Developing a kinematic and a dynamic model of a bipedal robot with twelve degrees of freedom

Developing a kinematic and a dynamic model of a bipedal robot with twelve degrees of freedom

W.Schwanen
DCT report: 2002-71

Eindhoven, November 2002

W.Schwanen, student id: 444316

Coach: Dipl.-Ing. Jens Hofschulte
Prof. dr. H. Nijmeijer

Contents

List of Figures

Chapter 1

Introduction

The Institute of Automatic Control at the University of Hannover is part of the Faculty of Electrotechnics. Within this department much research is carried out in the field of service robots. This research is carried out because it is believed that in the future service robots will be of great importance and relatively less is known about them at the moment. Furthermore service robots ask for totally different demands of the development and software. Therefore in the past BARt-UH has been built, a bipedal autonomous robot. BARt-UH possesses six degrees of freedom, three in each leg. These are all rotations around the same axis. It is therefore sufficient to use a planar model for the description of the robot.

Nowadays a bipedal autonomous robot with twelve degrees of freedom, BARt-UH II, which can move in three directions, is being developed. The final aim is to let BARt-UH II walk different patterns. Therefore a controller is needed. But before BART-UH II can be controlled a correct dynamic model has to be developed. There are various ways to derive a dynamic model, the Lagrange equation or the Newton-Euler method. The choice is made to develop the dynamic model by means of the Lagrange equation. For the Lagrange equation it is essential to have a correct kinematic model while the positions of the centers of mass of each body have to be determined. But firstly the generalised coordinates have to be chosen. It is then possible to determine the desired positions. Subsequently the direct kinematics has been determined. This is however not enough for the kinematic model. It is also necessary to determine the generalised coordinates as a function of the positions, this is called the inverse kinematics. The inverse kinemaitcs is necessary because prescribed trajectories are used. In the case of BARt-UH II the position of both feet will be described. Hence equations for the generalised coordinates will be developed.

This report documents a traineeship of 14 weeks at the Institute of Automatic Control. The aim of the traineeship is to develop a kinetic- and dynamic model for BARt-UH II. The report shows how the development of both models is carried out. Before these models are developed BARt-UH II is introduced. A multi-body model for the robot is derived and the important characteristics of each body are given. Another point of interest is the validation of the developed models. Therefore several simulations are carried out. Different algorithms are used during the simulations. So a closer look is taken at the results of these simulations. It is however necessary to develop a new simulation tool, because no standard test application is available which can visualise the motion of BARt-UH II. This test application is shown and the different modes are introduced. Furthermore it is explained how the different tools, used for the simulations, cooperate. At the end some conclusions are drawn and a couple of recommendations for further research are given.

Chapter 2

BARt-UH II

The Institute of Automatic Control is part of the University of Hannover. One of its main research areas is the field of service robots. In 1999 a Bipedal Autonomous Robot with six degrees of freedom has been built at the Institute of Automatic Control. All these degrees of freedom are rotations around the same axis. Therefore this robot, BARt-UH, (figure 2.1), can only move in two directions, forwards and backwards. Subsequently a planar model is sufficient to describe its motion [JH99].

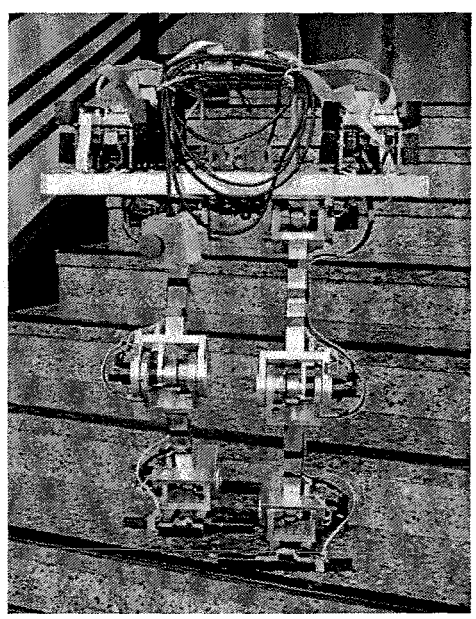

Figure 2.1: BARt-UH

Currently at the Institute of Automatic Control a successor for BARt-UH, BARt-UH II, which can furthermore move sideways and can turn around, is in development.

2.1 General description

BARt-UH II represents the structure of the human legs and can therefore move into three directions. Each leg consists of several rigid bodies interconnected with a variety of joints. The function of a joint is to restrict several degrees of freedom of a body. The robot consists in total of seven rigid bodies. It possesses one torso; each leg is connected with the torso by means of the

neck of femur. Furthermore each leg is divided into three bodies, the thigh, the lower leg and the foot. The robot is symmetric around the z-direction, i.e. the vertical direction, implying that the bodies in the left and the right leg have the same properties.

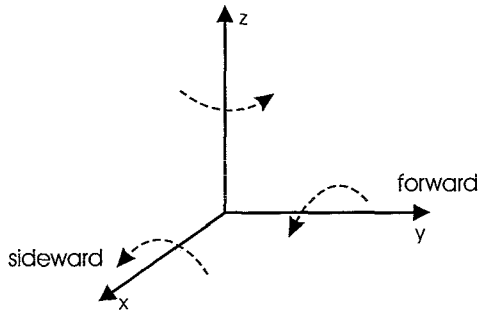

Figure 2.2: The coordinate frame

Figure 2.2 shows the coordinate frame which is used. The z-axis lies vertical, while the direction of the y-axis is in the direction BARt-UH II walks. Subsequently the x-axis lies orthogonal to both axes. In the same figure the positive direction for the rotations can be seen.

Each leg possesses six degrees of freedom. Three degrees of freedom in the hip, one in the knee and two in the ankle. In the hip a rotation around each axis is possible. In the knee only a rotation around the x-axis is possible. Subsequently, the ankle possesses rotations around the x- and y-axis.

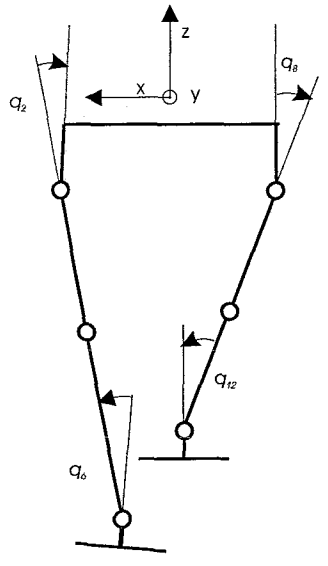

Figure 2.3: Front view of the model for BARt-UH II

So the rotation around the z-axis is the same for the whole leg, because only in the hip a rotation around the z-axis can be carried out. Furthermore the thigh and the lower leg also have the same rotation around the y-axis.

It is also important to notice that BARt-UH II, in contrast to BARt-UH, can only walk in a human-like manner (figure 2.5). This implies that the knee always points forward. Therefore the angles between the thigh and the lower leg, q_4, respectively q_{10}, always have a negative value. Another way of walking is the bird-like manner, in which the angles just have the opposite sign, so they are always above zero.

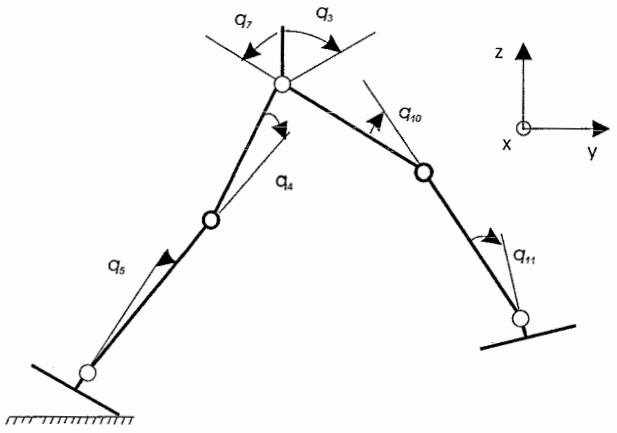

Figure 2.4: Side view of the model for BARt-UH II

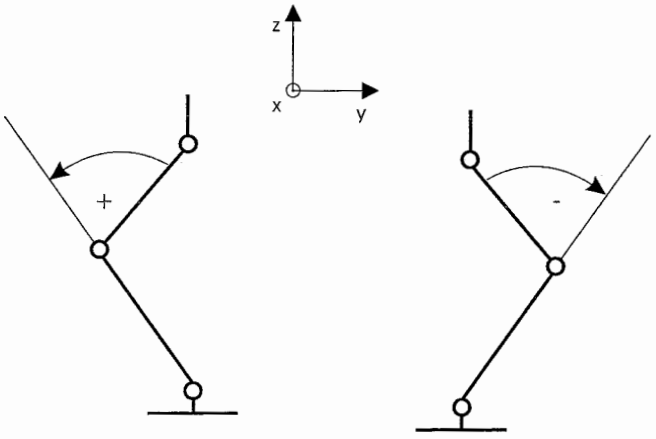

Figure 2.5: Bird-like and human-like walking

The presence of an actuator for each rotation is needed. Three actuators will be placed in the hip. These actuators can be placed in different ways. In the first method the actuators are placed above each other. In the second way a construction is made in which the actuators are placed in a circle. This construction, the Agile-Eye (figure 2.6), has the advantage that the actuators are part of the torso, so the actuators do not rotate when a rotation in the hip is carried out. Subsequently the rotated mass is smaller. The Agile-Eye however has the disadvantage that one actuator alone cannot rotate the leg around one specific axis. So two or three actuators are needed for one rotation.

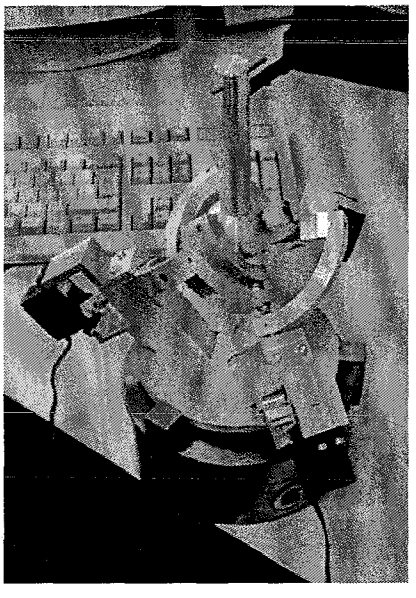

Figure 2.6: The Agile-Eye

As it seemed at the start of this project the actuators would be placed in the first manner, above each other. Therefore the rotations in the hip take place in the following way. First there is rotated around the z-axis because it is expected this rotation will be used less than the other two and therefore it has the greatest mass to rotate. Here after the rotation around the y-axis takes place. Subsequently the rotation around the x-axis is carried out last. During the project however it becomes clear that it is most likely that the Agile-Eye will be used. However the decision is made that the sequence of the rotations does not change.

2.2 Bodies of the BARt-UH II

During walking it is necessary to control the robot in order to carry out the prescribed motion. Therefore the main controller will be placed on the torso of the real robot. The torso has a mass m_1 and the moments of inertia J_{x1}, J_{y1} and J_{z1}. Its dimensions are the same as the dimensions of the torso of BARt-UH. It is assumed that the center of mass lies exactly in the middle. The aim is to ensure the torso keeps flat all the time, so the torso does not rotate. Due to the complex motion this cannot be ensured. Therefore it is necessary to take the rotations around all three axis of the torso into account.

