;   d8=par.d8;     lc8=par.lc8;        m8=par.m8;

foot_t=par.foot_t;    %dx_ankle_to_inside_too
foot_h=par.foot_h;    %dx_ankle_to_inside_heel
foot_i=par.foot_i;    %dx_ankle_to_inside_foot
foot_o=par.foot_o;    %dx_ankle_to_outside_foot

%%% rest %%%
g=par.g;
M=par.M;

%% angles
%%% right %%%       %%% left %%%
th1_r=angles_right(1);    th1_l=angles_left(1);
th2_r=angles_right(2);    th2_l=angles_left(2);
th3_r=angles_right(3);    th3_l=angles_left(3);
th4_r=angles_right(4);    th4_l=angles_left(4);
th5_r=angles_right(5);    th5_l=angles_left(5);
th6_r=angles_right(6);    th6_l=angles_left(6);

%% r_vectors of CoM of links analicitaly, origin in middle body
%%% right %%%
th1=th1_r; th2=th2_r; th3=th3_r; th4=th4_r; th5=th5_r; th6=th6_r;
link2_r=[a1;
         0;
         lc2];
link3_r=[a1;
         0;
         -(- d2 - lc3_z*cos(th2))];
link4_r=[a1 - (-lc4*sin(th1)*sin(th3) + lc4*cos(th1)*cos(th3)*sin(th2));
         -(lc4*cos(th1)*sin(th3) + lc4*cos(th3)*sin(th1)*sin(th2));
         -(- d2 - a3*cos(th2) - d4*sin(th2) - lc4*cos(th2)*cos(th3))];
link5_r=[a1 - (-lc5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         - lc5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))...
         -d4*cos(th1)*cos(th2) - a4*sin(th1)*sin(th3) + a4*cos(th1)*cos(th3)*sin(th2));
         -(lc5*cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
         + lc5*sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
         + a4*cos(th1)*sin(th3) + a4*cos(th3)*sin(th1)*sin(th2));
         -(lc5*cos(th2)*sin(th3)*sin(th4) - a3*cos(th2) - d4*sin(th2)...
         - a4*cos(th2)*cos(th3) - lc5*cos(th2)*cos(th3)*cos(th4) - d2)];
link6_r=[a1 - (-a5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
        - a5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))...
        - lc6*cos(th5)*(cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
        + sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3)))...
        - a4*sin(th1)*sin(th3) - lc6*sin(th5)*(cos(th4)*(cos(th3)*sin(th1)...
        + cos(th1)*sin(th2)*sin(th3)) - sin(th4)*(sin(th1)*sin(th3)...
        - cos(th1)*cos(th3)*sin(th2))) + a4*cos(th1)*cos(th3)*sin(th2));
        -(a5*cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
        + a5*sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
        + a4*cos(th1)*sin(th3) + lc6*cos(th5)*(cos(th4)*(cos(th1)*sin(th3...
        + cos(th3)*sin(th1)*sin(th2)) + sin(th4)*(cos(th1)*cos(th3...
```

Table B.5: Matlab function for calculating the factor α and CoM' part 2

```
        - sin(th1)*sin(th2)*sin(th3))) + lc6*sin(th5)*(cos(th4)*(cos(th1)*cos(th3)...
        - sin(th1)*sin(th2)*sin(th3)) - sin(th4)*(cos(th1)*sin(th3)...
        + cos(th3)*sin(th1)*sin(th2))) + a4*cos(th3)*sin(th1)*sin(th2));
        -(lc6*sin(th3 + th4)*cos(th2)*sin(th5) - a3*cos(th2) - d4*sin(th2)...
        - a4*cos(th2)*cos(th3) - lc6*cos(th3 + th4)*cos(th2)*cos(th5)...
        - d2 - a5*cos(th2)*cos(th3)*cos(th4) + a5*cos(th2)*sin(th3)*sin(th4))];
link7_r=[a1 - (-a5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         - a5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))..
         - lc6*cos(th5)*(cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         + sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3)))...
         - a4*sin(th1)*sin(th3) - lc6*sin(th5)*(cos(th4)*(cos(th3)*sin(th1)...
         + cos(th1)*sin(th2)*sin(th3)) - sin(th4)*(sin(th1)*sin(th3)...
         - cos(th1)*cos(th3)*sin(th2))) + a4*cos(th1)*cos(th3)*sin(th2));
         -(a5*cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
         + a5*sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
         + a4*cos(th1)*sin(th3) + lc6*cos(th5)*(cos(th4)*(cos(th1)*sin(th3...
         + cos(th3)*sin(th1)*sin(th2)) + sin(th4)*(cos(th1)*cos(th3)...
         - sin(th1)*sin(th2)*sin(th3))) + lc6*sin(th5)*(cos(th4)*(cos(th1)*cos(th3)...
         - sin(th1)*sin(th2)*sin(th3)) - sin(th4)*(cos(th1)*sin(th3)...
         + cos(th3)*sin(th1)*sin(th2))) + a4*cos(th3)*sin(th1)*sin(th2));
         -(lc7_x *cos(th6)*sin(th2) - a3*cos(th2) - d4*sin(th2)...
         - a4*cos(th2)*cos(th3) - d2 - lc7_y*sin(th3 + th4 + th5)*cos(th2)...
         - a6*cos(th3 + th4)*cos(th2)*cos(th5) + a6*sin(th3 + th4)*cos(th2)*sin(th5)...
         - a5*cos(th2)*cos(th3)*cos(th4) + a5*cos(th2)*sin(th3)*sin(th4)...
         + lc7_x*cos(th3 + th4 + th5)*cos(th2)*sin(th6))];

r_right=[link2_r link3_r link4_r link5_r link6_r link7_r];

