Creating stable full body motion for a humanoid robot using the Centroidal Momentum Matrix

Erik de Vries

***Abstract*—In this paper we show a way to create stable full body motion for a humanoid robot without defining all joint trajectories in advance. The full body motion is split in a task and compensation motion. The task motion can be generated in advance, while the compensation motion is obtained during execution of the task. We show that the compensation motion can be obtained by making use of the linear relation between the joint velocities and the linear momentum of the complete robot. A setpoint for the linear momentum is generated by making use of the error between the current Capture Point and the desired Capture Point location. We have implemented the control algorithm in simulation as well on a real humanoid robot. We were able to create stable full body motion for a bending forward task. It turns out that the control algorithm to create stable full body motion is insensitive for model errors in the internal model of the robot.**

I. Introduction

Humans have the ability to maintain balanced, while performing all kinds of task. However creating stabilising control laws for humanoid robots which performs a task is still a challenging job. But to be able to create a versatile humanoid that can function in a human environment. The humanoid should be able to remain balanced while performing all kinds of tasks.

Why is creating stable full body motion for a humanoid robot so difficult? When creating stable full body motion for a humanoid robot we are facing three major control issues. Firstly a humanoid robot is a highly unstable system, it has a high center of mass (CoM) and a small support area. Secondly a humanoid robot is an underactuated system, which means the number of actuators is less then the number of degrees of freedom. In a humanoid robot there is no actuator between the foot and the floor, which creates an unilateral constraint on the foot. For this reason the actuators in the robot cannot produce an arbitrary amount of torque and also the reaction force acting on the foot is constraint. Thirdly a humanoid robot is a complex system. A typical humanoid robot has 14 degrees of freedom (dof), which results in a redundant system.

Due to these control issues, online full body motion generating is challenging. That is the reason why most full body motion are generated offline, for example by recording motions generated by humans [1], [2], or by optimisation. The offline generated joint trajectories are played back on the robot. The stiff position controllers will try to follow the pre-generated joint trajectories as close as possible. This method works if we assume small uncertainties and disturbances. Furthermore, this method requires the design of joint trajectories for many different tasks, which is a time consuming job. And it is impossible to design the joint trajectories for all the situations a humanoid robot will encounter in the real world.

Department of Mechanical Engineering
Delft University of Technology Delft, The Netherlands
Email: e.devries-wb@student.tudelft.nl

But is there an other way to generate full body motions? Khatib et al. [3] mentioned that a full body motion can be split into two parts, due to redundancy. The first part of the motion is the task motion, this motion fullfills the commanded task. The second part is the compensation motion, which compensates the task motion without changing the task motion to avoid a fall. By splitting the full body motion in a task and compensation motion, the task motion will become independent from changes in the environment and initial conditions. Which means changes in the environment will not change the task motion, but will affect the compensation motion. Imagine for example bending forward while standing on one or two leg(s). The task motion stays the same, namely bending the hip of the stance leg(s), but the compensation motion will be totally different, respectively swinging the free leg backwards and bending the knees. This makes it possible to create the task motion offline. The compensation motion should be generated online while the task is executed.

Biomechanical research shows that humans try to regulate the momentum around their Center of Mass (CoM) [4], [5]. When a human is performing a task motion, the body segments used during this motion will disturb the linear and angular momentum around the CoM. But by moving the free body segments (body segments which are not used during the task motion) the human is still able to control the momentum around the CoM. This mechanism can also be used to create a stable full body motion for humanoid robots.

The main goal of this paper is to create stable full body motion, for a humanoid robot, without defining all joint trajectories in advance. Like [6], we will use the linear relation between the joint velocity and the momentum of the humanoid robot, like derived in [7]. However we will extend this approach by using the task motion as input and creating the reference momentum setpoint by using a capture point [8] controller, instead of using a pre-planned CoM trajectory. We show how joint limits can be implemented in this famework and how we can implement this on a real robot. We show that the framework is not sensitive for model errors in the internal model of the robot, which is used to derive the relation between the momentum and joint velocities. This is a nice property because it is often difficult and time consuming to determine an exact internal model of a humanoid robot.

The remaining of this paper will be organised in the following way. In section II we focus on the theoretical

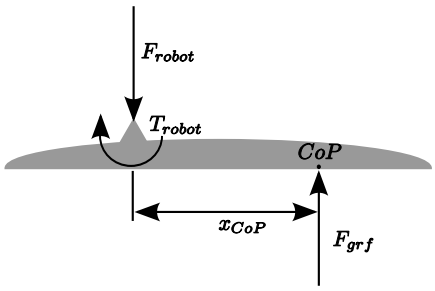


Fig. 1. A free body diagram of a typical robot foot

background. Section III shows how this framework was implemented in our soccer robot TUlip. In section IV the results of the simulation and the experiments on the real robot are shown. We end with the conclusion and future-work in section V

II. Generating full body motions

In this section we will focus on how to create a stable full body motion. We first show that the velocity of the CoM can be used to determine if the full body motion is stable. After that we show how we can use a momentum controller to obtain a stable fullbody motion.

A. Balance

Before we can create a compensation motion which stabilises the task motion. We should first focus on how we can determine if the resulting full body motion is stable. The fullbody motion will be stable if the robot does not fall. But how can we determine if the robot does not fall while perfroming a full body motion? The best known way to determine if the robot will not fall, is to determine if the robot is fully controlable. If the robot is fully controlable he will be able to avoid a fall. Due to the unilateral constraints on the foot, the robot is not directly connected to the ground, and can become underactuated, which increases the risk of falling. The robot will become underactuated if the foot starts to rotate. The foot will start to rotate if the forces and moments acting on the foot are not in equilibrium. Figure 1 shows a free body diagram of a weightless robot foot. The robot exerts a force F_{robot} and a torque T_{robot} at the ankle. If there is ground contact, the ground will produce a reaction force at each contact point of the foot with the ground. These forces can be represented by one force, the ground reference force (F_{grf}). The place where this ground reference force acts is called the center of pressure (CoP). The distance of the CoP to the ankle is x_{CoP}. The foot is in static equilibrium if $T_{robot} = F_{grf} \cdot x_{CoP}$ while $x_{CoP}^{min} < x_{CoP} < x_{CoP}^{max}$, where x_{CoP}^{min} and x_{CoP}^{max} represents the bounderies of the foot. Altough this is the best known way to determin the stability of a humanoid robot, it is not a sufficient condition to determine if a robot will fall or not. A robot can still fall while keeping the CoP inside the foot support polygon. Imagine a motion with zero ankle torque, this will put the CoP at the ankle ($x_{CoP} = 0$) even if the robot falls.

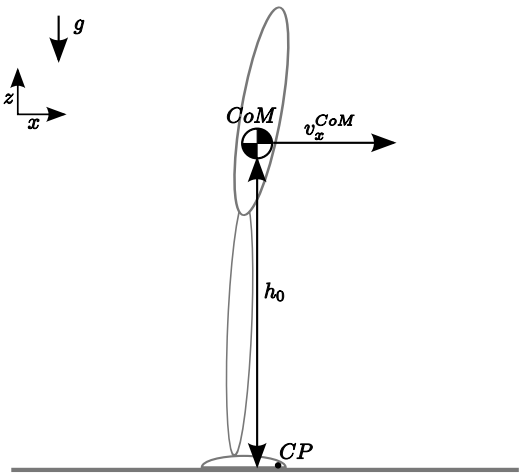


Fig. 2. Simple robot model which is used to calculate the CP position

A sufficient condition to determine if the robot does not fall, is to determine if the robot is able to stop [8]. This can be done by using the Capture Point (CP), which is a point on the ground in which the CoP can be placed in order to stop the robot. The robot is able to stop without stepping, if the capture point is located inside the support polygon. The position of the CP can be calculated by making use of a simplified model of the robot (see figure 2). If we assume that the Center of Mass (CoM) of the robot is moving in a horizontal plane, the CP can be calculated in the following way:

$$x_{Capture} = x_{CoM} + v_x \cdot \sqrt{\frac{h_0}{g}} \tag{1}$$

where x_{CoM} is the position of the CoM, v_x is the horizontal velocity of the CoM, h_0 is the height of the CoM and g is the gravitational constant. The equation shows that the position of the CP depends on the CoM position and CoM velocity. As mentioned earlier we are interested in a full body motion while the robot remains in his current location, in other words it should not take a step. Which means the CP should not move outside the foot support polygon. Combining this information with equation 1 we can determine the minimum and maximum allowed CoM velocity which create a stable full body motion.

$$\frac{x_{Capture}^{min} - x_{CoM}}{\sqrt{\frac{h_0}{g}}} < v_x < \frac{x_{Capture}^{max} - x_{CoM}}{\sqrt{\frac{h_0}{g}}} \tag{2}$$

If the CoM velocity of the full body motion stays within these limits, the robot is able to stop and will not fall, which means the fullbody motion is stable.