The legs are at a certain distance, *Beinabstand*, from each other. The thigh has length l_2 and its diameter is the so-called *Rohrdurchmesser*. These dimensions are measured. Furthermore its mass is m_2 and the moments of inertia are J_{x2}, J_{y2} and J_{z2}. The center of mass lies at a distance a_2 from the top of the thigh. The lower leg has length l_3 and its diameter is the so-called *Rohrdurchmesser*. These dimensions are measured too. Figure 2.7 shows the models for both

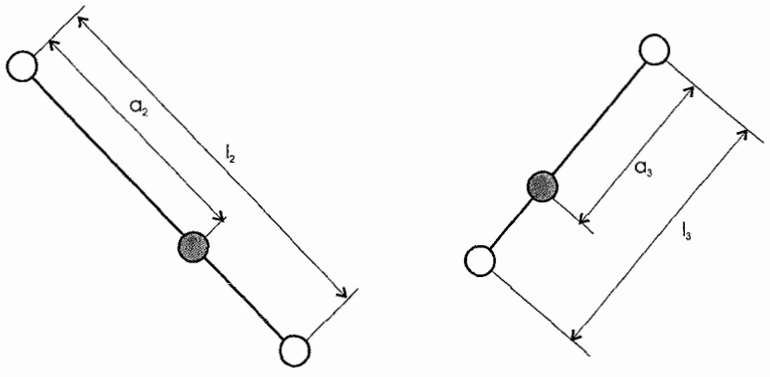

Figure 2.7: The model for the thigh and the lower leg

legs. Furthermore its mass is m_3 and the moments of inertia are J_{x3}, J_{y3} and J_{z3}. The center of mass lies at a distance a_3 from the top of the thigh.

The last body is the foot. It possesses a mass m_4 and its moments of inertia are J_{x4}, J_{y4} and J_{z4}. Its dimensions are taken from [Ger01]. Force sensors are integrated in both feet, so the contact forces can be measured.

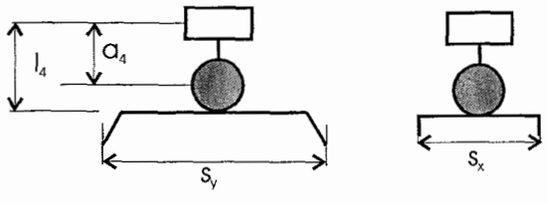

Figure 2.8: The model used for the foot

Figure (2.9) shows the part of BARt-UH II which is already finished. It shows the foot, the lower leg and the knee. Also the thigh is finished. Subsequently, only the hip and the torso have to be build.

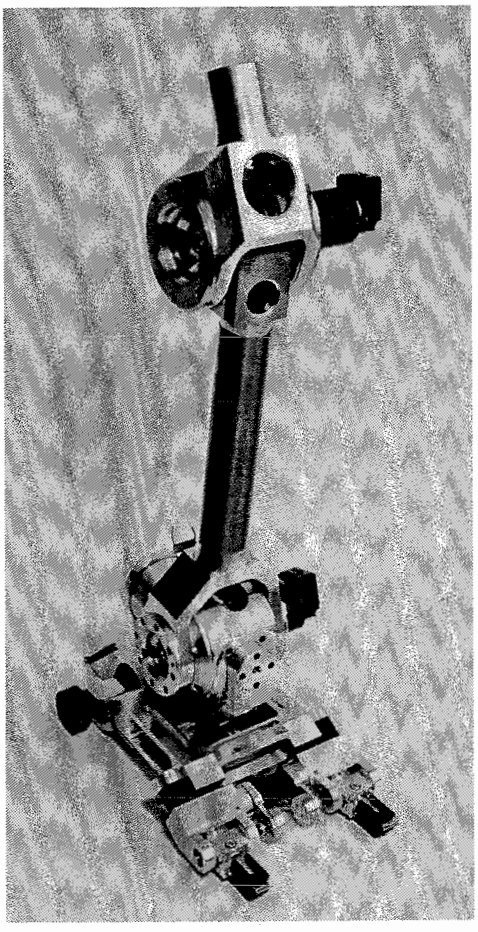

Figure 2.9: The leg of BARt-UH II

Chapter 3

Kinematics

The transformation which describes the relationship between the generalised coordinates, q, and the coordinates, r, is called the kinematics of a multi-body system. This relationship can be described in two ways: the direct kinematics and the inverse kinematics.

3.1 Direct kinematics

For the direct kinematics we have to develop equations for the positions of the center of mass of each body. The first step is to choose a 'reference point', the center of mass of the torso. It is then possible to calculate the positions of the center of mass for each body. Furthermore the direct kinematics of an additional point in each hip is derived. These points are needed for the development of the inverse kinematics. The last points wherefore the direct kinematics have to be derived are the virtual contact points. The feet rotate around these points (figure 3.1).

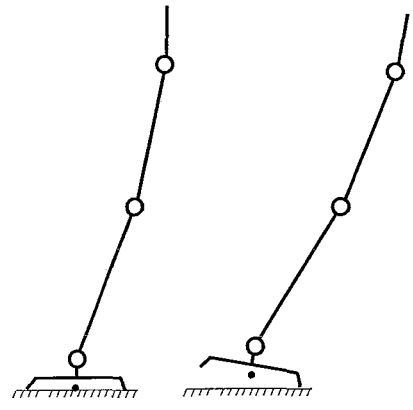

Figure 3.1: The virtual contact point

If the foot stands flat on the ground, the virtual contact point lies on the ground. If the foot starts to rotate, this point moves to a different position, because its position relative to the foot cannot change.

Commonly rotation matrices are used to describe the direct kinematics. For a rotation around the x-axisb the rotation matrix, R_x, has the following form

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix} \tag{3.1}$$

Similarly for rotations around the y- and z-axis the rotation matrices are given as

$$R_y(\psi) = \begin{pmatrix} \cos(\psi) & 0 & \sin(\psi) \\ 0 & 1 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) \end{pmatrix} \tag{3.2}$$

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

Except the rotations also several translations have to be carried out in order to describe the positions of the desired points. An elegant way of carrying out these combinations of rotations and translations is to use homogeneous coordinates. We use this method to describe the direct kinematics. A translation- and a rotation-procedure is written which we use several times.

$$\begin{aligned} T_{rot}(\phi,\psi,\theta) = & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \\ & \begin{pmatrix} \cos(\psi) & 0 & \sin(\psi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{3.4}$$

$$T_{trans}(x,y,z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

With these homogeneous coordinates it is easy to describe all the desired positions. In the rotation procedure only the angles have to filled in, while by the translation procedure the distance, over which has to be translated, must be filled in. The calculation for the center of mass of the right leg, the so-called *Oberschenkelr* (3.6) will be shown here. It is obvious that these calculations can be carried out in a similar way for the other points.

$$\begin{aligned} Oberschenkelr = & T_{trans}(x_1(t), y_1(t), z_1(t)) * T_{rot}(\phi_0, 0, 0) * T_{rot}(0, \psi_0, 0) \\ & * T_{rot}(0, 0, \theta_0) * T_{trans}(1/2 * Beinabstand, 0, -l_0/2) * T_{rot}(0, 0, q_3) \\ & * T_{rot}(0, q_2, 0) * T_{rot}(q_1, 0, 0) * T_{trans}(0, 0, -a_2) * Start \end{aligned} \tag{3.6}$$

Where *Start* represents the zero-vector. So *Start* is a reference point.

$$Start = (0,0,0,1)^T \tag{3.7}$$

The direct kinematics will be used in the equation of Lagrange in order to calculate the dynamic model (section 4.1).

3.2 Inverse kinematics

We now have the positions for each center of mass as a function of the generalised coordinates. But we are also interested in the generalised coordinates as a function of the positions of the centers of mass, because we will use prescribed trajectories. So the position of the feet will be prescribed (subsection 5.2.2). It is then necessary to calculate the generalised coordinates in order to calculate the orientation of the robot. Therefore we are interested in this function

$$q = f^{-1}(x) \tag{3.8}$$

This is called the inverse kinematics. This inverse kinematics is developed for many robots. An example for an industrial robot can be found in [HBK00]. BARt-UH II is however quite different from ordinary industrial robots. The difference is that ordinary industrial robots only possess two degrees of freedom at the base, while BARt-UH II possesses three degrees of freedom in the hip. Subsequently an approach which is quite different from the direct kinematics is followed. We assume that the position and the orientation of the feet is known. This is indeed the case because these trajectories will be prescribed later. Furthermore the positions of the additional points in the hip are known. It is then possible to calculate all the angles, because there is only one way how this orientation of the foot can be obtained, since BARt-UH II can only walk in the human-like way.

The calculation of the inverse kinematics starts in the virtual contact point. The position of the center of mass of the foot can be calculated then in the way as the direct kinematics was developed so by means of homogeneous coordinates. Subsequently it is possible to calculate the angles of the foot. All calculations are presented here for only the right leg. It is obvious that for the left leg the calculations can be carried out in a similar way.

$$\phi_r = arctan(-\frac{Tfussr_{23}}{Tfussr_{33}}) \qquad \psi_r = arcsin(Tfussr_{13}) \qquad \theta_r = arctan(-\frac{Tfussr_{12}}{Tfuss_{11}}) \tag{3.9}$$

In these formulas $Tfussr$ is the orientation matrix of the right foot and ϕ_r and ψ_r are the rotations around the x- and y-axis. In the ankle it is assumed that first will be rotated around the x-axis and then around the y-axis, so here we have Cardan-angles. Then it is possible to calculate the angles q_5 and q_6 by means of the following formulas,

$$q_5 = \pi/2 - \arccos(-PoKnoechel_2 \frac{l_3^2 - l_2^2 + l^2}{2l_3l^2} + \sqrt{(l - \frac{PoKnoechel_2^2}{l^2}(l - (l_3^2 + l^2)^2)}{(2l_3l)^2}) - \phi$$

$$q_6 = \arctan\frac{PoKnoechel_1}{PoKnoechel_3} - \psi \tag{3.10}$$

$$l = \sqrt{PoKnoechel_1^2 + PoKnoechel_2^2 + PoKnoechel_3^2} \tag{3.11}$$

PoKnoechel is the position of the ankle. It is assumed that for the inverse kinematics the hip-positions are zero, so it is possible to calculate the angles with the given equations. By means of the cosine-rule it is now possible to calculate the angle q_4.

$$q_4 = \pi + \arccos(\frac{l_3^2 + l_2^2 - l^2}{2l_2l_3}) \tag{3.12}$$

Furthermore the assumption is made that the torso stays flat all the time. Therefore the rotations of the torso, ϕ_0, ψ_0 and θ_0, which are the rotations around the x-, y- and z-axis respectively, are equal to zero. It is then possible to calculate the angles q_2 and q_3 by setting the sum of the angles equal to zero.

$$q_3 = -q_4 - q_5 - \phi_r, \qquad q_2 = -q_6 + \psi_r \tag{3.13}$$

No formula for q_1 is derived because we assume that if this rotation is needed, this rotation is carried out in one time. So BARt-UH II walks and then turns around and walks straight on again. If however it is not possible to do so and a formula is needed, the angle can easily be calculated by setting the sum of the angles around the z-axis equal to zero.

In practice however it will be very difficult to keep the torso flat all the time. Therefore the formulas for the rotation-angles of the torso still have to be developed. It is however possible to calculate the rotations around the y- and z-axis from the positions of both hips.

Chapter 4

Dynamic Model

The kinematic model is developed in the previous chapter. Now the dynamic model is derived in this chapter. A good dynamic model is absolutely necessary for the understanding of BARt-UH II. This is not only crucial for the control of the robot, but it is also very important to have a good understanding of the torques and forces which are involved.