%%% left %%%
th1=th1_l; th2=th2_l; th3=th3_l; th4=th4_l; th5=th5_l; th6=th6_l;
a1=-a1;
link2_l=[a1;
         0;
         lc2];
link3_l=[a1;
         0;
         -(- d2 - lc3_z*cos(th2))];
link4_l=[a1 - (-lc4*sin(th1)*sin(th3) + lc4*cos(th1)*cos(th3)*sin(th2));
         -(lc4*cos(th1)*sin(th3) + lc4*cos(th3)*sin(th1)*sin(th2));
         -(- d2 - a3*cos(th2) - d4*sin(th2) - lc4*cos(th2)*cos(th3))];
link5_l=[a1 - (-lc5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         - lc5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))...
         - d4*cos(th1)*cos(th2) - a4*sin(th1)*sin(th3)...
         + a4*cos(th1)*cos(th3)*sin(th2));
         -(lc5*cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
         + lc5*sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
         + a4*cos(th1)*sin(th3) + a4*cos(th3)*sin(th1)*sin(th2));
         -(lc5*cos(th2)*sin(th3)*sin(th4) - a3*cos(th2) - d4*sin(th2)...
         - a4*cos(th2)*cos(th3) - lc5*cos(th2)*cos(th3)*cos(th4) - d2)];
link6_l=[a1 - (-a5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         - a5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))...
         - lc6*cos(th5)*(cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         + sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3)))...
         - a4*sin(th1)*sin(th3) - lc6*sin(th5)*(cos(th4)*(cos(th3)*sin(th1)...
         + cos(th1)*sin(th2)*sin(th3)) - sin(th4)*(sin(th1)*sin(th3)...
         - cos(th1)*cos(th3)*sin(th2))) + a4*cos(th1)*cos(th3)*sin(th2));
         -(a5*cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
         + a5*sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
         + a4*cos(th1)*sin(th3) + lc6*cos(th5)*(cos(th4)*(cos(th1)*sin(th3)...
         + cos(th3)*sin(th1)*sin(th2)) + sin(th4)*(cos(th1)*cos(th3)...
         - sin(th1)*sin(th2)*sin(th3))) + lc6*sin(th5)*(cos(th4)*(cos(th1)*cos(th3)...
         - sin(th1)*sin(th2)*sin(th3)) - sin(th4)*(cos(th1)*sin(th3)...
         + cos(th3)*sin(th1)*sin(th2))) + a4*cos(th3)*sin(th1)*sin(th2));
         -(lc6*sin(th3 + th4)*cos(th2)*sin(th5) - a3*cos(th2) - d4*sin(th2)...
         - a4*cos(th2)*cos(th3) - lc6*cos(th3 + th4)*cos(th2)*cos(th5) - d2...
         - a5*cos(th2)*cos(th3)*cos(th4) + a5*cos(th2)*sin(th3)*sin(th4))];
```

Table B.6: Matlab function for calculating the factor α and CoM' part 3

```
link7_l=[a1 - (-a5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         - a5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))...
         - lc6*cos(th5)*(cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
         + sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3)))...
         - a4*sin(th1)*sin(th3) - lc6*sin(th5)*(cos(th4)*(cos(th3)*sin(th1)...
         + cos(th1)*sin(th2)*sin(th3)) - sin(th4)*(sin(th1)*sin(th3)...
         - cos(th1)*cos(th3)*sin(th2))) + a4*cos(th1)*cos(th3)*sin(th2));
         -(a5*cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
         + a5*sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
         + a4*cos(th1)*sin(th3) + lc6*cos(th5)*(cos(th4)*(cos(th1)*sin(th3)...
         + cos(th3)*sin(th1)*sin(th2)) + sin(th4)*(cos(th1)*cos(th3)...
         - sin(th1)*sin(th2)*sin(th3))) + lc6*sin(th5)*(cos(th4)*(cos(th1)*cos(th3)...
         - sin(th1)*sin(th2)*sin(th3)) - sin(th4)*(cos(th1)*sin(th3)...
         + cos(th3)*sin(th1)*sin(th2))) + a4*cos(th3)*sin(th1)*sin(th2));
         -(lc7_x *cos(th6)*sin(th2) - a3*cos(th2) - d4*sin(th2) - a4*cos(th2)*cos(th3)...
         - d2 - lc7_y*sin(th3 + th4 + th5)*cos(th2)...
         - a6*cos(th3 + th4)*cos(th2)*cos(th5) + a6*sin(th3 + th4)*cos(th2)*sin(th5)...
         - a5*cos(th2)*cos(th3)*cos(th4) + a5*cos(th2)*sin(th3)*sin(th4)...
         + lc7_x*cos(th3 + th4 + th5)*cos(th2)*sin(th6))];

r_left=[link2_l link3_l link4_l link5_l link6_l link7_l];

%% distance between ankles
%%% right %%%
x_right=link6_r(1);
y_right=link6_r(2);

%%% left %%%
x_left=link6_l(1);
y_left=link6_l(2);

%%% distances between ankles %%%
dx=x_right-x_left;
dy=y_right-y_left;
d_feet=sqrt(dx^2+dy^2);

%%% angles in feet %%%
angle_right=-atan(dy/dx);

%% calc CoM
%%% CoM %%%
masses=[m2 m3 m4 m5 m6 m7];
CoM=(masses*r_right'+masses*r_left')/M;

%%% CoM projected
pos_CoM=[CoM(1)-x_right CoM(2)-y_right];
length_CoM=sqrt(pos_CoM(1)^2+pos_CoM(2)^2);
angle_CoM=atan(pos_CoM(2)/pos_CoM(1));
CoM_projected=cos(angle_right+angle_CoM)*length_CoM; %length of vector from right
                                                     %ankle to CoM projected on line
                                                     %from right to left ankle

%% calc alpha
%%% alpha %%%
alpha=CoM_projected/d_feet;