In the remaining of this section we will show how we can control the velocity of the CoM of the robot by making use of the linear momentum of the robot.

B. Momentum

The velocity of the CoM of the robot is closely related to the linear momentum of the complete robot. The linear momentum of the CoM of the robot can be written down in two different ways:

1. In terms of the CoM velocities.

$$h_g^{lin} = m \cdot v^{CoM} \tag{3}$$

where h_g^{lin} is the linear momentum of the CoM, m the total mass of the robot and v^{CoM} a 3×1 vector which contains the linear velocities along the x,y and z-axis.

2. In terms of the joint velocities

$$h_g^{lin} = A_g(q) \cdot \dot{q} \tag{4}$$

where A_g is the so called Centroidal Momentum Matrix [7] (appendix A), this matrix $(6 \times n)$ shows how the joint velocities change the linear momentum of the system, q and $\dot{q}$ are $n \times 1$ vectors. They contain the joint angle and velocities of the n joints.

If we combine the equations 3 and 4 we get a linear relation between the momentum of the CoM and the joint velocities, equation 5, because the mass of the robot will not change, we have a linear relation between the joint velocities and the CoM velocities. In other words if we know the v^{CoM} that creates a stable full body motion, we can calculate the corresponding joint velocities.

$$m \cdot v^{CoM} = A_g(q) \cdot q \tag{5}$$

C. Momentum controller

The relation found in equation 5 forms the basis for the momentum controller shown in figure 3. This controller can derive the joint velocities which are needed to obtain a stable full body motion. Based on the current CP position and task motion.

As mentioned earlier a fullbody motion for a human-like structure can be splitted into two independed parts. The first part describes the task motion, the second part describes the motion which is needed to compensate for the task motion. The task motion for a humanoid robot is usually described as a position profile. This task position profile ($q^{task_{ref}}$) has to be converted into a task velocity profile ($\dot{q}^{task_{ref}}$) before we can use equation 5 to determine the stabilising joint velocities. To convert the task position profile into a task velocity profile we use the following P-controller:

$$\dot{q}_{ref}^{task} = K^{task} \cdot \left(q_{ref}^{task} - q\right) \tag{6}$$

where K^{task} is the gain, q_{ref}^{task} the desired joint angle to perform the task and q the current joint angle.

Besides the task velocity profile we need a reference velocity for the CoM to solve equation 5. The desired CoM velocity was determined by using the following controller, which will guide the CP to a desired CP position.

$$v_{ref}^{CoM} = K_p \cdot \frac{1}{\sqrt{\frac{z_{CoM}}{g}}} \cdot \left(x_{CP}^{ref} - x_{CP}\right) \tag{7}$$

where K_p is the gain, z_{CoM} the current height of the CoM, g the gravitational constant,x_{CP} the current CP position and x_{CP}^{ref} the desired CP position. The gain K_p should always be smaller then 1 to guide the CP to the reference position.

We have to scale the v_{CP}^{ref} by the total mass of the robot to obtain a reference for the linear momentum (h_{ref}^{lin}). This results in the following K^{CP} (see figure 3).

$$K^{CP} = \frac{K_p \cdot m}{\sqrt{\frac{z_{CoM}}{g}}} \tag{8}$$

We do not have to create a setpoint for every momentum in h_{ref}^{lin}. For example in a 2d case we only create a setpoint for the linear momentum in x direction to ensure a stable full body motion. The linear momentum in the z-direction is unconstraint. This means that it is allowed to change the height of the CoM of the robot. To be able to solve equation 5 we need at least one constraint momentum.

C.1 Obtaining the unknown joint velocities

The unknown joint velocities (the joint velocities which stabilises the task motion) can be obtained by soling equation 5. However the task motion can be described in two different ways, as a joint velocity profile of a constraint on the joint velocities. To be able to solve equation 5 for the unknown joint velocities we should rearrange it a little bit.

If the task motion is described in terms of joint velocity profiles, the total joint velocity vector ($\dot{q}$) of the system can be split in a part which is needed to perform the task ($\dot{q}^{task}$) and a free part($\dot{q}^{free}$) which can be used to create a compensation motion. The joint velocity vector can be written in the following way:

$$\dot{q} = \begin{bmatrix} \dot{q}^{task} & \dot{q}^{free} \end{bmatrix}^T \tag{9}$$

Combining this with equation 4 will result in equation 10, which has two decoupled parts, the momentum produced by the task motion and the momentum produced by the free joints to create the compensation motion.

$$h_G = A_G^{task} \cdot \dot{q}^{task} + A_G^{free} \cdot \dot{q}^{free} \tag{10}$$

The second way to describe a task motion is to formulate it as a constraint on the joint velocities. In this way it is possible to formulate the task motion as a velocity trajectory of a point on the robot. For example if we would like to kick the ball, we prescribe the velocity profile for the foot. This constraint can be formulated in the following way. The position of a point C on the robot can be written as :

$$x_C = G(q) \tag{11}$$

where x_C is the position of point C in global coordinates and $G(q)$ is the position of point C described in terms of the joint angles. The velocity of point C can be written in the following way:

$$\dot{x}_C = \frac{\partial G(q)}{\partial q} \cdot \dot{q} = J_C(q) \cdot \dot{q} \tag{12}$$

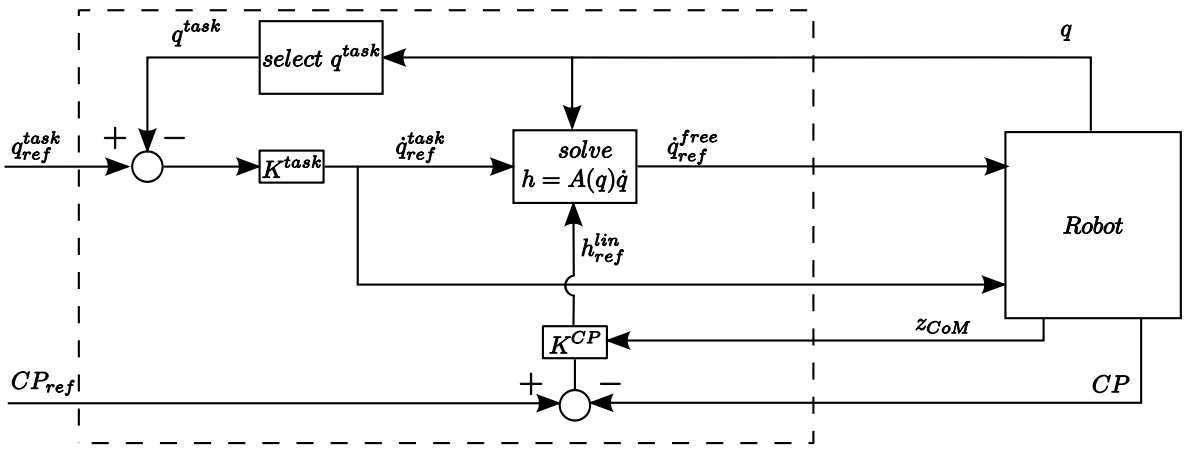


Fig. 3. Overview of the momentum controller used to generate stable full body motions

where $\dot{x}_C$ is the velocity of point C in the global coordinate frame, $J_C(q)$ is the Jacobian of point C and $\dot{q}$ the vector with joint velocities. All the velocity constraints can be written as:

$$\dot{x} = D(q) \cdot \dot{q} \tag{13}$$

where $\dot{x}$ is a vector which contains the velocities in the global coordinate frame and D contains the corresponding Jacobians. These constraints can be added to the momentum equation (equation 4):

$$\begin{bmatrix} h_G \\ \dot{x} \end{bmatrix} = \begin{bmatrix} A_G(q) \\ D(q) \end{bmatrix} \cdot \dot{q} \tag{14}$$

in this equation the upper part will ensure the stability of the robot and the lower part will create the task motion.

By combining equation 10 and 14, we get:

$$\begin{bmatrix} h_G \\ \dot{x} \end{bmatrix} = \begin{bmatrix} A_G(q) \\ D(q) \end{bmatrix}^{task} \cdot \dot{q}^{task} + \begin{bmatrix} A_G(q) \\ D(q) \end{bmatrix}^{free} \cdot \dot{q}^{free} \tag{15}$$

which describes the momentum around the CoM in two decoupled vectors. By solving equation 15 for the $\dot{q}^{free}$ we find the compensation motion, which stabilises the robot while it performs the task motion.

D. Compensation motion

There are two numbers which determine if equation 15 can be solved. These are the number of free joints (n) and the number of constrained momenta plus the number of velocity constraints(l). When solving equation 15 there are 3 possible situations:

1. $n < l$, there is no compensation motion which satisfies equation 15. One can do two things:
   (a) Release a momentum constraint.
   (b) Redesign the task motion in such a way that it can be created with fewer joints.
2. $n = l$, there is only one solution.
3. $n > l$, in this situation there are infinite many solutions which means some kind of selection mechanism should be used to select a compensation motion.