4.1 Equation of Lagrange

A dynamic model can be developed in various ways. We chose to use the equation of Lagrange. The Lagrangian approach is quite efficient for deriving the equations of motion for a system. All the equations are derived from two scalar functions, the kinetic energy, T, and the potential energy, V, as well as the virtual work, associated with the generalised forces. In this method the number of equations is equal to the number of independent generalised coordinates. This method however does not give any information about the constraint forces of the system [BdK01]. We assume that no friction is present in the joints.

$$\frac{d}{dt}\left(T_{,\underline{\dot{q}}}\right) - T_{,\underline{q}} + V_{,\underline{q}} = \left(Q^{nc}\right)^T \tag{4.1}$$

In equation (4.1) $\underline{q}$ is the column with the twelve generalised coordinates, $\underline{\dot{q}}$ are the generalised velocities and Q^{nc} are the torques. By introducing the Lagrangian, defined as the difference between the kinetic energy, T, and the potential energy, V, a more compact form is obtained.

$$L(\underline{q}, \underline{\dot{q}}, t) = T(\underline{q}, \underline{\dot{q}}, t) - V(\underline{q}, t) \tag{4.2}$$

Since by definition the potential energy does not depend on the general velocities (4.2) leads to

$$L_{,\underline{\dot{q}}} = T_{,\underline{\dot{q}}} \tag{4.3}$$

Combining these results leads to the more compact form of the Lagrange equation.

$$\frac{d}{dt}\left(L_{,\underline{\dot{q}}}\right) - L_{,\underline{q}} = \left(Q^{nc}\right)^T \tag{4.4}$$

The kinetic energy of each body consists of two parts. The first part is a result of the translation of the body. The second contribution comes from the rotation of each body. Because each body can rotate around three axis, the moments of inertia in three directions should be taken into account.

$$T = \frac{1}{2} * \sum_{i=1}^{7} m_i \dot{\vec{r}}_i \cdot \dot{\vec{r}}_i + \frac{1}{2} * \sum_{i=1}^{7} J_{xi}\dot{\omega}_{xi} \cdot \dot{\omega}_{xi} + J_{yi}\dot{\omega}_{yi} \cdot \dot{\omega}_{yi} + J_{zi}\dot{\omega}_{zi} \cdot \dot{\omega}_{zi} \tag{4.5}$$

The only contribution to the potential energy comes from the gravity force, because there are no elastic elements within the body and hence the internal energy function $U_{in}(\underline{q})$ is zero.

$$V = \sum_{i=1}^{7} m_i \cdot g \cdot z_i \tag{4.6}$$

In this equation z_i represents the vertical coordinate of a body. By substituting the formulas for the direct kinematics into equations (4.5) and (4.6) and calculating the equation of Lagrange the equations of motion for BARt-UH II are obtained. However, the contact forces between the foot and the ground are not taken into account.

4.2 Contact forces

As can be seen in section 4.1 the equation of Lagrange does not take kinematic constraints into account. Therefore the contact forces between the ground and the feet of the robot are not taken into account. As can be seen in section 2.2 force sensors have been integrated in the feet to measure the contact forces. So it will be very interesting to ensure that the contact forces will be taken into account. A closer look will be taken at two different methods to model these contact forces.

4.2.1 Modelling the contacts forces

We assume perfect rigidity between the feet and the floor. This can however lead to systems with no or several solutions. This is indeed the case when the foot stands flat on the ground. As can be seen in figure 2.8 four contact points between the foot and the floor are present when the foot stands flat on the ground. So twelve reaction forces have to be calculated then, F_x, F_y and F_z for each contact point. Only three forces and three moments are led through the ankle. Indeed it is an undetermined system. This undeterminancy can be avoided by adding some compliance to the system, actually meaning that the assumption of rigid bodies does not longer hold for the contact. In this method the ground will be modelled as a spring-dashdot element [Bro99]

$$F_n = K_f \delta + D_f \dot{\delta} \tag{4.7}$$

Here F_n is the ground normal force, δ is the penetration depth and K_f and D_f are the stiffness coefficient and the damping coefficient, respectively. During the walking cycle however the number of contact points is not constant.This method leads to a set of stiff ordinary differential equations, which are very expensive to integrate. Another main defect is the ground normal force may be negative because $\dot{\delta}$ easily becomes negative. Physically this is not correct because it implies that the ground pulls the contact foot. It is however possible with a critical damped system to ensure that this is not the case.

A second way to take the contact forces into account is to formulate the problem as a Non-Linear Complementary Problem. This leads however to much more difficult mathematics. Examples for the two dimensional case can be found in [LRvCD02]. The three dimensional case can be found in [Glo01].

4.3 Calculations in Maple

All the symbolic computations are carried out in Maple V, a symbolic computer algebra system. However some problems occur which make it necessary to calculate some elements differently then it is expected. For a robot it is convenient to write the equations in the following way where M is the mass-matrix and CG the coriolis-vector.

$$Q = M(\underline{q})\ddot{\underline{q}} + CG(\underline{q}, \dot{\underline{q}}) \tag{4.8}$$

Maple generates the dynamic model as a set of equations. It is then possible by means of the command *genmatrix* to compute a matrix out of a set of equations. This command is therefore used to calculate the mass-matrix. But it turns out that some rows and columns are interchanged. So the command does not function correctly. Hence a procedure is written to calculate the mass-matrix. Within this procedure each element of the mass-matrix has to be calculated separately. This is done with the command *coeff* which extracts the coefficients of a polynomial. It is possible to use this command because each equation can be looked upon as a polynomial in $\ddot{q}$ and q After all the matrices and vectors are calculated, it is necessary to export them as an optimized C-function, because the simulation tool which is build needs the kinetic and kinematic in this language. Maple is able to export vectors as an optimized C-function. However due to the size and the complexity of the coriolis-vector, this vector has to be splitted in two parts, before it can be exported. The Maple-file which we use to develop both models and to export the equations into an optimized C-function can be found in appendix A.

4.4 Numerical computations

Another problem which occurrs, is that the inverse of the mass-matrix, M^{-1}, can not be calculated in Maple symbolic, due to the complexity. It is therefore necessary to calculate it numerically. We want to calculate the inverse of the mass-matrix because the accelerations have our interest.

$$\ddot{q} = M(\underline{q})^{-1}(Q - CG(\underline{q}, \dot{\underline{q}})) \tag{4.9}$$

It can be seen from (4.9) why the inverse of the mass-matrix has to be calculated. Furthermore it can be seen that the set of equations is equivalent to a standard linear set of equations.

$$Ax = b \tag{4.10}$$

While a mass-matrix is per definition symmetric, $M_{ij} = M_{ji}$ and positive definite, $x \cdot M \cdot x > 0$ for $x \neq 0$ the Cholesky Decomposition can be used. Instead of seeking arbitrary lower and upper triangular factors, L and U, the Cholesky Decomposition constructs a lower triangular matrix, L, whose transpose, L^T itself can serve as the upper triangular part. In other words we can say

$$L(\underline{q}) \cdot L(\underline{q})^T = M(\underline{q}) \tag{4.11}$$

The advantage of using the Cholesky Decomposition is the reduction of the calculating time. The Cholesky Decomposition is a factor of two faster than other methods for solving linear equations. Because for each step the inverse of the mass-matrix has to be calculated again it is very important that the calculation time is reduced to a minimum.

Furthermore it is necessary to develop an integrator. So therefore a fourth order Runga Kutta formula is developed. This is not standard available while it has to be developed in Delphi. A fourth order Runga Kutta formula uses a parabola to approximate the derivative.

$$\frac{dq}{dt} = f(q, t) \tag{4.12}$$

$$\begin{aligned} k_1 &= hf(t_n, q_n) \\ k_2 &= hf(t_n + \frac{h}{2}, q_n + \frac{h}{2}) \\ k_3 &= hf(t_n + \frac{h}{2}, q_n + \frac{k_2}{2}) \\ k_4 &= hf(t_n + h, q_n + k_3) \end{aligned} \tag{4.13}$$

The solutions to the differential equation (4.12) is given iteratively by running through (4.13) given the point t_n and q_n and the step size h between t_n and t_{n+1}. [Hea97]. With the solution of these equations it is possible to calculate q_{n+1}.

$$t_{n+1} = t_n + h \tag{4.14}$$

$$q_{n+1} = q_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.15}$$

This integration has to be carried out twice. After the first integration the angular velocities are obtained. After the second integration the orientation of the whole robot is known.

Chapter 5

The development and the application of the simulation tool

The dynamic as well as the kinematic model has been developed in the previous chapters. It is now important to run several simulations to validate the models. It is therefore necessary to develop a simulation tool to test and visualise several algorithms for BARt-UH II.

5.1 Development of the test application

In order to validate the created models it is necessary to develop a test application, because no standard simulator is available, which can also visualise the walking of BARt-UH II. As is clear from the previous chapters different tools are used to calculate different parts of this simulation. It is therefore necessary to create the application tool in such a way that all these tools can cooperate in a correct way.

Within the test application the following tools are used.

- Maple
- C
- Delphi
- OpenGL

Maple is used to generate the equations of motion. It exports the equations of motion to an optimized C-function. Before this can be carried out it is necessary to find a way to write down the generalised coordinates, their derivatives and the initial conditions. Other points of interest are the positions of the centers of mass of the torso and both feet. This method has to be well-defined so that it can be used throughout the whole test application. Furthermore everything has to be indexed. Hence for the generalised coordinates the matrix, q is created, where each column represents the corresponding generalised coordinates. So the first row contains the generalised coordinates, the second row contains the angular velocities and the third row the accelerations.

$$\begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,12} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,12} \\ q_{3,2} & q_{3,2} & \cdots & q_{3,12} \end{pmatrix}$$

For the point, r_1, the center of mass of the torso, the matrix $Torso$ is developed.

$$\begin{pmatrix} x_1 & y_1 & z_1 & \phi_1 & \psi_1 & \theta_1 \\ \dot{x_1} & \dot{y_1} & \dot{z_1} & \dot{\phi_1} & \dot{\psi_1} & \dot{\theta_1} \\ \ddot{x_1} & \ddot{y_1} & \ddot{z_1} & \ddot{\phi_1} & \ddot{\psi_1} & \ddot{\theta_1} \end{pmatrix}$$

So x_1 is exported as $Torso_{11}$. For the right and the left foot two matrices with the same structure are created, respectively *Rechts* and *Links*.

The second program which is used in the test application is C++. As already mentioned, the equations of motion are imported into this program. Furthermore the Cholesky Decomposition is carried out here. By means of so-called dll-files the link is made towards Delphi, the third program. Within Delphi the actual visualization is made. For this visualization OpenGL is used, which is a graphical interface that can be used for creating computer generated images and interactive applications, both in 2D as 3D. Not only the robot is build up in Delphi but also the camera positions etc. are created here. The robot is created according to the real dimensions. The foot is not totally created as the real foot, because that would be too complicated. However the important features, like the dimensions and the number of contact points is the same. Also the Runga Kutta Fourth Integrator is developed in Delphi. A last part that is created in Delphi is the path-planning, which is essential to let BARt-UH II walk.