%% position of the feet
pos_right=link7_r;
pos_left=link7_l;
```

Table B.7: Matlab function for calculating the required torques for the swing leg

```
function [G]=calc_torks_swing(angles,index)
param

c0y=par.c0y;
c0z=par.c0z;
c10z=par.c10z;
c11z=par.c11z;
c12x=par.c12x;
c12z=par.c12z;
c1z=par.c1z;
c2y=par.c2y;
c2z=par.c2z;
c3z=par.c3z;
c4z=par.c4z;
c5z=par.c5z;
c7z=par.c7z;
c8y=par.c8y;
c8z=par.c8z;
c9z=par.c9z;
g=par.g;
l0y=par.l0y;
l10z=par.l10z;
l11z=par.l11z;
l1z=par.l1z;
l2y=par.l2y;
l2z=par.l2z;
l3z=par.l3z;
l4z=par.l4z;
l5z=par.l5z;
l7z=par.l7z;
l8y=par.l8y;
l8z=par.l8z;
l9z=par.l9z;
m0=par.m0;
m1=par.m1;
m2=par.m2;
m3=par.m3;
m4=par.m4;
m5=par.m5;
m6=par.m6;

phi1=angles(1);
phi2=angles(2);
phi3=angles(3);
phi4=angles(4);
phi5=angles(5);
phi6=angles(6);
phi7=angles(7);
phi8=angles(8);
phi9=angles(9);
phi10=angles(10);
phi11=angles(11);
phi12=angles(12);

if index==1
    G=gravcompR(c0y,c0z,c10z,c11z,c12x,c12z,c1z,c2y,c2z,c3z,c4z,c5z,c7z,c8y,c8z,c9z,...
                g,l0y,l10z,l11z,l1z,l2y,l2z,l3z,l4z,l5z,l7z,l8y,l8z,l9z,m0,m1,m2,m3,...
                m4,m5,m6,phi1,phi2,phi3,phi4,phi5,phi6,phi7,phi8,phi9,phi10,phi11,phi12);
end

if index==2
    G=gravcompL(c0y,c0z,c10z,c11z,c12x,c12z,c1z,c2y,c2z,c3z,c4z,c5z,c7z,c8y,c8z,c9z,...
                g,l0y,l10z,l11z,l1z,l2y,l2z,l3z,l4z,l5z,l7z,l8y,l8z,l9z,m0,m1,m2,m3,...
                m4,m5,m6,phi1,phi2,phi3,phi4,phi5,phi6,phi7,phi8,phi9,phi10,phi11,phi12);
end
```

Table B.8: Matlab function for calculating the required torques for the stance leg part1

```
function [torques]=calc_torques_stance(angles, alpha, par, CoM, foot, pos_right, pos_left)
%% values
%%% robot parameters %%%
a1=par.a1;   d1=par.d1;
a2=par.a2;   d2=par.d2;
a3=par.a3;
a4=par.a4;   d4=par.d4;
a5=par.a5;
a6=par.a6;   d6=par.d6;
a7=par.a7;   d7=par.d7;
a8=par.a8;   d8=par.d8;

%%% rest %%%
g=par.g;
M=par.M;

%% angles
th1=angles(1);
th2=angles(2);
th3=angles(3);
th4=angles(4);
th5=angles(5);
th6=angles(6);

%% calc forces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% calc distances to hip from point where force acts %%%
x_hip=-(-d8*(cos(th6)*(cos(th5)*(cos(th4)*(sin(th1)*sin(th3)...
      - cos(th1)*cos(th3)*sin(th2)) + sin(th4)*(cos(th3)*sin(th1)...
      + cos(th1)*sin(th2)*sin(th3))) + sin(th5)*(cos(th3)*cos(th4)*sin(th1)...
      - sin(th1)*sin(th3)*sin(th4) + cos(th1)*cos(th3)*sin(th2)*sin(th4)...
      + cos(th1)*cos(th4)*sin(th2)*sin(th3))) - cos(th1)*cos(th2)*sin(th6))...
      - d7*(sin(th5)*(cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
      + sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3)))...
      - cos(th5)*(cos(th3)*cos(th4)*sin(th1) - sin(th1)*sin(th3)*sin(th4)...
      + cos(th1)*cos(th3)*sin(th2)*sin(th4) + cos(th1)*cos(th4)*sin(th2)*sin(th3)))...
      - a5*cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
      - a5*sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3))...
      - d4*cos(th1)*cos(th2) + a3*cos(th1)*sin(th2)...
      - a6*cos(th5)*(cos(th4)*(sin(th1)*sin(th3) - cos(th1)*cos(th3)*sin(th2))...
      + sin(th4)*(cos(th3)*sin(th1) + cos(th1)*sin(th2)*sin(th3)))...
      - a4*sin(th1)*sin(th3) + a7*sin(th6)*(cos(th5)*(cos(th4)*(sin(th1)*sin(th3)...
      - cos(th1)*cos(th3)*sin(th2)) + sin(th4)*(cos(th3)*sin(th1)...
      + cos(th1)*sin(th2)*sin(th3))) + sin(th5)*(cos(th3)*cos(th4)*sin(th1)...
      - sin(th1)*sin(th3)*sin(th4) + cos(th1)*cos(th3)*sin(th2)*sin(th4)...
      + cos(th1)*cos(th4)*sin(th2)*sin(th3))) - a6*sin(th5)*(cos(th3)*cos(th4)*sin(th1)...
      - sin(th1)*sin(th3)*sin(th4) + cos(th1)*cos(th3)*sin(th2)*sin(th4)...
      + cos(th1)*cos(th4)*sin(th2)*sin(th3)) + a7*cos(th1)*cos(th2)*cos(th6)...
      + a4*cos(th1)*cos(th3)*sin(th2));
y_hip=d8*(cos(th6)*(cos(th5)*(cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
      + sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3)))...
      + sin(th5)*(cos(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
      - sin(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))))...
      + cos(th2)*sin(th1)*sin(th6)) - d7*(cos(th5)*(cos(th4)*(cos(th1)*cos(th3)...
      - sin(th1)*sin(th2)*sin(th3)) - sin(th4)*(cos(th1)*sin(th3)...
      + cos(th3)*sin(th1)*sin(th2))) - sin(th5)*(cos(th4)*(cos(th1)*sin(th3)...
      + cos(th3)*sin(th1)*sin(th2)) + sin(th4)*(cos(th1)*cos(th3)...
      - sin(th1)*sin(th2)*sin(th3)))) + a5*cos(th4)*(cos(th1)*sin(th3)...
      + cos(th3)*sin(th1)*sin(th2)) + a5*sin(th4)*(cos(th1)*cos(th3)...
      - sin(th1)*sin(th2)*sin(th3)) - a7*sin(th6)*(cos(th5)*(cos(th4)*(cos(th1)...
      *sin(th3) + cos(th3)*sin(th1)*sin(th2)) + sin(th4)*(cos(th1)*cos(th3)...
      - sin(th1)*sin(th2)*sin(th3))) + sin(th5)*(cos(th4)*(cos(th1)*cos(th3)...
      - sin(th1)*sin(th2)*sin(th3)) - sin(th4)*(cos(th1)*sin(th3)...
      + cos(th3)*sin(th1)*sin(th2)))) + a4*cos(th1)*sin(th3)...
      + a6*cos(th5)*(cos(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2))...
      + sin(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3)))...
```

Table B.9: Matlab function for calculating the required torques for the stance leg part 2