D.1 Selecting a compensation motion

A humanoid robot has many of degrees of freedom, for this reason there is a high chance that we end up with infinite many soloutions for equation 15. The pseudo inverse can be used to select the best soloution for equation 15. The pseudo tries to minimise $\| \dot{q}^{free} \|$, it selects the solution with the smallest joint speeds. But we favour to keep the velocities of the joints which move the largest mass/inertia as low as possible. We can give the pseudo inverse some extra information by using weight factors, by using the generalised inertia matrix (I_{gen}) as weight matrix the pseudo inverse will minimise $\| I_{gen}\dot{q}^{free} \|$. When the weighted pseudo inverse is used the soloution to equation 15 becomes:

$$\dot{q}^{free} = WLS \cdot \left(h_G - \begin{bmatrix} A^{task}_G \\ D^{task} \end{bmatrix}^T \cdot \dot{q}^{task} \right) \tag{16}$$

$$WLS = W^{-1} \cdot \begin{bmatrix} A^{free}_G \\ D^{free} \end{bmatrix}^T \cdot \left(\begin{bmatrix} A^{free}_G \\ D^{free} \end{bmatrix} \cdot W^{-1} \cdot \begin{bmatrix} A^{free}_G \\ D^{free} \end{bmatrix}^T \right)^{-1} \tag{17}$$

$$W = \begin{bmatrix} I^{free}_{gen} & 0 \\ 0 & I \end{bmatrix} \tag{18}$$

where W is the weight matrix which contains $I^{free}_g en$, the entries from the generalised inertia matrix which corresponds with $\dot{q}^f ree$ and an $n \times n$ identity matrix I, where n are the number of velocity constraints.

E. Handling joint and aquator speed limits

After we have used equation 16 to determine the unknown joint velocities, we should check if we can produce these velocities. The joint velocities which can be produced by the robot are bounded to certain limits, which are created by motor performance and geometry of the robot. These joint velocity limits are not incorporated in equation 16. This equation assumes that every joint speed

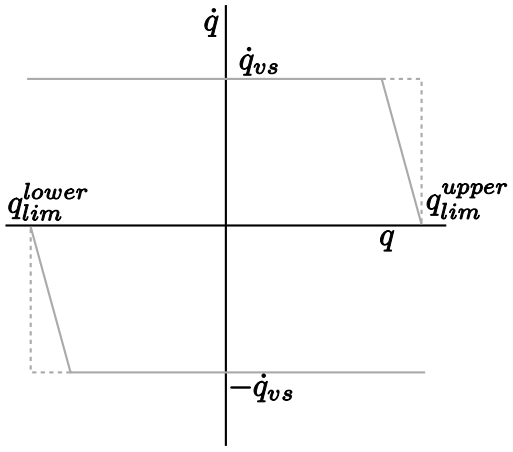


Fig. 4. The limits on the joint velocity at different joint angles

Fig. 5. The soccer robot TUlip

can be realized. Which means we have to check if the calculated joint speeds are with in limits.

Due to velocity saturation the rotational velocity of the motor is limited to $\dot{q}_{vs}$. If a joint is driven into a joint limit the joint velocity will be equal to zero because it is limited by some geometrical constraint. These velocity limits can be plot in a graph, see figure 4. To prevent the joint from reaching his joint limit with a high joint velocity the maximal joint velocity near the joint limit is:

$$\dot{q}_{lim} = k \cdot (q_{lim} - q) \tag{19}$$

where q_{lim} the position of the joint limit and q the current joint angle. The joint speed,$\dot{q}$, for a certain joint is valid if the following holds:

$$\min(\dot{q}_{lim}^{low}, -\dot{q}_{vs}) \leq \dot{q} \leq \min(\dot{q}_{lim}^{upper}, \dot{q}_{vs}) \tag{20}$$

where $\dot{q}_{lim}^{low}$ and $\dot{q}_{lim}^{upper}$ are the maximal velocities which corresponds with the lower and upper joint limits.

We have to check for each joint velocity in $\dot{q}^{free}$ if it is a valid joint speed. If it is valid it can be put in the velocity controller. If not we have to perfrom the following steps until we find a set of valid joint speeds:

1. Set the first invalid joint velocity to the maximal allowed joint velocity.
2. Add the joint velocity to the joint velocity vector $\dot{q}_{lim}$
3. Recalculate $\dot{q}^{free}$ by using equation 16, where $\dot{q}^{task} = \begin{bmatrix} \dot{q}^{task} & \dot{q}_{lim} \end{bmatrix}^T$ and check if the joint speeds are valid.
4. If we have found a set of valid joint velocities, the joint velocities $\dot{q}_{lim}$ are removed from $\dot{q}^{task}$

If $\dot{q}_{lim}$ contains too much joint velocities so equation 16 can not be solved, we have to release a momentum or task constraint. If this is done $\dot{q}_{lim}$ should be removed from the $\dot{q}^{task}$. And it can be tried to find a set of valid joint speeds.

This algorithm does not guarantee that we will find a set of valid joint velocities. In the worst case scenario it has to take a lot of steps before it finds a set of valid joint speeds, which increases the computational load. At the moment the computational load is not a problem, but if it becomes a problem, the algorithm can be enhanced with a predictor which predicts which joint limit constraints are active.

III. Implementation in TUlip

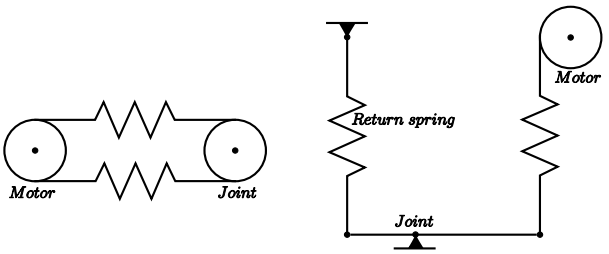
In this section it is shown how the momentum controller was successfully implemented in our soccer robot TUlip (figure 5).

A. TUlip

The soccer robot TUlip was build to participate in RoboCup, an annual robot soccer competition. The final goal of RoboCup is to develop a fully autonomous team of humanoid robots that can play and win against the human world champion soccer team, by 2050 [9].

TUlip is a $1.2m$ tall fully autonomous humanoid robot, with a total mass of $20kg$[10]. It has 12 degrees of freedom (dof), 6 in each leg, all joints are electrically actuated. Most of the joints are actuated by Series Elastic Actuators (SEA) [11], where a spring is placed in the steel cable connecting the motor with the joint. Two encoders are used to measure the rotational difference between the motor and the joint, this determines the spring expansion and thus the actuation torque. The advantage of SEA are low output impedance, high shock tolerance and measurable force output. However the main disadvantage is the low force control bandwidth, around 15 Hz determined experimentally. For this reason it is not possible to use stiff PD-controllers incombination with the SEA.

There are two types of SEA implemented in TUlip. The first type is a bidirectional SEA (figure 6(a)). This actuatot can exert an actuation torque in both directions. The second type an unidirectional SEA (see figure 6(b)). This actuator can only exert an actuation torque in one direction. A return spring is used to create a actuation torque



(a) Bidirectional SEA (b) Unidirectional SEA

Fig. 6. The different SEA types

TABLE I
PROPERTIES OF THE INTERNAL ROBOT MODEL WITHOUT MODEL ERRORS

Link	Length		Mass [kg]		CoM (x,z)	
	x	z			x	z
Torso	0.0	0.600	11.81		-0.010	0.278
Leg	0.0	0.550	7.17	$(2 \cdot 3.58)$	0.010	0.364
Foot	0.1	0.000	0.87	$(2 \cdot 0.43)$	0.027	-0.017

in the opposite direction. This type of SEA is only used in the ankle joint. There are two difficulties controlling the torque for this type of SEA, namely friction and non linearities. The friction is induced by Bowden cables, they are used to connect the motor, located in the torso, to the ankle. A change in joint angle will induce non linearities, because it changes the arm and the direction of the spring forces.

The software system controlling TUlip is based on the concept of independent modules each performing their own specific task, such as motion, vision, communication, world model and strategy. All these modules run on an on-board computer (Diamond Poseidon SBC with $1GHz$ and $512MB$ ram) running Linux. The motion module is implemented as a real-time process running at $1kHz$.

B. Internal model of TUlip

Before we can use the momentum controller an internal model of the robot is needed to derive the Centroidal Momentum Matrix, $(A_g(q))$. This model contains the masses, locations of the CoM of the various links and how they are linked together. Altough TUlip is a 3d robot, we will use an 2d internal model to determine A_g in this research. This is done to simplify the internal model and avoid 3d rotations. Due to control problems of the knee in the real robot the internal model of the robot (figure 7) consists of 3 links (namely a torso, leg and foot), connected by two joints (hip and ankle). The mass of the legs is twice the mass of the lowerleg and upperleg of the real robot and the mass of the foot is twice the mass of the real foot. To determine the properties of the different links of the real robot. The robot was taken apart and the CoM and the mass of each link was determined In table I the properties of the internal model can be found.