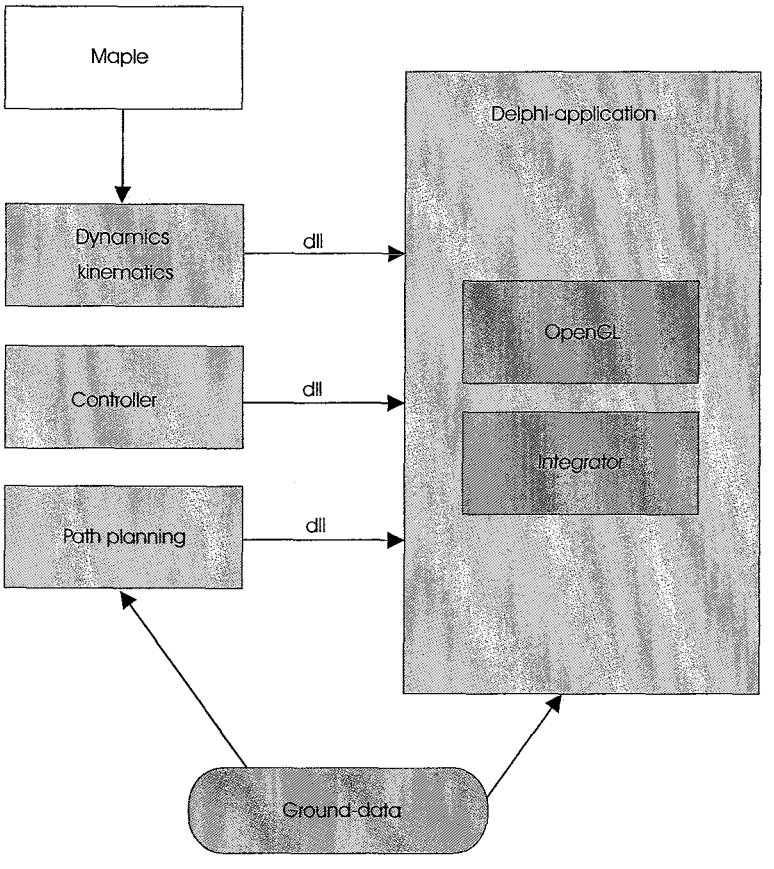

Figure 5.1: The lay-out of the application tool

The only thing left that has to be created now are the boundary conditions. A result from these boundary conditions is the ground on which BARt-UH II walks. This is actually nothing more than a set of data. These data are interconnected with the visualizer and the path-planning figure 5.1. From figure 5.1 it can be seen that everything is a different block. The advantage of these settings is that with means of a TCP/IP connection it is possible to compute the path-planning, the kinematics and the dynamics on an external computer.

Within the test application three modes are available:

- kinematic visualisation
- dynamic emulation

- dynamic simulation

The first mode, the kinematic visualisation can be used to validate the kinematic model or a trajectory. It can return the rotations of the different angles and their derivatives by prescribing a certain movement, but it can also return the position of a certain point when an angle has been prescribed. The dynamic emulation can be used to calculate the moments which are present within BARt-UH II. So within this mode the test application is able to calculate all the moments in the hip, knee and ankle. Within the last mode, the dynamic simulation, it is possible to implement a controller for the robot. So it is therefore possible to visualise the difference between the actual path and the prescribed path.

5.2 Simulations

Now that the test application has been build, it is possible to carry out some simulations in order to verify the models, so that we get a good impression whether the model returns correct values when for instance a periodic motion is prescribed. The motion can be calculated without looking at the model but by just calculating the motion for one angle. By comparing this with the visualisation and the results of the simulator and the model a good impression about the correctness of the kinematic model is obtained. In order to do so we use the kinematic visualisation and the dynamic emulation. In the dynamic emulation we implement a controller and we take a look at the results when using this controller.

5.2.1 Kinematic visualisation

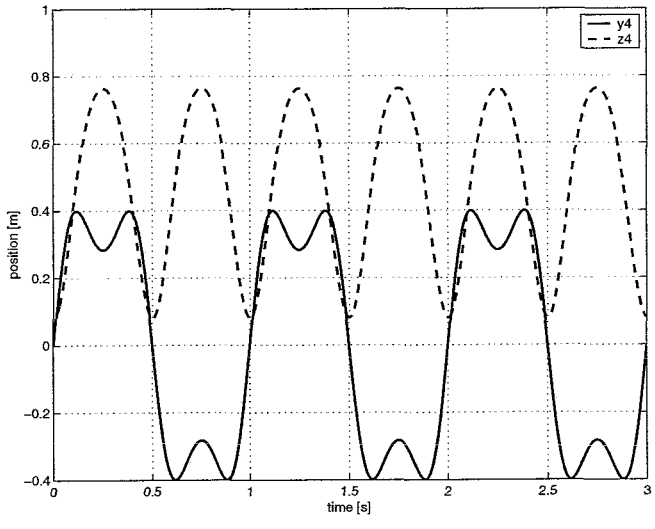

Figure 5.2: The position for the center of mass of the right foot

The first simulation is for the kinematic model. This simulation is carried out by prescribing a periodic motion for each angle. It is then possible by means of the direct kinematics to calculate the position of the centers of mass. This is carried out for every angle. It must be noticed that for the real robot a complete sine for q_4 and q_{10} will be impossible because BARt-UH II can only walk in the human-like way. But it is a good simulation to validate the kinematic model, because the model becomes less much difficult. By calculating some important points, e.g. when the prescribed angle is equal to $\pi/2$, it is possible to get a good impression whether the model is correct or not. Especially when it is possible to compare the calculated equations with the visualiser. Figure 5.2 shows the result when angle q_4 is a sine-function.

$$q_4 = \frac{3\pi}{4} \cdot sin(2\pi t) \tag{5.1}$$

The periodic motion can be distinguished very well in figure 5.2. It is important to have a good understanding of the kinematics. By filling in the dimensions of BARt-UH II the position of the center of mass of the right foot, x_4, y_4 and z_4, can be calculated.

$$x_4 = 0.1 \qquad y_4 = 0.4 * sin(q_4) \qquad z_4 = 0.45 - 04 * cos(q_4) \tag{5.2}$$

These formula return the same values what would be expected by just examining the visualiser. As can be seen from figure 5.3 the y-position has a maximum value when $q_4 = \pi/2$.

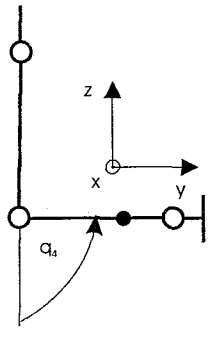

Figure 5.3: The orientation of the lower leg for $q_4 = \pi/2$

This maximum is obtained when $\sin(2\pi t) = 2/3$ or $t \approx 0.12$ (figure (5.2)). It is also clear from (5.1) that q_4 has its maximum when $\sin(2\pi t) = 1$, so when $t = 0.25$ s. So after y_4 reaches its maximum its value should become smaller again. This can be distinguished in figure 5.2. It can also be calculated from (5.2). Furthermore it is possible to calculate the velocities and accelerations in the test application. Figure 5.4 shows the velocities in the y- and z-direction. It is obvious that the velocity $\dot{y}_4$ has its minimum when the y-position is equal to zero. Furthermore can be distinguished by comparing figure 5.2 and 5.4 that $\dot{z}_4$ is equal to zero when z_4 reaches a minimum or a maximum.

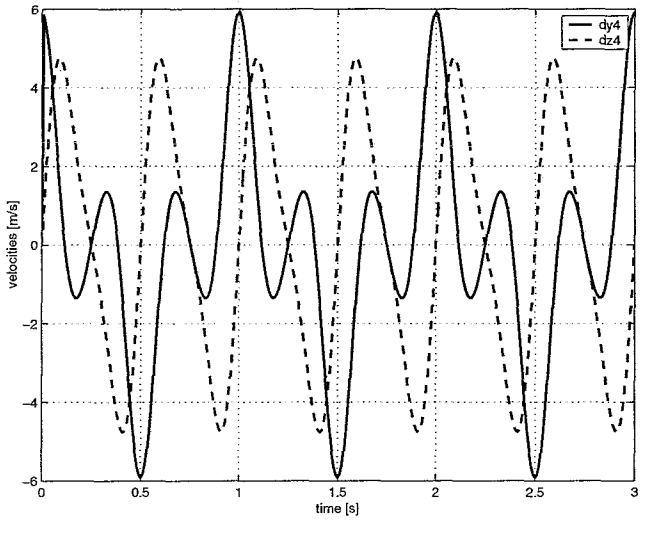

Figure 5.4: The velocities of the center of mass of the foot

The same simulation is carried out for q_2. Again the periodic character can be distinguished. It is obvious that the x-position x_4 shows the same behaviour as y_4 in the first simulation.

$$q_2 = \frac{3\pi}{4} \cdot sin(2\pi t) \tag{5.3}$$

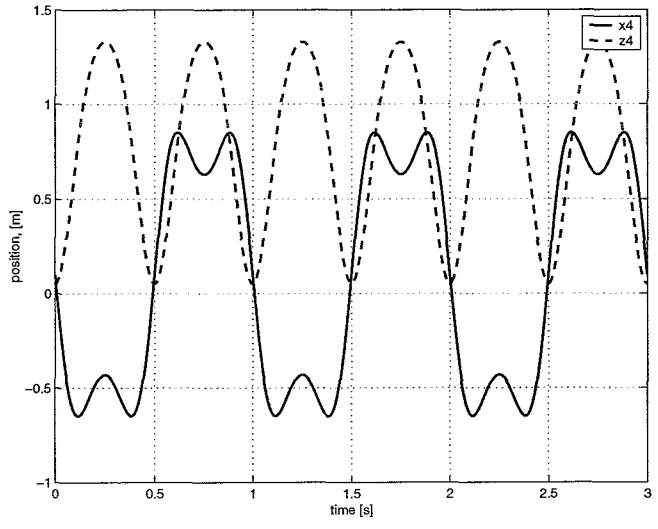

Figure 5.5: The position for the center of mass for the right foot

Similar results can be obtained for the other angles, except for q_1 respectively q_7. Because when the leg is totally stretched, the centers of mass will not change their positions because they were assumed to be at the middle of the cylinder. This simulation can however be carried out by taking a certain position for the robot, in which the leg is not completely stretched. A possible orientation for the robot to carry out a periodic motion is for $q_4 = -\pi/2$. Figure 5.6 shows the results for this orientation when q_1 is prescribed.

$$q_1 = \frac{3\pi}{4} \cdot sin(2\pi t) \tag{5.4}$$

All this simulations, including those for the other angles, can be found on the CD-rom, which is attached to this document.

The second simulation that has to be carried out is for the inverse kinematics. This can be done in a similar way. However this time a periodic motion is prescribed for a position. But before this is carried out first a certain position is prescribed. It is then possible with the test application to visualise the motion of BARt-UH II. This simulation is carried out again for all angles.

$$z_2 = \frac{3\pi}{4} \cdot sin(2\pi t) \tag{5.5}$$

There is however a second way to validate the inverse kinematics. That can be done by letting BARt-UH II walk. This is carried out in the following way. The torso is kept flat all the time, so it is not allowed to rotate. Therefore its position of the center of mass only changes in the y-direction (figure 5.7).

Furthermore the feet are also assumed to be flat. In that way BARt-UH II was able to walk. So it is necessary to further develop the walking in order to take the influence of the torso into account. Furthermore it is important to carry out simulations, in which the feet do not keep flat all the time. As we saw earlier BARt-UH II is able to walk in that way. On the CD-Rom it is possible to take a closer look at the results of the simulations.