```
      - d4*cos(th2)*sin(th1) + a3*sin(th1)*sin(th2)...
      + a6*sin(th5)*(cos(th4)*(cos(th1)*cos(th3) - sin(th1)*sin(th2)*sin(th3))...
      - sin(th4)*(cos(th1)*sin(th3) + cos(th3)*sin(th1)*sin(th2)))...
      + a7*cos(th2)*cos(th6)*sin(th1) + a4*cos(th3)*sin(th1)*sin(th2);
z_hip=-(d8*(sin(th2)*sin(th6) - cos(th3 + th4 + th5)*cos(th2)*cos(th6))...
    - a3*cos(th2) - d4*sin(th2) - a4*cos(th2)*cos(th3) + a7*cos(th6)*sin(th2)...
    - d7*sin(th3 + th4 + th5)*cos(th2) - a6*cos(th3 + th4)*cos(th2)*cos(th5)...
    + a6*sin(th3 + th4)*cos(th2)*sin(th5) - a5*cos(th2)*cos(th3)*cos(th4)...
    + a5*cos(th2)*sin(th3)*sin(th4) + a7*cos(th3 + th4 + th5)*cos(th2)*sin(th6));

x=x_hip;
y=y_hip;
z=z_hip;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% calc distances to CoM from point where force acts %%%
if foot==1;
    x_CoM=pos_right(1)-CoM(1);
    y_CoM=pos_right(2)-CoM(2);
    z_CoM=pos_right(3)-CoM(3);
end

if foot==2;
    x_CoM=pos_left(1)-CoM(1);
    y_CoM=pos_left(2)-CoM(2);
    z_CoM=pos_left(3)-CoM(3);
end

x=x_CoM;
y=y_CoM;
z=z_CoM;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%% calc the angle that the resultante force in the foot makes with the vertical %%%
xy=sqrt(x^2+y^2);
angle_F=atan(xy/z);

%%% calc forces in x-, y- and z-direction %%%
F=alpha*g*M;
F_w=-tan(angle_F)*F;
F_w_x=(x/xy)*F_w;
F_w_y=(y/xy)*F_w;

if x==0;
    F_w_x=0;
end
if y==0
    F_w_y=0;
end

%force vector
f=[F F_w_x F_w_y 0 0 0]'; %1st force is Z-direction, 2nd is X-direction,
                          %3th is y-direction.

%% calc torques
J=calc_J(angles,par);
torques=J'*f;
```

Table B.10: Matlab function for calculating the numerical Jacobian

```
function [J]=calc_J(angles,par)
%% values
%%% robot parameters %%%
a1=par.a1;   d1=par.d1;
a2=par.a2;   d2=par.d2;
a3=par.a3;
a4=par.a4;   d4=par.d4;
a5=par.a5;
a6=par.a6;   d6=par.d6;
a7=par.a7;   d7=par.d7;
a8=par.a8;   d8=par.d8;

%%% rest %%%
g=9.81;

%% angles
th1=angles(1);
th2=angles(2);
th3=angles(3);
th4=angles(4);
th5=angles(5);
th6=angles(6);