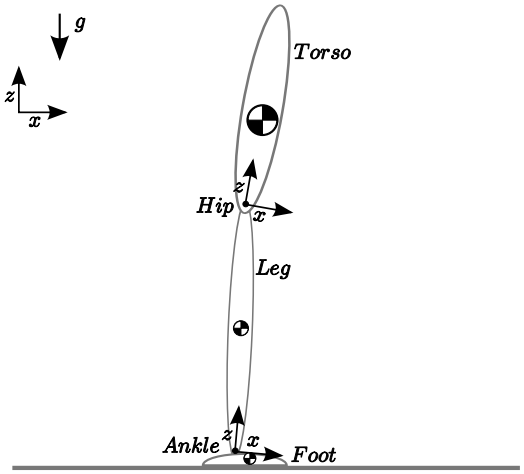


Fig. 7. Internal model of TUlip

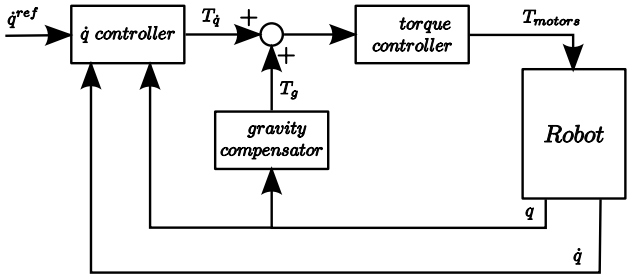


Fig. 8. Overview of the qd controllers

C. Joint Velocity Control

Now we have determined the Centroidal Momentum matrix for TUlip, we can determine the velocities that realize a stable full body motion. In this section we show which controller was used to control the joint velocities of TUlip. Figure 8 shows how the velocity controllers are implemented in TUlip. Besides the velocity controller we use a feedforward gravity compensator, to reduce the nonlinear disturbance induced by the gravity. The torque controllers are implemented as PD-controllers (see appendix B) and convert joint torque into motor torques.

There is one problem controlling the joint velocities. The joint velocity is not measured, but calculated by differentiating the discrete position signal. This results in a very noisy velocity signal and makes it hard to use as a control input. However we can measure the joint position and create a setpoint by intergrating the reference velocity. For this reason the following PD controller was used;

$$T^{ref} = K_p \left(q^{ref} - q\right) + K_d \left(\dot{q}^{ref} - \dot{q}\right) \qquad (21)$$

where K_p and K_d are the proportional and differential gains, q^{ref} is the reference joint angle, q the current joint

angle, $\dot{q}^{ref}$ the reference joint velocity calculated by the momentum controller described in section II and $\dot{q}$ the current joint velocity.

The reference joint position was estimated unsing an integration window. We can estimate the reference joint angle at sample time $k+1$ by using the joint position, q_k, the reference velocity $\dot{q}_k^{ref}$ and the sample time h in the following way:

$$q_{k+1}^{ref} = q_k + \dot{q}_k^{ref} \cdot h \tag{22}$$

Due to time delay, it takes a while before a control signal results in a motion of the joint, we have to use some information from the past. The reference joint angle, q_{k+1}^{ref}, is determined by calculating the mean of N-estimates of q_{k+1}^{ref} each estimate started intergrating at a different q_i $(i = [k-N+1, k])$:

$$q_{k+1}^{ref} = \frac{1}{N} \sum_{i=k-N+1}^{k} \left(q_i + \sum_{j=i}^{k} \dot{q}_i^{ref} \cdot h \right) \tag{23}$$

For our actuators we have chosen $N = 1500$

IV. Experiments

Experiments have been performed on a simulation model (for 3d simulation see appendix C) and the real robot to determine if an stable full body motion could be generated using the momentum controller described in section II and how sensitive the method is for errors in the internal model. This model is needed to generate the Centroidal Momentum Matrix (A_g). But it is often a challenging and time consuming job to determine the internal model. We have investigated two kind of errors in the internal model. The first error was induced by varying the torso mass of the internal model. The second model error was induced by changing the position of the torso mass in the internal model. It was chosen to induce an error on the torso mass and the position of the torso mass, because it is the heaviest mass in the robot. So it is likely that it affects the performance of the robot the most. By varying the torso mass and the location of this mass the calculation of the CP and the Centroidal Momentum matrix are affected. The gravity compensator is not affected because it is likely one will tune the gravity compensator to give a satisfactory result.

The performance of the robot was measured by the position of the CP with model errors (CP^0), this CP is calculated using the internal model without model errors. The closer the CP^0 stays to the setpoint, the better the result. If the CP^0 moves outside the foot, the robot will fall. During the experiments it was tried to keep the CP with model errors (CP^*) in the middle of the foot. This position of the CP leaves the largest room for errors and disturbances with random directions.

During all experiments the hip was commanded to follow the task motion shown in figure 9. The robot will bend forward and move back to his initial position.

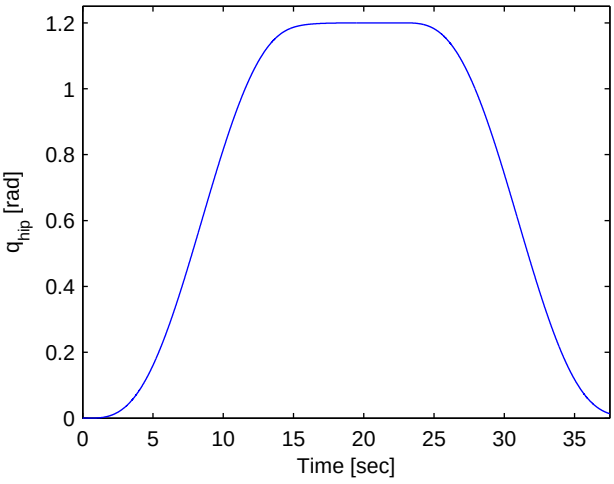


Fig. 9. The task motion for the hip joint

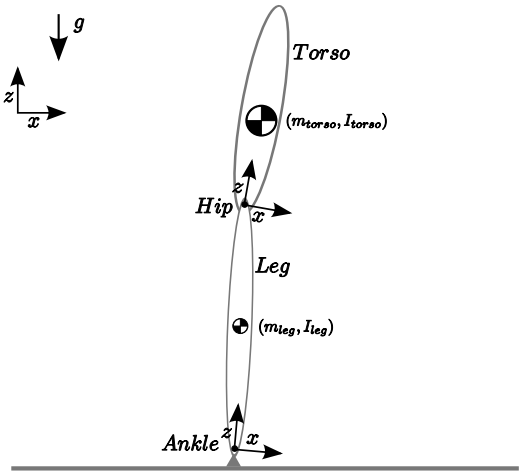


Fig. 10. The simulation model

A. Simulation

In the simulation TUlip was modeled as a double inverted pendulum (see figure 10). The links represents the torso and legs. The connection between the ground an the robot is modeled as a hinge at the ankle. It was assumed that there is enough friction between the ground and the foot (i.e the foot will not move in x-direction). The SEA are not modeled, the actuators are modeled as ideal torque sources. The properties of the links can be found in table II

The ankle is located in the middle of the foot. For this reason the simulation starts in a upright position, where the CP is located at the ankle.

Two experiments where performed. In the first experiment the torso mass (m_{torso}) in the internal model was varied (see table III). Figure 11 shows the full body mo-

TABLE II

PROPERTIES OF THE SIMULATION MODEL

Link	Length x	Length z	Mass [kg]		CoM (x,z) x	CoM (x,z) z	Inertia
Torso	0.0	0.600	11.81		-0.010	0.278	0.35
Leg	0.0	0.550	7.17	$(2 \cdot 3.58)$	0.01	0.178	0.18

TABLE III

DIFFERENT m_{torso} FOR THE INTERNAL MODEL DURING SIMULATION

modelerror [%]	-100	-75	-50	-20	0	20	50	75	1000
m_{torso} [kg]	0.0	2.95	5.9	9.4	11.8	14.2	17.7	20.7	118

tion for 3 cases, namely no model error, an model error of -100% and an model error of $+1000\%$. How the CP^0 changes can be seen in figure 12. This figure shows the CP^0 stays in the middle of the foot if there is no model error. If the internal model mass is decreased, the mass is under estimated, the CP^0 will move away from the middle of the foot. The motion will become unstable if the torso mass becomes close to zero. Notice that the motion with a model error of -100% will cause a fall, because the CP^0 moves out side the foot support polygon. If the torso mass is over estimated, the CP^0 will also move away from the middle of the foot, however it will not become unstable. The CP^0 will converge to the CP^0 calculated for a internal model where all the mass is located at the torso.