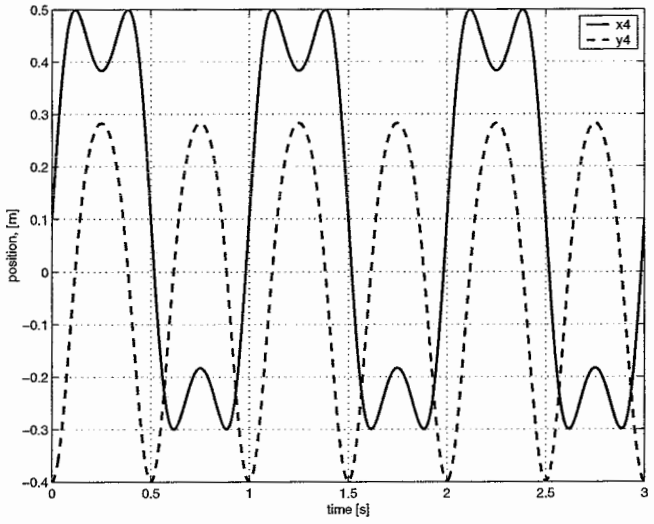

Figure 5.6: The position of the center of mass for the right foot

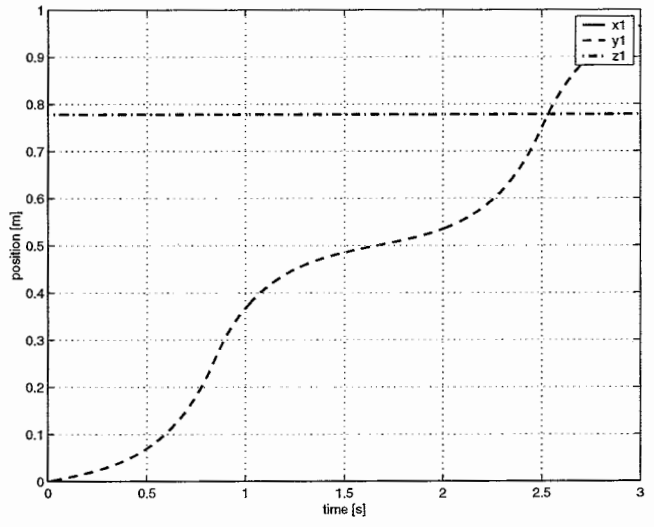

Figure 5.7: The motion of the Torso

5.2.2 Dynamic emulation

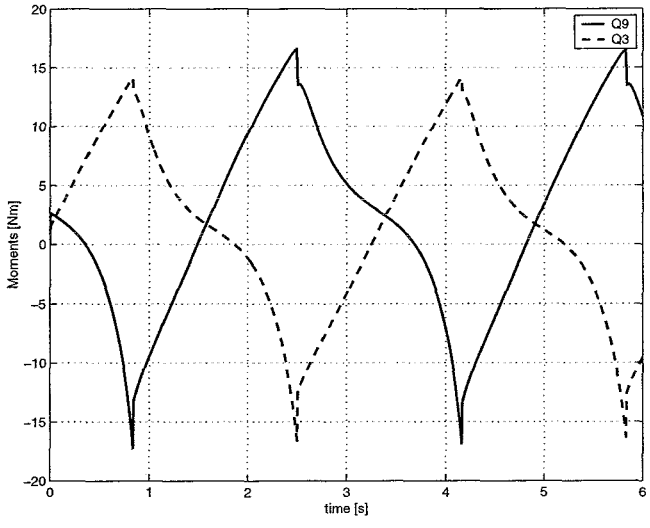

Figure 5.8: The moments in the hip around the x-axis

It is also necessary to carry out simulations for the dynamic model. These are actually carried out in the same way as the simulation for the inverse kinematics as BARt-UH II walks in the same way. Again the torso and the feet are not allowed to rotate. It is clear that the moments which are calculated do not have the correct value when walking like this, due to the facts that the mass and the moments of inertia of the torso are not taken into account. It can however give a good idea how the moments vary during time. Figure 5.8 shows the moments in the hips for the rotations around the x-axis, q_3 respectively q_9 when BARt-UH II walks in the same way as in the kinematic visualisation. Here, Q_3 is the moment around the x-axis in the right hip and Q_9 the corresponding moment in the left leg. It can be seen very well that both moments have approximately the same values, but they are not totally equal as is expected. There can be two reasons for this. The first one can be numerical errors. However the difference is too big for numerical approximations only. Therefore the second reason can be a reasonable explanation. It can be that the motion is not totally symmetric since the initial velocities are not the same for both legs, while there is a jump in the velocities and accelerations at $t = 0$ s. It is therefore interesting to take a closer look at some more moments.

Figure 5.9 shows the moments around the y-axis in the hip. Q_2 and Q_8 are the moments in the right and the left leg respectively. It also shows the moments around the x-axis in both ankles, Q_5 in the right ankle, Q_{11} in the left ankle. As can be seen there is again a difference in the absolute values. This difference does not become less when the simulation is carried out longer.

Furthermore it is convenient to look at the angular velocities and accelerations in the hip. Figure 5.10 shows the angular velocities and accelerations for the angles q_3 and q_9. It shows that there are indeed some inequalities, especially in the angular accelerations. So besides the numerical error the motion is probably not totally symmetric either.

The results for the other moments can be found on the attached CD-rom. It is then also possible to calculate the moments when a periodic motion is prescribed.

5.2.3 Dynamic simulation

The last mode in the test-application is the dynamic simulation. Within this mode it is possible to develop a controller. A PD-controller is used to control BARt-UH II. The aim in this simulation is to let BARt-UH II walk after the implementation of the PD-controller. Because it is not the

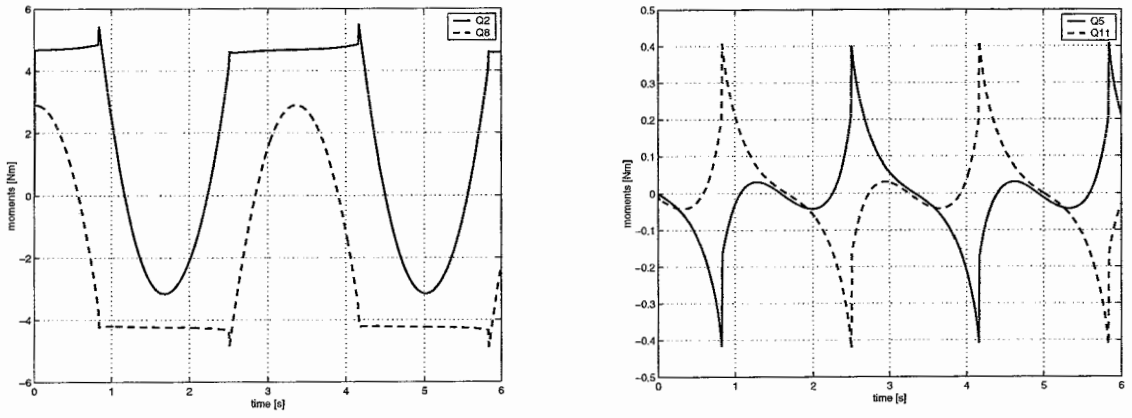

Figure 5.9: The moments in the hip around the y-axis (left) and the moments in the ankle around the x-axis (right)

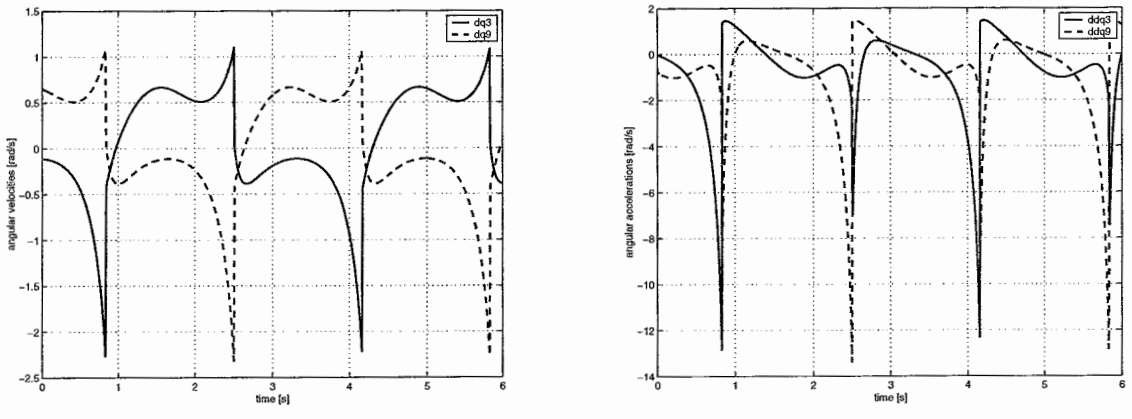

Figure 5.10: The angular velocities and the angular accelerations in the hip around the x-axis

aim to tune a controller, not all the controllers have been tuned. We focus on two controllers just to watch how the test application works when a controller is developed. These are the controllers for the angles, q_3 and q_4. The controllers are built in the following way. The trajectory is built of many points. The trajectory is controlled for every point. So the angles are controlled for this point. After the next increment this point is changed. Hence the controller adjusts the angles.

Figure 5.11 shows not only the the actual angles, but we can also see the desired angles, $sollq_3$ and $sollq_4$. It can be distinguished very well that some problems occur around the points where the angles have their maximum value. One reason for these problems can be instability. It is quite clear that the problems occur when the angle reaches its maximal value because the velocity changes sign around this point.

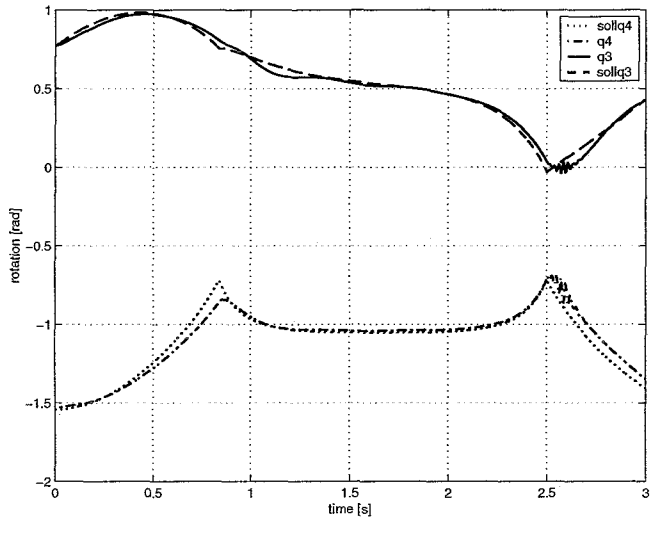

Figure 5.11: The rotations around the x-axis in the hip and the knee

One other reason for the instability can be the numerical approximation. Due to the fact that two integrators are used and that the velocity is almost zero this can play a part. So probably it is a combination of the instability and numerical approximation. On the CD-Rom the corresponding simulation can be found. It is then also possible to take a look at the other angles. Furthermore it is possible to tune the controllers for the other angles.

Chapter 6

Conclusion

6.1 Conclusions

It is clear from the previous chapters that the aim is to develop the kinematic and dynamic model for BARt-UH II, a bipedal robot with twelve degrees of freedom. Each leg possesses six degrees of freedom, three in the hip, one in the knee and two in the ankle. First of all a model of the robot is derived. The angles are numbered and a coordinate frame is developed. Furthermore the important characteristics for each body has been introduced.

The first model which is developed is the kinematic model. Both the direct and the inverse kinematics are developed. We notice that everything is still in robot coordinates. So it is still necessary to transform the kinematic model into world coordinates. Beside the center of mass for each body also the equations for an additional point in each hip and the virtual contact points are developed.

After the development of the kinematic model the dynamic model is developed. This is carried out by means of the Lagrange equation. The kinematic constraints are however not taken into account yet, so the contact forces between the ground and the feet are not taken into account. However some methods are investigated how this can be carried out. Furthermore the mass and the moments of inertia of the torso are not taken into account because the kinematic model is still in robot coordinates.

Besides the development of both models also a test application is developed to carry out simulations for BARt-UH II, because no standard simulation tool is available. Within this test application it possible to make use of three modes, the kinematic visualisation, the dynamic emulation and the dynamic simulation. With these modes it is possible to test and visualise several algorithms for BARt-UH II.

Several simulations are carried out. In these simulations BARt-UH II is able to walk in in this simulations. The torso and the feet keep flat all the time, so they are not allowed to rotate.