%% calc J
J=[...
   ...
   ...
   ...
   ...
   ...]
```

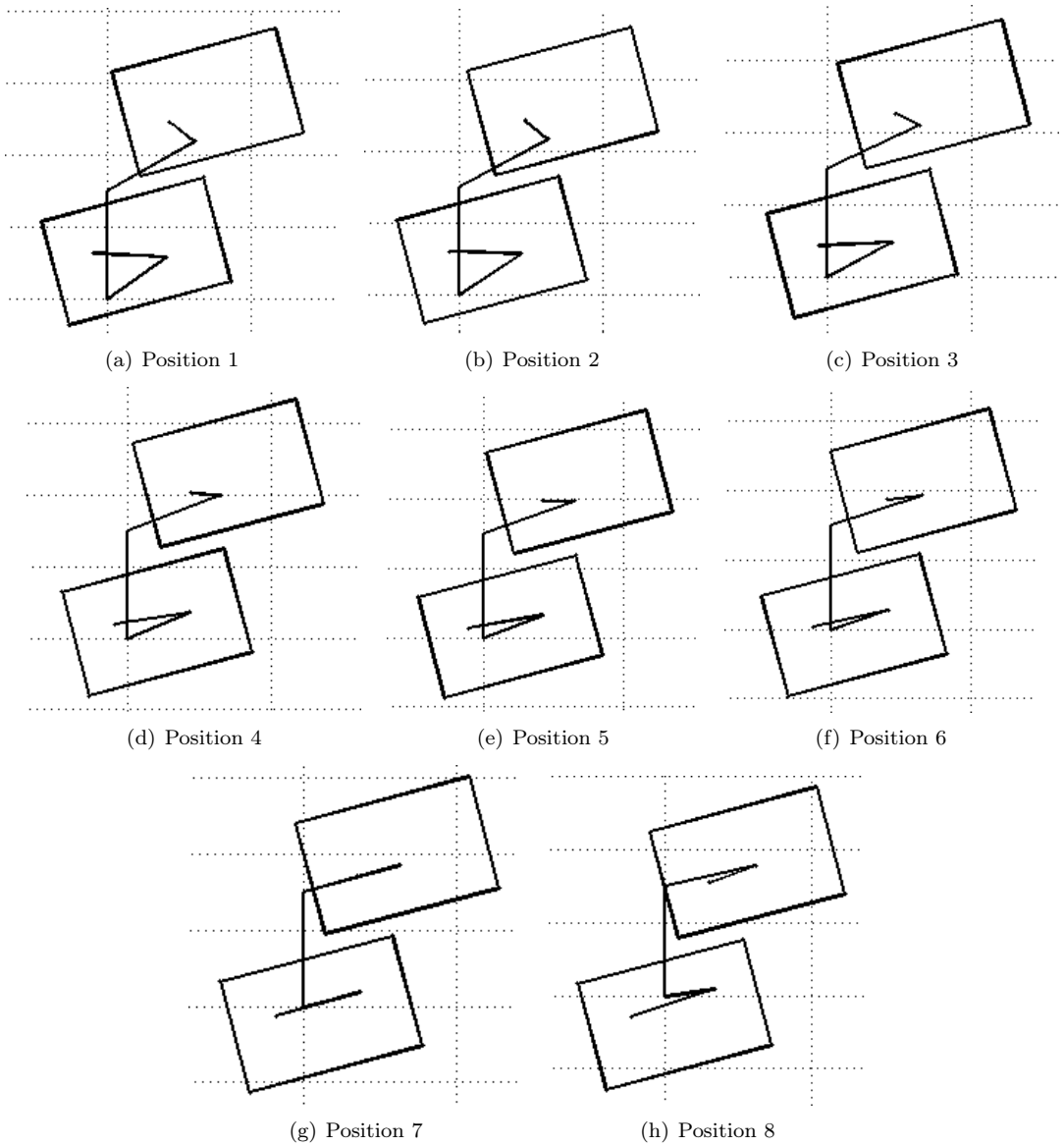
Appendix C

Positions of the static simulations

In this Appendix the joint angles for the static simulations are stated in table C.1. Also schematic representations of the different positions are shown in Figure C.1. The first position is at the beginnen of the double support phase of a stable walking gait and the eigth position is at the end of it.

Table C.1: Joint angles for the positions for the simulations in degrees

	Position							
Joint	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	15,00	15,00	15,00	15,00	15,00	15,00	15,00	14,99
2	6,55	6,09	4,44	2,39	1,88	1,03	-0,25	-1,53
3	-17,02	-17,58	-17,85	-16,73	-16,28	-15,40	-13,79	-12,10
4	37,57	37,93	38,84	39,29	39,27	39,11	38,52	37,86
5	-20,55	-20,35	-20,99	-22,56	-22,99	-23,70	-24,73	-25,79
6	-6,55	-6,09	-4,44	-2,39	-1,88	-1,03	0,25	1,53
7	15,00	15,00	15,00	15,00	15,00	15,00	15,00	14,99
8	7,05	6,60	4,94	2,90	2,39	1,54	0,26	-1,02
9	-24,91	-25,21	-25,75	-25,59	-25,42	-25,03	-24,19	-23,15
10	30,48	30,30	31,71	34,08	34,64	35,50	36,56	37,33
11	-5,58	-5,08	-5,96	-8,49	-9,23	-10,47	-12,37	-14,20