In the second experiment the position of the torso mass in the internal model was changed (see table IV). Figure 13 shows the screenshots of the full body motion for 3 cases, namely if there is no model error, if there is a model error of $+100\%$ and a model error of -100%. The postion of the CP^0 can be seen in figure 14. If the model error is -100%, the torso mass is located at the hip. The motion will cause a fall, because the CP^0 will move outside the foot support polygon. With a model error of $+100\%$, the torso mass is almost located at the top of the torso. The controller is still able to create a stable full body motion, however the CP^0 is located close to the edge of the foot,

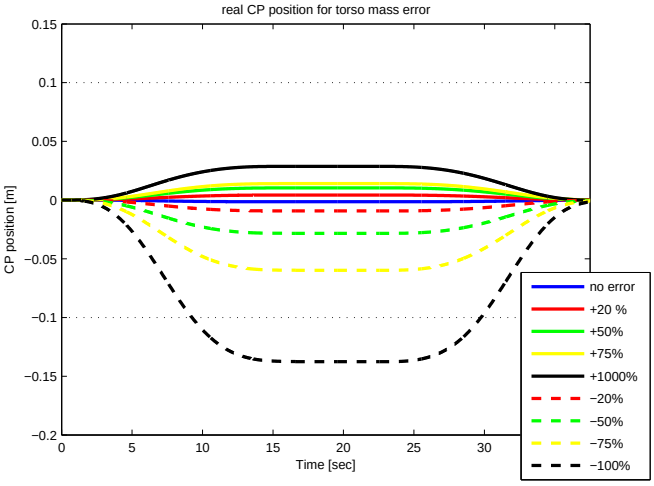


Fig. 12. The position of the CP^0 with varying m_{torso}

TABLE IV

CHANGE OF THE TORSO MASS POSITION DURING SIMULATION EXPERIMENTS

modelerror [%]	-100	-75	-50	-20	0
z_{CoM}^{torso} [m]	0.0	0.070	0.139	0.222	0.278
modelerror [%]	20	50	75	100	
z_{CoM}^{torso} [m]	0.334	0.417	0.487	0.556	

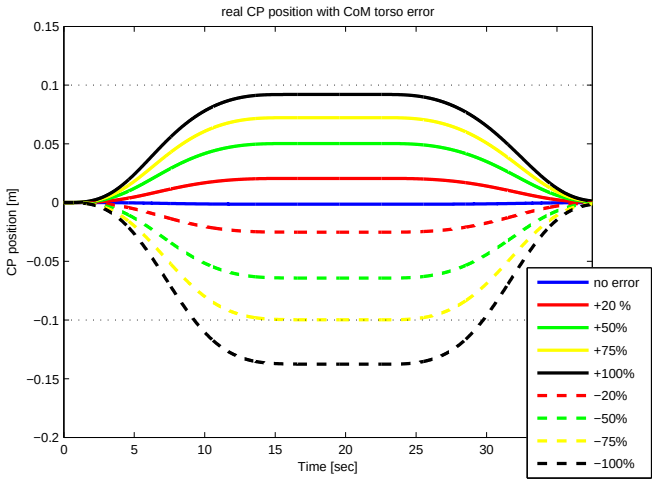


Fig. 14. The position of the real CP model error CoM position torso

this leaves almost no room for disturbances.

A.1 Conclusion

The simulation shows that the internal model does not have to be very accurate to create a stable full body motion using the momentum controller. The algorithm can handle model error for the torso mass of -90% to $\infty\%$ and a miss estimation of the torso mass location of -75% to $+100\%$. The simulation also shows that it is better to over estimate the dominant mass (torso mass) in the internal model then to under estimate it. If the error in the estimation of the location of the dominant mass is small ($< 20\%$), the effect of an over- and under estimation are the same. The simulation shows that it is more important to know the position of the mass then the exact mass.

B. Real Robot

During the experiments with the real robot it was assumed that both ankle and hip joints move in the same way, and that each leg carries half of the robot weight. The joint angles of the left leg where used to calculate the $A(q)$ and $\dot{q}_{task}^{ref}$. The momentum controller was used to calculate the reference joint velocity for the ankle joint. Each joint had his own velocity controller, but the reference velocity for the left and right joint was the same.

At the beginning of the experiment TUlip was put in an upright standing posture. In this posture the CP^0 was located close to the middle of the foot. TUlip stood in this position for at least 30 seconds, before the task motion started. The torso mass of the internal model was varied (see table V), which means $A(q)$ and the calculation of CP

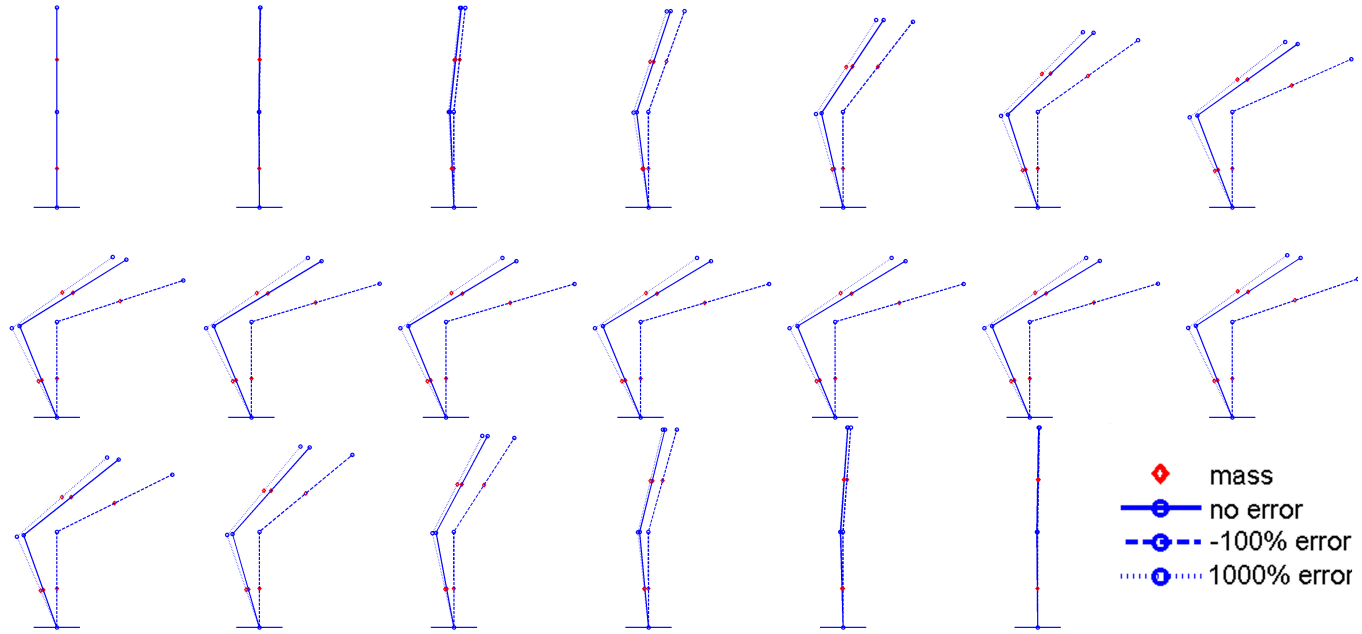


Fig. 11. Screenshots of the bending motion calculated by the simulation with varying m_{torso}.