6.2 Recommendations

For further research the following items should be taken into account. First of all the contact forces have to be implemented in the dynamic model. The best way to carry out this step is by adding some compliance to the system. Furthermore it is necessary to transform the robot coordinates into world coordinates so that the influence of the torso is taken into account. When this transformation is carried out it will be very interesting to further develop the simulations within the dynamic emulation, implying that the torso and the feet do not keep flat all the time. In doing so a better understanding of the moments in the hip, ankle and knee will be available. Furthermore it will be necessary to add some compliance to the system. This must be done because the real robot certainly possesses some compliance in the joints. Furthermore the model still needs to be validated experimentally. At the moment this is not possible yet because BARt-UH II is not

totally developed yet. A last recommendation is to develop the rotor dynamics in order to know which voltage is needed. When all this recommendations are carried out many more issues about BARt-UH II will be known.

Bibliography

[BdK01] Dick H van Campen Bram de Kraker. *Mechanical Vibrations.* Shaker Publishing, 2001. ISBN 90-423-0165-1.

[Bro99] Bernard Brogliato. *Nonsmooth Mechanics.* Springer-Verlag London, 2nd edition, 1999. ISBN 1-85233-143-7.

[Ger01] Marc Gerecke. Entwicklung und Aufbau der Fussfläche für einen zweibeinigen Roboter mit 12 Freiheitsgraden. Master's thesis, Institut für Regelungstechnik, Universität Hannover, 2001.

[Glo01] Christoph Glocker. *Set-Valued Force Laws Dynamics of Non-Smooth Systems*, volume 1 of *Lecture Notes in Applied Mechanics.* Springer, 2001. ISBN 3-540-42436-3.

[HBK00] Gerth W. Heimann B. and Popp K. *Mechatronik, Komponenten, Methoden, Beispeile.* Fachbuchverlag Leipzig, 2nd edition, 2000. ISBN 3-446-21711-8.

[Hea97] Michael T Heath. *Scientific Computing: An Introductory Suvey.* McGraw-Hill, 1997. ISBN 0-07-027684-6.

[JH99] Olaf Schermeier Jens Hofschulte. Entwurf, Aufbau und Inbetriebnahme eines bipedalen Roboters. Master's thesis, Institut für Regelungstechnik, Universität Hannover, 1999.

[LRvCD02] Glocker Ch. Leine R.I. and van Campen D.H. Nonlineair dynamics and modeling of some wooden toys with impact and friction. *Journal of Vibration and Control*, 2002.

Appendix A

Maple code

```
=====================================================
 Traineeship of Wout Schwanen
 University of Technology Eindhoven, Department Mechanical Engineering 2002
 Universitt Hannover, Institut fr Regelungstechnik 2002
 Email: W.Schwanen@student.tue.nl
=====================================================

 Matemathical model of a Bipedal Autonomous Robot

> restart:with(linalg):with(codegen):readlib(algsubs):
 Konstanten
Description of the kinematics etc.
Kinematics
 direct kinematics
Definition der Transformationsmatrix-Prozedur
> Trot:=proc(phi,psi,theta)
>  evalm(
>   matrix(4,4,[1,0,0,0,0,cos(phi), -sin(phi),0,0,sin(phi), cos(phi),0,0,0,0,1])&*
>   matrix(4,4,[cos(psi),0,sin(psi),0,0,1,0,0,-sin(psi),0,cos(psi),0,0,0,0,1])&*
>   matrix(4,4,[cos(theta),-sin(theta),0,0,sin(theta),cos(theta),0,0,0,0,1,0,0,0,0,1])):
> end:
> Ttrans:=proc(x,y,z)
>   matrix(4,4,[1,0,0,x,0,1,0,y,0,0,1,z,0,0,0,1]):
> end:
> Start         :=vector(4,[0,0,0,1]):
> TTorso        :=evalm(Ttrans(x1(t),y1(t),z1(t))&*Trot(phi0,0,0)&*Trot(0,psi0,0)&*
                      Trot(0,0,theta0)):
> Torso         :=evalm(TTorso&*Start):
> THuefter      :=evalm(TTorso&*Ttrans(1/2*Beinabstand,0,-l0/2)):
> Huefter       :=evalm(THuefter&*Start):
> TOberschenkelr:=evalm(THuefter&*Trot(0,0,q1(t))&*Trot(0,q2(t),0)&*Trot(q3(t),0,0)&*
                        Ttrans(0,0,-a2)):
> Oberschenkelr :=evalm(> TOberschenkelr&*Start):
> Tknier        :=evalm(THuefter&*Trot(0,0,q1(t))&*Trot(0,q2(t),0)&*Trot(q3(t),0,0)
                        &*Ttrans(0,0,-l2)):
> Unterschenkelr:=evalm(Tknier&*Trot(q4(t),0,0)&*Ttrans(0,0,-a3)&*Start):
> TKnochelr     :=evalm(Tknier&*Trot(q4(t),0,0)&*Ttrans(0,0,-l3)):
> Fussr         :=evalm(TKnochelr&*Trot(q5(t),0,0)&*Trot(0,q6(t),0)&*Ttrans(0,0,-a4)&*
                        Start):
```

```
> Grundpunkt    :=evalm(TKnochel&*Trot(q5(t),0,0)&*Trot(0,q6(t),0)&*Ttrans(0,0,-14)&*
                        Trot(0,psir,0)&*Trot(0,0,thetar)&*Trot(phir,0,0)&*Start):
> THueftel      :=evalm(TTorso&*Ttrans(-1/2*Beinabstand,0,-l0/2)):
> Hueftel       :=evalm(THueftel&*Start):
> TOberschenkell:=evalm(THueftel&*Trot(0,0,q7(t))&*Trot(0,q8(t),0)&*Trot(q9(t),0,0)&*
                        Ttrans(0,0,-a2)):
> Oberschenkell :=evalm(TOberschenkell&*Start):
> Tkniel        :=evalm(THueftel&*Trot(0,0,q7(t))&*Trot(0,q8(t),0)&*Trot(q9(t),0,0)&*
                        Ttrans(0,0,-l2)):
> Unterschenkell:=evalm(Tkniel&*Trot(q10(t),0,0)&*Ttrans(0,0,-a3)&*Start):
> TKnochell     :=evalm(Tkniel&*Trot(q10(t),0,0)&*Ttrans(0,0,-l3)):
> Fussl         :=evalm(TKnochell&*Trot(q11(t),0,0)&*Trot(0,q12(t),0)&*Ttrans(0,0,-a4)&*
                        Start):
> Grundpunktl   :=evalm(TKnochel&*Trot(q5(t),0,0)&*Trot(0,q6(t),0)&*Ttrans(0,0,-14)&*
                        Trot(0,psil,0)&*Trot(0,0,thetal)&*Trot(phil,0,0)&*Start):
> dirKin:=[ x1(t)=evalm(Torso[1]),
>           y1(t)=evalm(Torso[2]),
>           z1(t)=evalm(Torso[3]),
>           x2(t)=evalm(Oberschenkelr[1]),
>           y2(t)=evalm(Oberschenkelr[2]),
>           z2(t)=evalm(Oberschenkelr[3]),
>           x3(t)=evalm(Unterschenkelr[1]),
>           y3(t)=evalm(Unterschenkelr[2]),
>           z3(t)=evalm(Unterschenkelr[3]),
>           x4(t)=evalm(Fussr[1]),
>           y4(t)=evalm(Fussr[2]),
>           z4(t)=evalm(Fussr[3]),
>           x5(t)=evalm(Oberschenkelll[1]),
>           y5(t)=evalm(Oberschenkelll[2]),
>           z5(t)=evalm(Oberschenkelll[3]),
>           x6(t)=evalm(Unterschenkell[1]),
>           y6(t)=evalm(Unterschenkell[2]),
>           z6(t)=evalm(Unterschenkell[3]),
>           x7(t)=evalm(Fussl[1]),
>           y7(t)=evalm(Fussl[2]),
>           z7(t)=evalm(Fussl[3])]:
> DdirKin:=diff(dirKin,t): DDdirKin:=diff(DdirKin,t):
> dirKin:=subs(Ableitungen,dirKin):DdirKin:=subs(Ableitungen,DdirKin):
  DDdirKin:=subs(Ableitungen,DDdirKin):
inverse kinematics
> assume(l2,real);assume(l2>0);
> assume(l3,real);assume(l3>0);
> assume(l4,real);assume(l4>0);
> VKP:=vector(4,[x,y,z,1]):
> TFuss:=evalm(Ttrans(x,y,z)&*
>              Trot(phi,0,0)&*
>              Trot(0,psi,0)&*

>              Trot(0,0,theta)&*
>              Ttrans(0,0,14)):
> PoKnoechel:=evalm(TFuss&*vector(4,[0,0,0,1])):
> OrKnoechel:=vector(3,[arctan(-TFuss[2,3]/TFuss[3,3]),
>                       arcsin(TFuss[1,3]),