12	-7,06	-6,60	-4,94	-2,90	-2,39	-1,54	-0,26	1,02



(a) Position 1 (b) Position 2 (c) Position 3

(d) Position 4 (e) Position 5 (f) Position 6

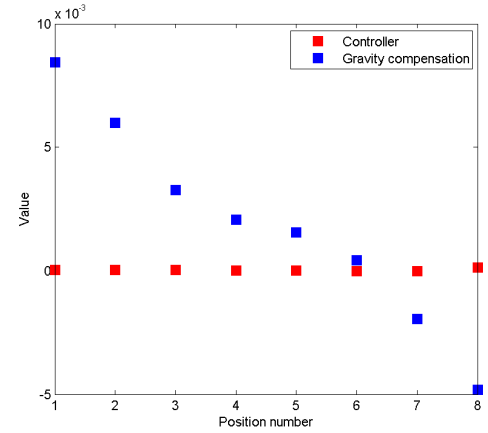
(g) Position 7 (h) Position 8

Figure C.1: Top view of the schematic representation the position used for the simulations

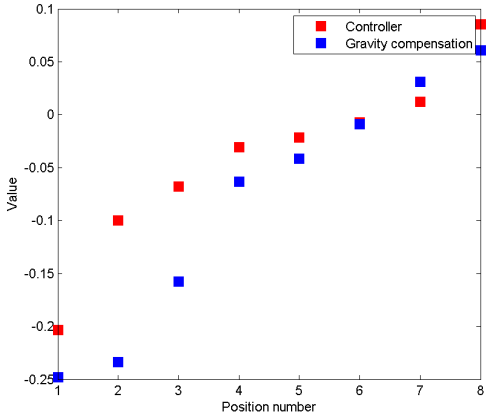
Appendix D

Simulation results from simulations with estimated ground contact forces

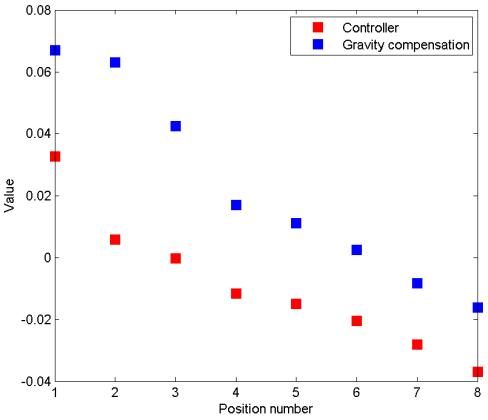
In this Appendix the results from the simulations with measured ground contact forces are shown graphically. In Figures D.1 and D.2 the values from the feedback controller (red) and the gravity compensation algorithm (blue) are plotted. If the gravity compensation algorithm would give satisfactory values the red and blue squares would lie on top of each other. This is clearly not the case so the obtained gravity compensation algorithm does not calculate satisfactory values.



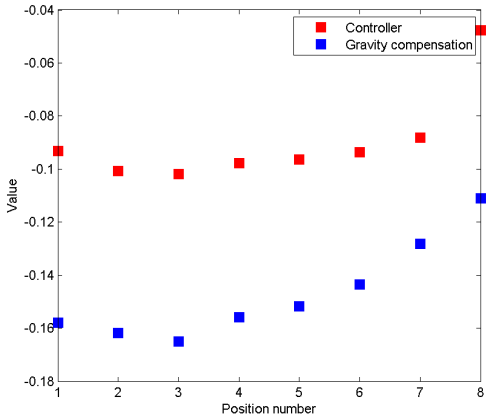
(a) Joint 1



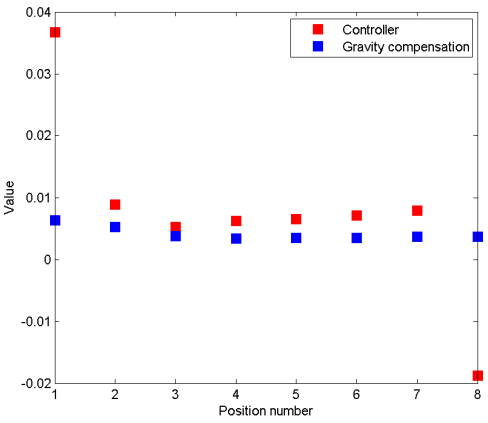
(b) Joint 2



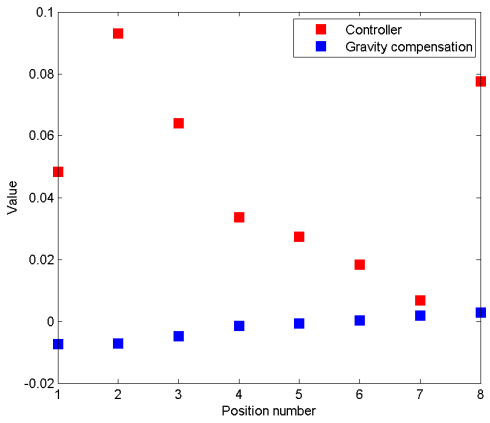
(c) Joint 3



(d) Joint 4