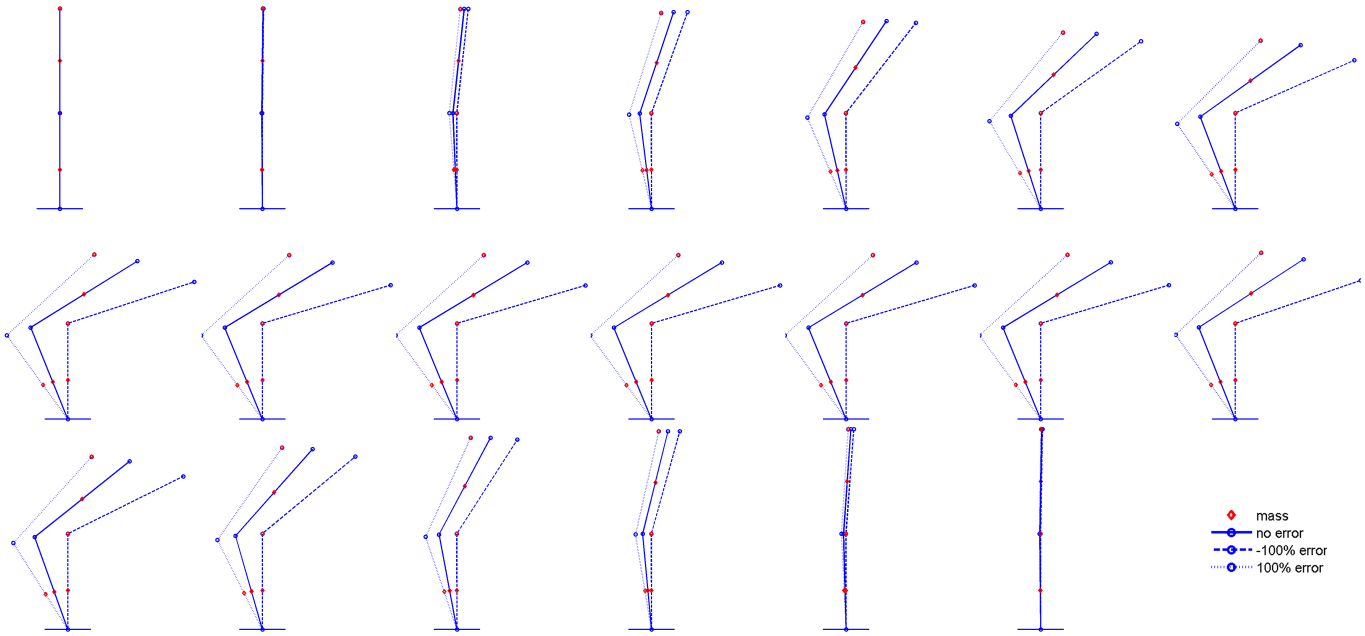


Fig. 13. Screenshots of the bending motion performed by the simulation, with error in the torso mass position.

changes. The task motion was performed at least 5 times for each torso mass, except for the motion with a model error of -75% because this motion was not stable. Figure 15 shows screenshots of the bending motion performed by TUlip. In figure 16 the position of the CP^*,the capture point calculated with the model errors, is shown. The figure shows that almost all the CP^* are located close to the middle of the foot, this indicates that the CP controller is working well. The deviation of the CP^* from the middle of the foot is mostly due to friction in the ankle joint. Figure 17 shows the position of CP^0 for the different model errors, it has almost the same shape as figure 12. Except that in the simulation the momentum controller can still generate a stable full body motion with an model error of -75% and in the real robot this error caused a fall. However this is not strange, because in the simulation the under estimation of the torso mass is the only error, while in the real robot there are all kind of errors induced by for example the controllers, friction, environment etc.

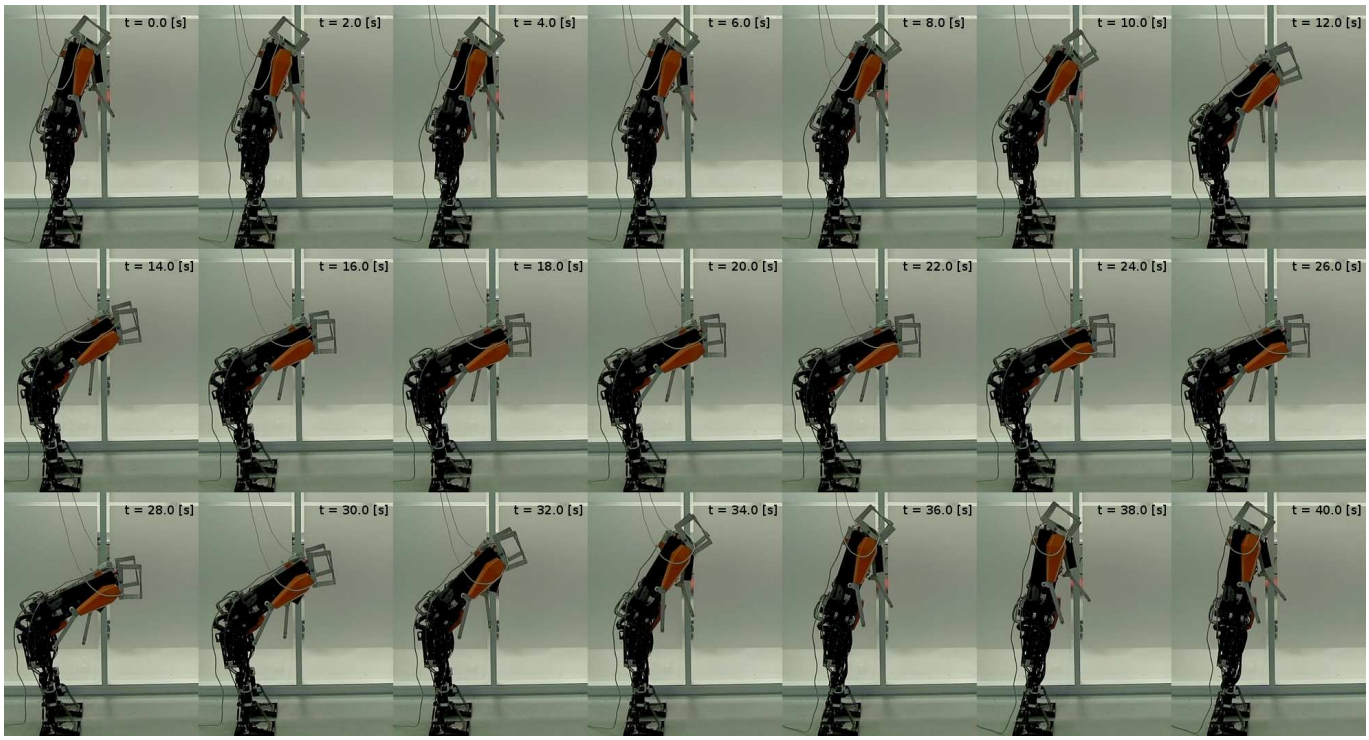


Fig. 15. Screenshots of the bending motion performed by the TUlip

TABLE V

DIFFERENT m_{torso} FOR THE INTERNAL MODEL

moddel error [%]	-75	-50	-20	0	20	50	75	100
m_{torso} [kg]	2.95	5.9	9.4	11.8	14.2	17.7	20.7	23.6

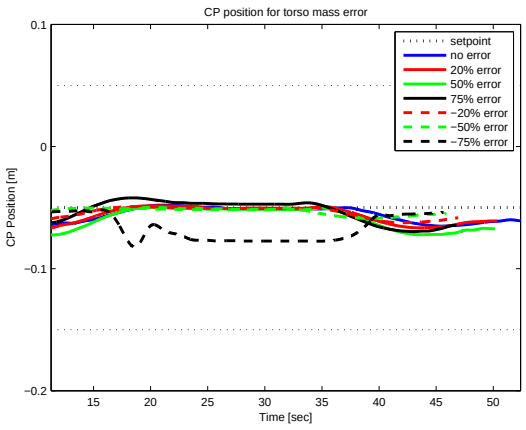


Fig. 16. The CP^* position during bending forward motion, with different model errors

V. CONCLUSION

The main goal of this paper was to obtain stable full body motion, for a humanoid robot, without defining all joint trajectories in advance. To create stable full body motion the motion was split in a task and compensation part. The task motion was generated offline while the compensation motion was created online, by using the linear

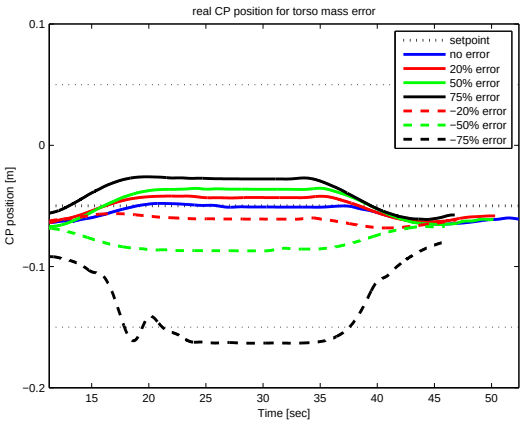


Fig. 17. The CP^0 position during bending forward motion, with different model errors

relation between joint velocities and the momentum of the robot. In this way we where able to create stable full body motion for a bending forward task, in simulation and on a real humanoid robot.

We have also shown that the internal model of the robot does not have to be very accurate to get good results. It turned out that the controller can handle errors in the torso mass and the location of the torso mass of 75 % and more. An error in the position of the CoM of the torso mass has a larger influence on the performance of the controller then an error in the torso mass. So it is more important to estimate the position of the CoM then to estimate the exact weight.

A. Future work

It has been shown that the momentum controller can be used to create a full body motion for a robot in a 2d case. Different steps need to be taken to make better use of the frame work.

The first step is to make a 3d implementation. To do this a 3d internal model of the robot is needed to derive the A_g matrix which solves equation 16 for a 3d robot. In the 3d case the CP controller (equation 8) should be implemented twice to create a setpoint for the linear momentum in x and y-direction. Before the compensation joint speeds can be calculated in the 3d case, one have to be sure that the matrix $\begin{bmatrix} A_G & D \end{bmatrix}^T$ is not close to singularity. This can be checked by calculating the singular value decompensation of the matrix. If one of the singular values becomes close to zero, the matrix is close to singularity. The singular values give an indication of how easy it is to generate the specified momentum or meet the constraint. For example if the singular value which corresponds to the linear momentum in x direction is close to 0, it means that it is hard to change the momentum in x-direction [7].