>                       arctan(-TFuss[1,2]/TFuss[1,1])]):
```

```
> l:=sqrt(PoKnoechel[1]^2+PoKnoechel[2]^2+PoKnoechel[3]^2):
> qFuss:={q6=arctan(PoKnoechel[1]/PoKnoechel[3]) + psi,
>         q5=pi/2-arccos(-PoKnoechel[2]*(l3^2-l2^2+l^2)/(2*l3*l^2) +
>            sqrt((1-PoKnoechel[2]^2/l^2)*(1-(l3^2-l2^2+l^2)^2/(2*l3*l)^2))) - phi,
>         q4=pi+arccos((l3^2+l2^2-l^2)/(2*l2*l3))}:
> TBein:=subs(qFuss,
>        evalm(Trot(0,-q6,0)&*
>              Trot(-q5,0,0)&*
>              Ttrans(0,0,l3)&*
>              Trot(-q4,0,0)&*
>              Ttrans(0,0,l2))):
> PoBein:=evalm(TFuss&*TBein&*vector(4,[0,0,0,1])):
> OrBein:=vector(3,[arctan(-TBein[2,3]/TBein[3,3]),
>                   arcsin(TBein[1,3]),
>                   arctan(-TBein[1,2]/TBein[1,1])]):
> qHuefte:={q3= -q4 -q5 -phi,
>           q2= -q6 +psi,
>           q1= theta}:
> iKin:=qFuss union qHuefte:
> invKin1:=evalm(subs(mitt,matrix(1,6,
>                [0,0,0,subs(iKin,q4), subs(iKin,q5), subs(iKin,q6)]))):
> invKin2:=evalm(stackmatrix(invKin1,map(x->diff(x,t),invKin1))):
> invKin3:=evalm(stackmatrix(invKin2,map(x->diff(x,t,t),invKin1))):
> invKin:=subs(toMatrix,subs(ohnet,dimensions,evalm(invKin3))):
Lagrange
Kinetic and Potential Energy
> EKin:=1/2*Jx2*(dq3)^2+1/2*Jy2*(dq2)^2+1/2*Jz2*(dq1)^2
>      +1/2*Jx3*(dq3+dq4)^2+1/2*Jy3*(dq2)^2+1/2*Jz3*(dq1)^2
>      +1/2*Jx4*(dq3+dq4+dq5)^2+1/2*Jy4*(dq2+dq6)^2+1/2*Jz4*(dq1)^2
>      +1/2*m2*((dx2)^2+(dy2)^2+(dz2)^2)
>      +1/2*m3*((dx3)^2+(dy3)^2+(dz3)^2)
>      +1/2*m4*((dx4)^2+(dy4)^2+(dz4)^2)
>
>      +1/2*Jx2*(dq9)^2+1/2*Jy2*(dq8)^2+1/2*Jz2*(dq7)^2
>      +1/2*Jx3*(dq9+dq10)^2+1/2*Jy3*(dq8)^2+1/2*Jz3*(dq7)^2
>      +1/2*Jx4*(dq9+dq10+dq11)^2+1/2*Jy4*(dq8+q12)^2+1/2*Jz4*(dq7)^2
>      +1/2*m2*((dx5)^2+(dy5)^2+(dz5)^2)
>      +1/2*m3*((dx6)^2+(dy6)^2+(dz6)^2)
>      +1/2*m4*((dx7)^2+(dy7)^2+(dz7)^2):
> EPot:=m2*g*z2+m3*g*z3+m4*g*z4
>      +m2*g*z5+m3*g*z6+m4*g*z7:
Calculating the dynamical model
> # All parts of the LaGrange equation will be calculated
> EKin:=subs(dirKin,subs(DdirKin,subs(DDdirKin,EKin))):
> EPot:=subs(dirKin,subs(DdirKin,subs(DDdirKin,EPot))):
> # Calculating the parts which involve the kinematic energy
> T:=EKin:
> dTddq:=subs(Ersetzung1,grad(T,
              vector([dq1,dq2,dq3,dq4,dq5,dq6,dq7,dq8,dq9,dq10,dq11,dq12]))):
> dTddqdt:=subs(Ersetzung2,map(x->diff(x,t),dTddq)):
> dTdq:=grad(T,vector([q1,q2,q3,q4,q5,q6,dq7,q8,q9,q10,q11,q12])):
> # Calculating the part with corresponds to the potential energy
> U:=subs(Ableitungen,EPot):
> dUdq:=vector(12):
```

```
> dUdq:=grad(U,vector([q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12])):
> Qla:=vector(12):ddq:=vector(12):
> Qla:=subs({ddq1=ddq[1], ddq2=ddq[2], ddq3=ddq[3], ddq4=ddq[4],
             ddq5=ddq[5], ddq6=ddq[6], ddq7=ddq[7], ddq8=ddq[8],
             ddq9=ddq[9], ddq10=ddq[10], ddq11=ddq[11], ddq12=ddq[12]},
             evalm(dTddqdt-dTdq+dUdq)):
> # Aufstellen der MCG-Darstellung: M(q)*ddq + cg(q,dq) = Q
> Dimensions:=12:
> Mla:=matrix(Dimensions,Dimensions):
> for i from 1 by 1 to Dimensions do
>  for j from 1 by 1 to Dimensions do
>   Mla[i,j]:=coeff(Qla[i],ddq[j]):
>  od:
> od:
> unapply('Dimensions'):
> CGla:=combine(subs({ddq[1]=0,ddq[2]=0,ddq[3]=0,ddq[4]=0,ddq[5]=0,ddq[6]=0,
                      ddq[7]=0,ddq[8]=0,ddq[9]=0,ddq[10]=0,ddq[11]=0,ddq[12]=0}, Qla),trig):
Zahlenwerte einsetzen
Ausgabe als C-Code
> q:=matrix(3,12):Torso:=matrix(3,6):rechts:=matrix(3,6):links:=matrix(3,6):
> dMatrix:={q1=q[1,1], q2=q[1,2], q3=q[1,3],q4=q[1,4], q5=q[1,5], q6=q[1,6],
>           q7=q[1,7], q8=q[1,8], q9=q[1,9],q10=q[1,10], q11=q[1,11], q12=q[1,12],
>           dq1=q[2,1], dq2=q[2,2], dq3=q[2,3],dq4=q[2,4], dq5=q[2,5], dq6=q[2,6],
>           dq7=q[2,7], dq8=q[2,8], dq9=q[2,9],dq10=q[2,10], dq11=q[2,11], dq12=dq[2,12],
>           ddq1=q[3,1], ddq2=q[3,2], ddq3=q[3,3], ddq4=q[3,4], ddq5=q[3,5],ddq6=q[3,6],
>           ddq[1]=q[3,1], ddq[2]=q[3,2], ddq[3]=q[3,3],ddq[4]=q[3,4],
>           ddq[5]=q[3,5], ddq[6]=q[3,6], ddq[7]=q[3,7], ddq[8]=q[3,8],
>           ddq[9]=q[3,9],ddq[10]=q[3,10], ddq[11]=q[3,11], ddq[12]=q[3,12],
>           Q1=Q[1], Q2=Q[2], Q3=Q[3], Q4=Q[4], Q5=Q[5], Q6=Q[6],
>           Q7=Q[7], Q8=Q[8], Q9=Q[9], Q10=Q[10], Q11=Q[11], Q12=Q[12]
>           x1=Torso[1,1],y1=Torso[1,2],z1=Torso[1,3],
           phi0=Torso[1,4],psi0=Torso[1,5], theta0=Torso[1,6],
>           dx1=Torso[2,1],dy1=Torso[2,2],dz1=Torso[2,3],
           dphi0=Torso[2,4],dpsi0=Torso[2,5], dtheta0=Torso[2,6],
>           ddx1=Torso[3,1],ddy1=Torso[3,2],ddz1=Torso[3,3],
           ddphi0=Torso[3,4],ddpsi0=Torso[3,5], ddtheta0=Torso[3,6],
>           x4=rechts[1,1], y4=rechts[1,2], z4=rechts[1,3],
           phir=rechts[1,4], psir=rechts[1,5], thetar=rechts[1,6],
>           dx4=rechts[2,1], dy4=rechts[2,2],dz4=rechts[2,3],
           dphir=rechts[2,4],dpsir=rechts[2,5],dthetar=rechts[2,6],
>           ddx4=rechts[3,1], ddy4=rechts[3,2], ddz4=rechts[3,3],
           ddphir=rechts[3,4], ddpsir=rechts[3,5],ddthetar=rechts[3,6],
>           x7=links[1,1],y7=links[1,2],z7=links[1,3],
           phil=links[1,4],psil=links[1,5],thetal=links[1,6],
>           dx7=links[2,1],dy7=links[2,2],dz7=links[2,3],
           dphil=links[2,4],dpsil=links[2,5],dthetal=links[2,6],
>           ddx7=links[3,1],ddy7=links[3,2],ddz7=links[3,3],
           ddphil=links[3,4],ddpsil=links[3,5],ddthetal=links[3,6]}:
> M:=subs(dMatrix,evalm(Mla)):
> CG:=subs(dMatrix,evalm(CGla)):
> CG1:=evalm(vector([CG[1], CG[2], CG[3], CG[4], CG[5], CG[6],0,0,0,0,0,0])):
> CG2:=evalm(vector([0,0,0,0,0,0,CG[7], CG[8], CG[9], CG[10], CG[11], CG[12]])):
> MProc:=makeproc(M,[Torso,q]):
> fremove("W:\\CBuilder\\BARt\\Robot\\M.mc"):
```

```
> C(MProc,optimized,ansi,filename="W:\\CBuilder\\BARt\\Robot\\M.mc");
> CG1Proc:=makeproc(CG1,[Torso,q]):
> fremove("W:\\CBuilder\\BARt\\Robot\\CG1.mc");
> C(CG1Proc,optimized,ansi,filename="W:\\CBuilder\\BARt\\Robot\\CG1.mc");
> CG2Proc:=makeproc(CG2,[Torso,q]):
> fremove("W:\\CBuilder\\BARt\\Robot\\CG2.mc");
> C(CG2Proc,optimized,ansi,filename="W:\\CBuilder\\BARt\\Robot\\CG2.mc");
Versuche
> dKin:=subs(Abmessungen, subs(dMatrix,Ableitungen,Ersetzung2,
             subs(dirKin,DdirKin,DDdirKin,
>            matrix([[  x4,  y4,  z4,  x7, y7, y8],
>                    [ dx4, dy4, dz4, dx7, dy7, dy8],
>                    [ddx4,ddy4,ddz4, ddx7, ddy7, ddy8]]))):
> dKinProc:=makeproc(dKin,[rechts,links,Torso,q]):
> fremove("W:\\CBuilder\\BARt\\Robot\\dirKin.mc");
> C(dKinProc,optimized,ansi,filename="W:\\CBuilder\\BARt\\Robot\\dirKin.mc");
> invKinProc:=makeproc(invKin,[Fuss]):
> fremove("W:\\CBuilder\\BARt\\invKin.mc");
> C(invKinProc,optimized,ansi,filename="W:\\CBuilder\\BARt\\invKin.mc");
```