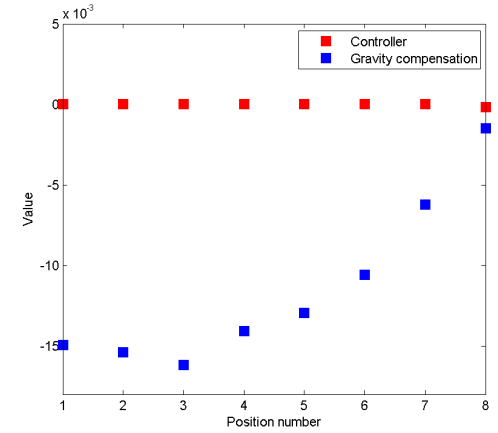


(e) Joint 5

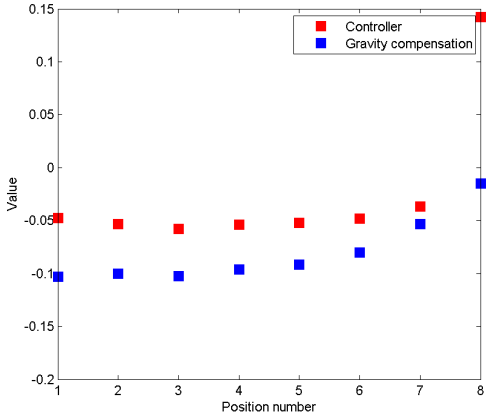


(f) Joint 6

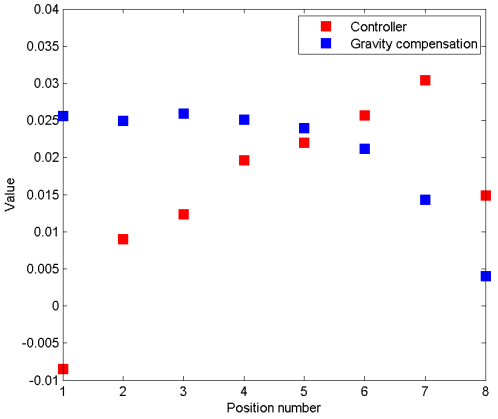
Figure D.1: Results from the simulations with estimated ground contact forces; graphical presentation of the controller values of the right leg. From hip to ankle: link 1 to 6



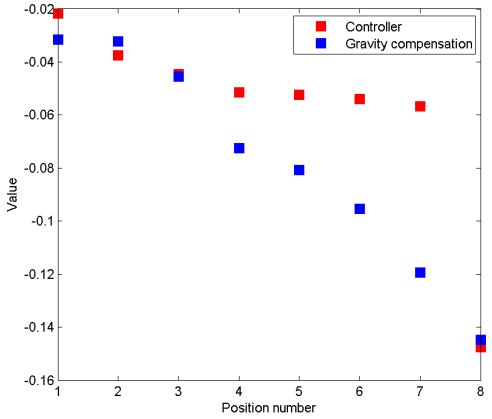
(a) Joint 7



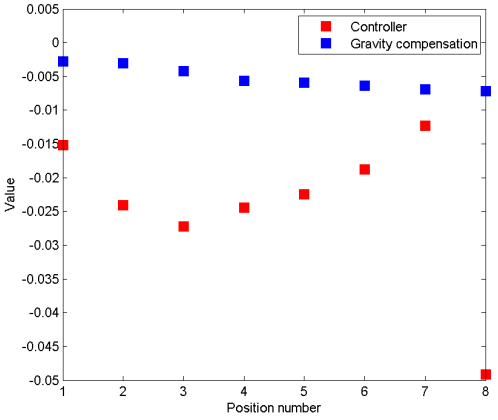
(b) Joint 8



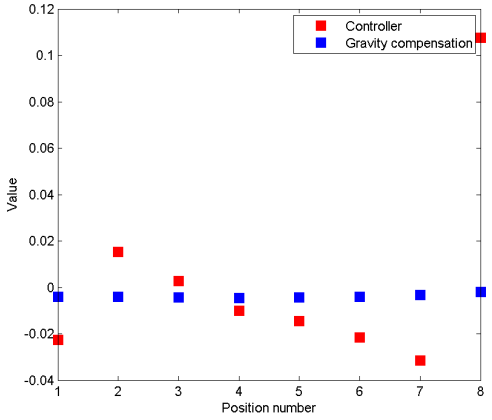
(c) Joint 9



(d) Joint 10



(e) Joint 11



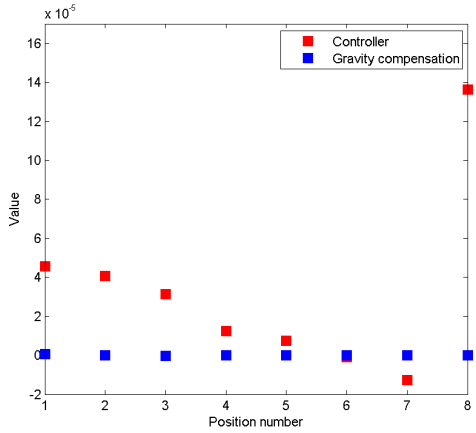
(f) Joint 12

Figure D.2: Results from the simulations with estimated ground contact forces; graphical presentation of the controller values of the left leg. From hip to ankle: link 7 to 12

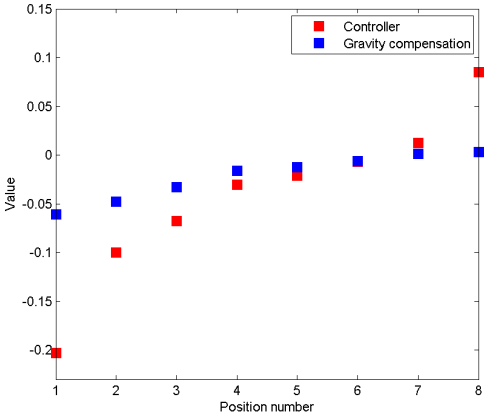
Appendix E

Simulation results from simulations with measured ground contact forces

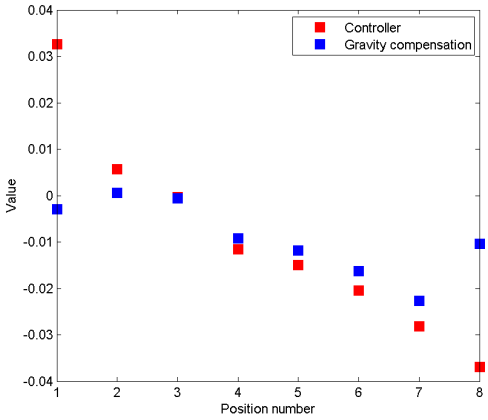
In this Appendix the results from the simulations with measured ground contact forces are shown graphically. In Figures E.1 and E.2 the values from the feedback controller (red) and the gravity compensation algorithm (blue) are