The second step is how to deal with large disturbances. The current implementation of the momentum controller is able to handle small disturbances. But the disturbance handling can be improved. For example by implementing a stepping strategy if the CP moves outside the foot support polygon. Or a so called hip strategy, which is observed by humans. A human will bend forward if he is receives a large push in the back. Which creates a moment around the CoM that helps to remain balanced.

It is expected that the performance of the momentum controller can be improved if it is implemented as a moddel predictive controller especially if external disturbances in the future are known. In this way information of the future can already be used to generate a compensation motion.

The use of a weightmatrix to select the best free joint velocities need some more investigation. This was not really tested due to insufficient free joints. The selection mechanism will become more important in the 3d case, due to more redundancy.

Appendices

A. Centroidal Momentum Matrix

The Centroidal Momentum Matrix is a projection of the system momentum matrix (A) onto the centroidal coordinate frame [7]. The system momentum matrix is a product of the system inertia matrix (I) and the system Jacobian (J). The system Jacobian gives the relation between the system speeds and the joint velocity.

$$A = I \cdot J \tag{24}$$

The Centroidal Momentum Matrix can be expressed as follows:

$$A_G = X_G^T \cdot A \tag{25}$$

where X_G is the projection matrix from centroidal to link coordinates. The matrix A_G is the same as $\begin{bmatrix} M_{\dot{\theta}} & H_{\dot{\theta}} \end{bmatrix}^T$ which is mentioned by Kajita [6].

B. Torque controllers

To control the torque in the SEA, digital PID-control[12] is used. The input for the controller are the reference joint torque (u_c) and the current joint torque (y). The last inputs is calculated in the following way:

$$y = 2 \cdot K_s \cdot r_j \cdot (\phi_m \cdot r_m - \phi_j \cdot r_j) \tag{26}$$

where K_s is the spring constant of the spring in the SEA, r_m and r_j are the motor and joint pulley radius and ϕ_m and ϕ_j are the motor and joint angle

The proportional part of the controller is:

$$P(kh) = K(u_c(kh) - y(kh)) \tag{27}$$

where K is the gain, k sample number and h the sample time.

The first part of the integral part of the controller approximates the integral term $I(t) = \frac{K}{T_i}\int^t e(s)\,ds$. The last part resets the integrator to prevent integrator windup:

$$I(kh+h) = I(kh) + \frac{Kh}{T_i} \cdot e(kh) + \frac{h}{T_t}(y - v) \tag{28}$$

where T_i is the integration time, e is the error between the commanded value u_c and the process output y, T_t the tracking time constant and v is the control torque. The differentiation part of the controller is estimated by:

$$D(kh) = \frac{T_d}{T_d + Nh} D(kh-h) - \frac{KT_dN}{T_d + Nh}(y(kh) - y(kh-h)) \tag{29}$$

where T_d is the derivative time and N the maximum derivative gain. Calculating a derivative of a discrete signal will introduce a lot of noise. The above approximation will approximate the derivative at low frequencies and will limit the gain at high frequencies. The controller out put is calculated in the following way:

$$v(kh) = P(kh) + I(kh) + D(kh) \tag{30}$$

The parameters K, T_i, T_d,T_t and N must be chosen for the controller. The parameter N was given a fixed value for all the joint controllers namely $N = 1.5$. T_t is realted to the integration time for this reason T_t was set equal to T_i. The values for K, T_i and T_d were determined experimentally by using the Ziegler and Nichols step response method [12].

The openloop step response of the SEA was determined after the joint was locked. The tangent to the steepest slope of the openloop response was drawn (see figure 18) and the parameters R ,the slope of the tangent, and L, the apparent dead time, were determined. The control parameters are obtained by using the following formulas:

$$K = \frac{1.2}{R \cdot L} \qquad T_i = 3 \cdot L \qquad T_d = 0.5 \cdot L \tag{31}$$

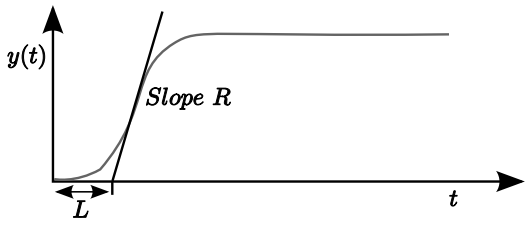


Fig. 18. Determination of the parameters R slope of the steepest tangent and L,apparent deadtime

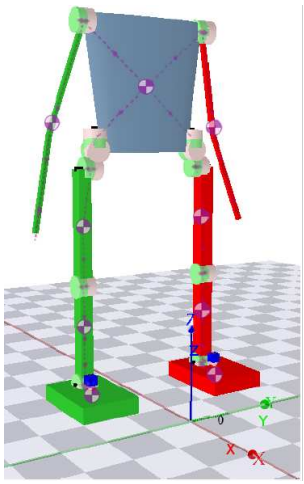


Fig. 19. Simulation model of TUlip

C. 20-Sim simulation

To speedup the implementation of the momentum controller into the motion control software of TUlip and to test the algorithm. The motion control software was used in combination with the simulation package 20-sim. An 3d multi-body simulation model of TUlip was already available in this simulation package [13]. This simulation model can be controlled with the same software which is used to control the real robot. This makes it possible to test the motion control software in simulation (an ideal world)first and if it works it can be adjusted for the real robot.

The 20-Sim simulation model of TUlip has also 14 dof, see figure 19. All the dof are actuated. All tree types of actuators are modeled in the simulation model. The direct actuators are modeled as ideal inertia free motors, which can deliver unlimited torque. The actuator model of the SEA is more advanced. This model contains the springs between the motor and the joint, see table VI for spring properties, the motor model takes electrical resistance, inductance, torque constant and speed constant into account. The motor model data can be found in table VII.

The mass and the mass distribution of the simulation model is different from the real robot. The mass of the real robot is 20 kg while the mass of the simulation model is 10 kg. In the simulation model it is assumed that the mass is homogeneous distributed along the link which is not true for the real robot. In table VIII the size of the different masses and locations of the CoM can be found. For this reason the simulation can be used to test the motion control software, but if we use it on the real robot we have to re-tune the controllers.

TABLE VI
Spring properties

Spring	Stiffness	Unit
Single pulley	14.5	[kN/m]
Single pulley return	32.16	[kN/m]
Double pulley	87.5	[kN/m]

TABLE VII
Motor model data based on maxon RE30

Parameter	Value	Unit
electrical resistance	0.611	[Ω]
inductance	0.119	[mH]
torque constant	25.9	[mNm/A]
speed constant	38.64	[rad/sV]

A. Internal robot model

In the simulation the internal model of the robot consisted out of 4 links, namely torso, upperleg, lowerleg and a foot. The properties of these links can be found in table IX.

B. results

At the start of the simulation experiment the robot was put in a initial position, where the CP was located in the middle of the foot. In stead of following a position task, the hip joint was commanded to follow the velocity task shown in figure 20. So the P-controller in figure 3 was not needed to create $\dot{q}_{ref}^{task}$. It was tried to keep the CP in the middle of the foot during the whole motion. Figure 21 shows snapshots of the resulting motion. Figure 20 shows the desired joint velocities for all the joints which creates a stable bending motion, as well as the real joint velocities. We can see that the velocity controllers follow the desired

TABLE VIII
Masses of the simulation model and the real robot

link	Simulation mass	CoM x	CoM y	CoM z	Real Robot mass	CoM x	CoM y	CoM z
Trunk	5.40	0.00	0.10	0.30	11.81	-0.009	0.104	0.278
Upperleg	1.75	0.00	0.00	0.15	2.50	-0.001	-0.030	0.177
Lowerleg	0.25	0.00	0.00	0.15	1.08	0.038	0.000	0.160
Foot	0.30	0.05	0.00	0.00	0.43	0.027	0.000	-0.017

TABLE IX
Properties of the internal robot model used in simulation

Link	Length x	Length z	Mass [kg]		CoM (x,z) x	CoM (x,z) z
Torso	0	0.6	5.4		0	0.30
Upperleg	0	0.3	3.5	$(2 \cdot 1.75)$	0	0.15
Lowerleg	0	0.3	0.5	$(2 \cdot 0.25)$	0	0.15
Foot	0.1	0.0	0.6	$(2 \cdot 0.3)$	0	0.05

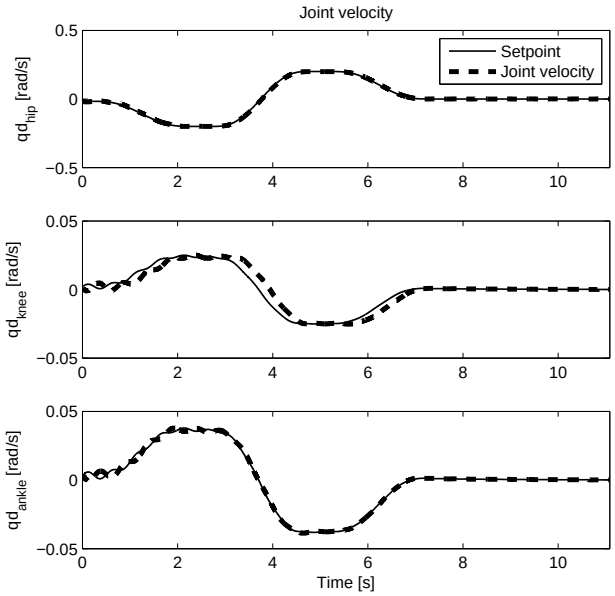


Fig. 20. The joint velocities which create a stable fullbody motion, for the known hip velocity

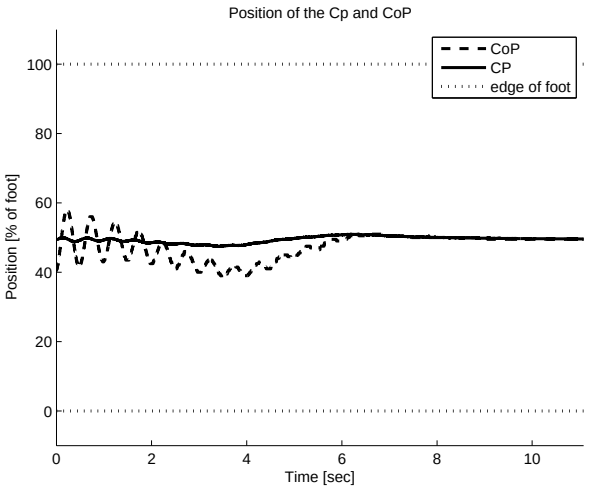


Fig. 22. The position of the CP while TUlip bend forward

joint velocities quite well. It can also been seen that the knee and ankle joint are used during the compensation motion. Figure 22 shows that the CP stays close to the middle of the foot during the whole motion.

The simulation results show that the momentum controller is able to create stable full body motion, even if we have a redundant system. In this case we controlled only the linear momentum in x-direction and two joint were used to control this momentum.

References

[1] Nancy S. Pollard, Jessica K. Hodgins, Marcia J. Riley, and Christopher G. Atkeson, "Adapting human motion for the control of a humanoid robot," in *Proceedings of International Conference on Robotics and Automation*, 2002, pp. 1390–1397.

[2] Anirvan Dasgupta and Yoshihiko Nakamura, "Making feasible walking motion of humanoid robots from human motion capture data," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1999, vol. 2, pp. 1044–1049.

[3] O. Khatib, L. Sentis, J. Park, and J. Warren, "Whole-body dynamic behavior and control of human-like robots," *International Journal of Humanoid Robotics*, vol. 1, pp. 29–43, 2004.

[4] Marko Popovic, Andreas Hofmann, and Hugh Herr, "Angular momentum regulation during human walking: biomechanics and control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004, pp. 2405–2411.

[5] DA Winter, "Human balance and posture control during standing and walking," *Gait & Posture*, vol. 3, pp. 193–214, 1995.

[6] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Resolved momentum control: humanoid motion planning based on the linear and angular momentum," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, 27–31 Oct. 2003, vol. 2, pp. 1644–1650.

[7] D. E. Orin and A. Goswami, "Centroidal momentum matrix of a humanoid robot: Structure and properties," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008*, 22–26 Sept. 2008, pp. 653–659.

[8] J.E. Pratt and R. Tedrake, "Velocity-based stability margins for fast bipedal walking," in *Fast Motions in Biomechanics and Robotics*. 2006, vol. 340 of *Lecture Notes in Control and Information Sciences*, pp. 299–324, Springer Berlin / Heidelberg.

[9] "Robocup," http://www.robocup.org.

[10] Philip Heijkoop, Tomas de Boer, Arjan Smorenberg, Eelko van Breda, Guus Liqui Lung, Freerk Wilbers, Corne Plooij, Gijs van der Hoorn, Edwin Dertien, Gijs van Oort, Martijn Wisse, Pieter Jonker, Stefano Stramigioli, Henk Nijmeijer, and Thom Warmerdam, "Dutch robotics 2008 teen-size team description," *Humanoid League Team descriptions*, 2008.

[11] G. A. Pratt and M. M. Williamson, "Series elastic actuators," in *Proc. IEEE/RSJ Int Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots' Conf*, 1995, vol. 1, pp. 399–406.

[12] Karl Johan Aström and Bjorn Wittenmark, *Computer-Controlled Systems: Theory and Design*, LinkUpper Saddle River : Prentice-Hall, 1997.

[13] Peter Daemen, "Zmp based control in 3d passive dynamic walking," M.S. thesis, University of Twente, February 2008.

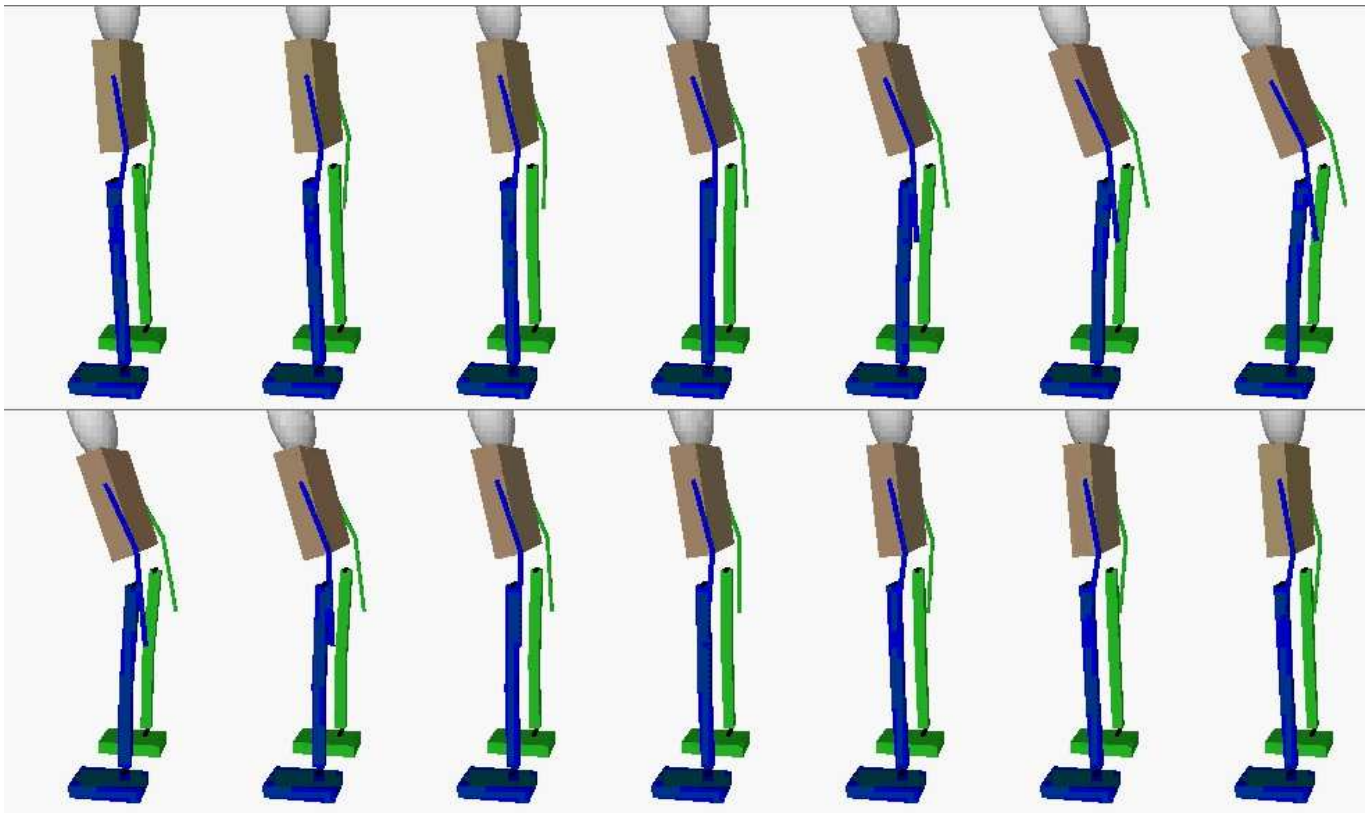


Fig. 21. Snapshots of TUlip bending forward motion in simulation