plotted. If the gravity compensation algorithm would give satisfactory values the red and blue squares would lie on top of each other. This is clearly not the case so the obtained gravity compensation algorithm does not calculate satisfactory values.



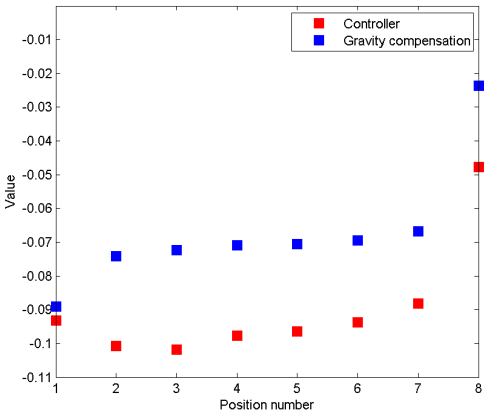
(a) Joint 1



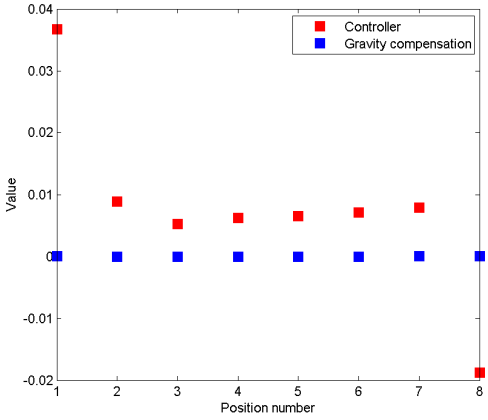
(b) Joint 2



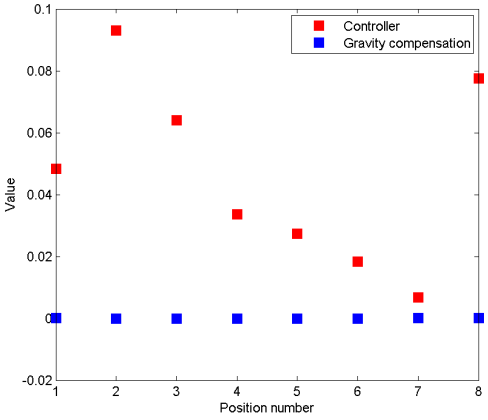
(c) Joint 3



(d) Joint 4

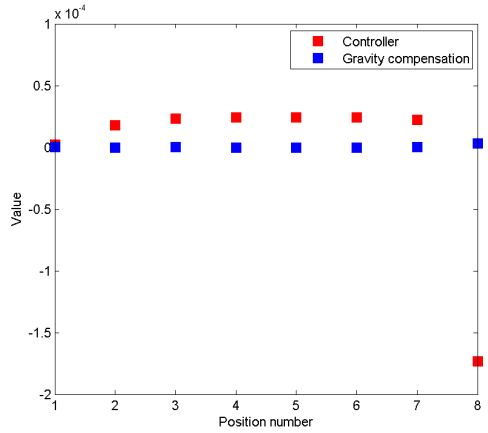


(e) Joint 5

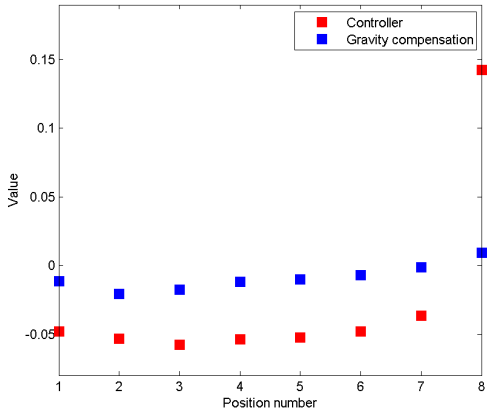


(f) Joint 6

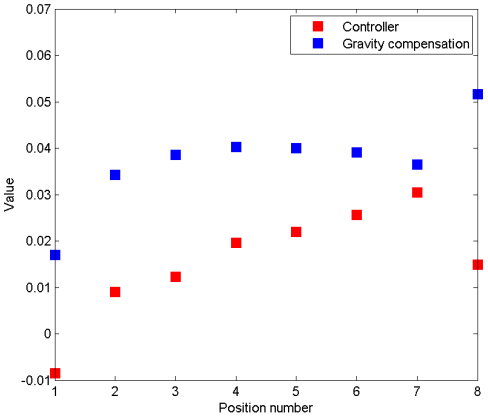
Figure E.1: Results from the simulations with measured ground contact forces; graphical presentation of the controller values of the right leg. From hip to ankle: link 1 to 6



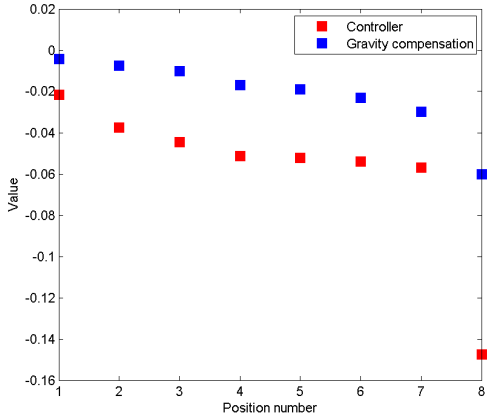
(a) Joint 7



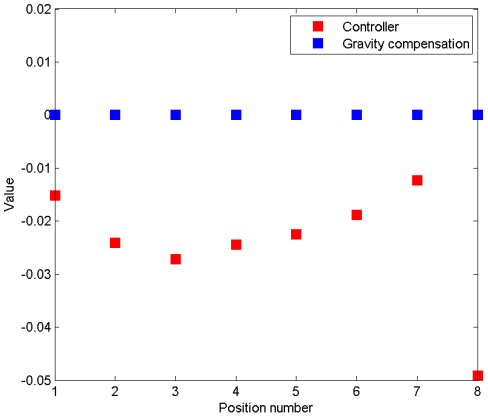
(b) Joint 8



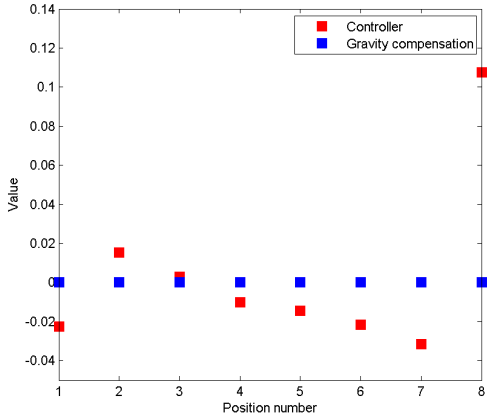
(c) Joint 9



(d) Joint 10



(e) Joint 11



(f) Joint 12

Figure E.2: Results from the simulations with measured ground contact forces; graphical presentation of the controller values of the right leg. From hip to ankle: link 1 to 6