Appendix B

Delphi program

```
unit visualisier;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, OpenGL, GL, GLU, GLUT, groessen;

const
 camReset  = 0;
 camVorne  = 1;
 camHinten = 2;
 camRechts = 3;
 camLinks  = 4;
 camOben   = 5;


type tCamera = record
                phi,    // Azimutwinkel
                alpha,  // Polabstand (theta)
                radius : real;
                center : tPunkt;
               end;

type
  TVisualisierForm = class(TForm)
    StatusBar: TStatusBar;
    procedure FormActivate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private-Deklarationen }
```

```
  public
    { Public-Deklarationen }
    DC      : HDC;
    hrc     : HGLRC;
    Palette : HPALETTE;
    Cam     : tCamera;
    MouseAlt  : record x, y : real; end;
    KameraAlt : tCamera;


    procedure DrawScene;
    procedure KameraPosChanged(Sender : tObject);
    procedure SetKameraPos(Nr:Integer);

   protected
    procedure WMPaint(var Msg: TWMPaint); message WM_PAINT;
    procedure WMQueryNewPalette(var Msg: TWMQueryNewPalette); message WM_QUERYNEWPALETTE;
    procedure WMPaletteChanged(var Msg: TWMPaletteChanged);   message WM_PALETTECHANGED;

  end;

var
  VisualisierForm: TVisualisierForm;

implementation

uses dataModule, Fussteile, OpenGLPrimitive;

{$R *.DFM}

const
  glfLightAmbient : Array[0..3] of GLfloat = (0.1, 0.1, 0.1, 1.0);
  glfLightDiffuse : Array[0..3] of GLfloat = (0.7, 0.7, 0.7, 1.0);
  glfLightSpecular: Array[0..3] of GLfloat = (0.0, 0.0, 0.0, 1.0);

const
  glfTorsoColor :   Array[0..3] of GLfloat = (0.0, 0.0, 1.0, 1.0);
  glfBeinColor  :   Array[0..3] of GLfloat = (0.4, 0.1, 0.0, 0.1);
  glfGelenkColor:   Array[0..3] of GLfloat = (1.0, 1.0, 1.0, 1.0);
  glfFloorColor:    Array[0..3] of GLfloat = (0.2, 0.2, 0.2, 1.0);


procedure TVisualisierForm.FormCreate(Sender: TObject);
var hHeap      : THandle;
    nColors, i : Integer;
    lpPalette  : PLogPalette;
    byRedMask, byGreenMask, byBlueMask : Byte;
    nPixelFormat : Integer;
    pfd        : TPixelFormatDescriptor;

begin
 DC:=GetDC(Handle);
 Fillchar(pfd, SizeOf(pfd),0);
 with pfd do
```

```
 Begin
  nSize           := sizeof(pfd);
  nVersion        := 1;
  dwFlags         := PFD_DRAW_TO_WINDOW or PFD_SUPPORT_OPENGL or PFD_DOUBLEBUFFER;
  iPixelType      := PFD_TYPE_RGBA;
  cColorBits      := 24;
  cDepthBits      := 32;
  iLayerType      := PFD_MAIN_PLANE;
 end;

nPixelFormat := ChoosePixelFormat(DC,@pfd);
SetPixelFormat(DC, nPixelFormat, @pfd);

DescribePixelFormat(DC, nPixelFormat, sizeof(TPixelFormatDescriptor), pfd);

if ((pfd.dwFlags and PFD_NEED_PALETTE)<>0) then
   Begin
    nColors                  := 1 shl pfd.cColorBits;
    hHeap                    := GetProcessHeap;
    lpPalette                := HeapAlloc(hHeap, 0 ,sizeof(TLogPalette)+
     (nColors*sizeof(TPaletteEntry)));
    lpPalette^.palVersion    := $300;
    lpPalette^.palNumEntries := nColors;

    byRedMask   := (1 shl pfd.cRedBits)  -1;
    byGreenMask := (1 shl pfd.cGreenBits)-1;
    byBlueMask  := (1 shl pfd.cBlueBits) -1;

    for i:=0 to nColors-1 do
     begin
      lpPalette^.palPalEntry[i].peRed:=  (((i shr pfd.cRedShift) and byRedMask)* 255)
           DIV byRedMask;
      lpPalette^.palPalEntry[i].peGreen:=(((i shr pfd.cGreenShift) and byGreenMask)* 255)
           DIV byGreenMask;
      lpPalette^.palPalEntry[i].peBlue:= (((i shr pfd.cBlueShift) and byBlueMask)* 255)
           DIV byBlueMask;
     end;

    Palette:=CreatePalette(lpPalette^);
    HeapFree(hHeap, 0, lpPalette);

    if (Palette<>0) then
     begin
      SelectPalette(DC, Palette, False);
      RealizePalette(DC);
     end;
   end;

 hrc:=wglCreateContext(Canvas.Handle);
 wglMakeCurrent(DC, hrc);

 gluPerspective(45.0, Width/Height, 1.0, 1000.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
```

```
  glShadeModel(GL_SMOOTH);


  //
  // Enable depth testing and backface culling.
  //
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);

  //
  // Add a light to the scene.
  //
  glLightfv(GL_LIGHT0, GL_AMBIENT, @glfLightAmbient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, @glfLightDiffuse);
  glLightfv(GL_LIGHT0, GL_SPECULAR,@glfLightSpecular);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  glutInit(MS_LIB);
  Zylinder(true);
  Zylinderzu(true);

 FormResize(Sender);
 SetKameraPos(camReset);
end;



procedure TVisualisierForm.FormDestroy(Sender: TObject);
begin
  wglMakeCurrent(0, 0);
  wglDeleteContext(hrc);
  ReleaseDC(Handle, DC);
  if (Palette <> 0) then DeleteObject(Palette);
  Datas.ActiveVisualisierung:=Nil;
end;


procedure TVisualisierForm.FormActivate(Sender: TObject);
begin
 Datas.ActiveVisualisierung:=Self;
end;


procedure TVisualisierForm.FormResize(Sender: TObject);
begin
  // Redefine the viewing volume and viewport when the window size changes.
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity;
  gluPerspective(30.0,            // Field-of-view angle
                 Width/Height,    // Aspect ratio of viewing volume
                 1.0,             // Distance to near clipping plane
                 100.0);          // Distance to far clipping plane
```

```
  glViewport(0, 0, ClientWidth, ClientHeight);
  glMatrixMode(GL_MODELVIEW);
  InvalidateRect(Handle, nil, False);
end;


procedure TVisualisierForm.WMPaint(var Msg: TWMPaint);
var ps : TPaintStruct;
begin
  // Draw the scene.
  BeginPaint(Handle, ps);
  DrawScene;
  EndPaint(Handle, ps);
end;


procedure TVisualisierForm.WMQueryNewPalette(var Msg : TWMQueryNewPalette);
begin
  //
  // If the program is using a color palette, realize the palette
  // and update the client area when the window receives the input
  // focus.
  //
  if (Palette <> 0) then begin
    Msg.Result := RealizePalette(DC);
    if (Msg.Result <> GDI_ERROR) then
      InvalidateRect(Handle, nil, False);
  end;
end;

procedure TVisualisierForm.WMPaletteChanged(var Msg : TWMPaletteChanged);
begin
  //
  // If the program is using a color palette, realize the palette
  // and update the colors in the client area when another program
  // realizes its palette.
  //
  if ((Palette <> 0) and (THandle(TMessage(Msg).wParam) <> Handle)) then begin
    if (RealizePalette(DC) <> GDI_ERROR) then
      UpdateColors(DC);

    Msg.Result := 0;
  end;
end;

procedure TVisualisierForm.KameraPosChanged;
begin
     with Cam do begin;
       if phi < 2*pi then phi := phi + 2*pi;
       if phi > 2*pi then phi := phi - 2*pi;
     end;
{     if Sender is tScrollBar then
        rKamera := ZoomBar.Position/100
     else
```

```
        ZoomBar.Position := round(rKamera * 100); }
    DrawScene;
end;

procedure DrawTorso;  //size of the Torso is the same as BARt-UH I
 const Tx=0.6;
       Ty=0.4;
 begin
  glColor3f(0.5,0.5,0.5);
  glPushMatrix;
  glScalef(Tx,Ty,11);
  glutSolidCube(1);
  glPopMatrix;
 end;

procedure DrawOberschenkel;
 begin
  glColor3f(0,0.5,0);
  glpushMatrix;
  glScale(Rohrdurchmesser,Rohrdurchmesser, 12);
  glCallList(glZylinder);
  glpopMatrix;
 end;

procedure DrawUnterSchenkel;
 begin
  glColor3f(0,0.5,0);
  glpushMatrix;
  glScale(Rohrdurchmesser,Rohrdurchmesser,13);
  glCalllist(glZylinder);
  glpopMatrix;
 end;

procedure DrawGelenk;
 begin
 glColor3f(1,1,1);
 glpushMatrix;
 glutSolidSphere(Rohrdurchmesser,30,30);
 glpopMatrix;
end;

procedure DrawFuss;
  begin
   glPushMatrix;
   glTranslate(0,0,0.042);
   Zentralteil;
   glTranslate(0,0,0.0295);
   Knochel;
   glTranslate(0.0569,0.0669,-0.0595);
   glRotate(30,1,0,0);
   Querstrebev;
   glTranslate(-0.1138,0,0);
   Querstrebev;
   glPopMatrix;
```

```
   glPushMatrix;
   glTranslate(0.0569,0.10856,0);
   Zeh;
   glTranslate(-0.1138,0,0);
   Zeh;
   glTranslate(0,-0.18,0.032);
   glRotate(-30,1,0,0);
   Hackenhalter;
   glTranslate(0.1138,0,0);
   Hackenhalter;
   glPopMatrix;
   glPushMatrix;
   glRotate(90,0,1,0);
   glTranslate(0,-0.10032,0);
   Querstrebeh;
   glPopMatrix;
   end;


procedure TVisualisierForm.DrawScene;
begin
 If not visible then exit;
 glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
 glLoadIdentity;

  with Cam do
   if alpha < 180 then
       gluLookAt(radius * sin(alpha) * sin(phi), // eye point
                 radius * sin(alpha) * cos(phi),
                 radius * cos(alpha),
                 Center.x, Center.y, Center.z,       // center of scene
                 -cos(alpha) * sin(phi),
                 -cos(alpha) * cos(phi),
                  sin(alpha))      // up vector
   else gluLookAt(radius * sin(alpha) * sin(phi), // eye point
                 radius * sin(alpha) * cos(phi),
                 radius * cos(alpha),
                 Center.x, Center.y, Center.z,       // center of scene
                 0, 0, -1);      // up vector



  with AktuellerZustand do
   begin
   BeginRead;
    glPushMatrix;
    glScale(1.0,1.0,0.02);
     glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfFloorColor);
    glutSolidCube(1);
    glPopMatrix;

    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfTorsoColor);
    glTranslate(Torso.x,Torso.Y,Torso.z);
    glRotate(Torso.phi/pi*180,1,0,0);
```

```
glRotate(Torso.psi/pi*180,0,1,0);
glRotate(Torso.theta/pi*180,0,0,1);
DrawTorso;
glPushmatrix;
glTranslatef(beinabstand/2,0,-l1/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
DrawGelenk;
glRotate(q[3]/pi*180,1,0,0);
glRotate(q[2]/pi*180,0,1,0);
glRotate(q[1]/pi*180,0,0,1);
glTranslatef(0,0,-l2/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfBeinColor);
DrawOberschenkel;
glTranslatef(0,0,-l2/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
DrawGelenk;
glRotatef(q[4]/pi*180,1,0,0);
glTranslatef(0,0,-l3/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfBeinColor);
DrawUnterschenkel;
glTranslatef(0,0,-l3/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
DrawGelenk;
glRotatef(q[5]/pi*180,1,0,0);
glRotatef(q[6]/pi*180,0,1,0);
glTranslate(0,0,-0.052-0.049);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
DrawFuss;
glPopmatrix;
glTranslatef(-beinabstand/2,0,-l1/2);
glPushmatrix;
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
Drawgelenk;
glRotatef(q[11]/pi*180,1,0,0);
glRotatef(q[10]/pi*180,0,1,0);
glRotatef(q[9]/pi*180,0,0,1);
glTranslatef(0,0,-l2/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfBeinColor);
Drawoberschenkel;
glTranslatef(0,0,-l2/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
DrawGelenk;
glRotatef(q[12]/pi*180,1,0,0);
glTranslatef(0,0,-l3/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfBeinColor);
DrawUnterschenkel;
glTranslatef(0,0,-l3/2);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
Drawgelenk;
glRotatef(q[13]/pi*180,1,0,0);
glRotatef(q[14]/pi*180,0,1,0);
glTranslate(0,0,-0.052-0.049);
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfGelenkColor);
DrawFuss;
```

```
    glPopMatrix;
    {glPushMatrix;
     glScale(1.0,1.0,0.02);
     glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfFloorColor);
    glutSolidCube(1);
    glPopMatrix;
    glTranslatef(Links.x,Links.y,Links.z);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfSphereColor);
    glRotatef(180,0,0,1);
    glRotatef(90,1,0,0);
    glRotatef(q[0]/pi*180,1,0,0);
    glutSolidSphere(a*1.8,20,20);
    glTranslate(0,l1/2,0);
    glPushMatrix;
     glScalef(a,l1,a);
     glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfMaterialColor);
     glutSolidCube(1);
    glPopMatrix;

    glTranslate(0,l1/2,0);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfSphereColor);
    glRotatef(q[1]/pi*180,1,0,0);
    glutSolidSphere(a*1.2,20,20);
    glTranslate(0,l2/2,0);
    glPushMatrix;
     glScalef(a,l2,a);
     glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfMaterialColor);
     glutSolidCube(1);
    glPopMatrix;

    glTranslate(0,l2/2,0);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfSphereColor);
    glRotatef(q[2]/pi*180,1,0,0);
    glutSolidSphere(a*1.2,20,20);
    glTranslate(0,l3/2,0);
    glPushMatrix;
     glScalef(a,l3,a);
     glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, @glfMaterialColor);
     glutSolidCube(1);
    glPopMatrix;}
    SwapBuffers(DC);
    StatusBar.Panels[0].Text:='t='+FloatToStr(round(t*10000)/10000)+'s';
    EndRead;
   end;
end;


procedure TVisualisierForm.SetKameraPos(Nr:Integer);
begin
 with cam do
  Case Nr of
   camReset : begin
              alpha:=pi/3;phi:=pi/3;radius:=3;
              center.x:=0;center.y:=0;center.z:=0;
```

```
            end;
   camVorne: begin
              alpha:=pi/2;
              phi:=0;
             end;
   camHinten: begin
              alpha:=pi/2;
              phi:=pi;
             end;
   camRechts: begin
              alpha:=pi/2;
              phi:=pi/2;
             end;
   camLinks: begin
              alpha:=pi/2;
              phi:=-pi/2;
             end;
   camOben : begin
              alpha:=0;
              phi:=0;
            end;
   else MessageDLG('Unbekannte Kamera-Position. Poisitonierung nicht mglich!'
                   ,mtError,[mbOk],0);
  end;
 KameraPosChanged(Nil);
end;



procedure TVisualisierForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
const
  DeltaWinkel = Pi/50;
  DeltaEntfernung = 0.05;
  DeltaRadius = 0.2;

begin
 with Cam do case Key of
  13 : SetKameraPos(camReset); { Kamera Reset }
  39 : if ssShift in Shift then center.x := center.x + DeltaEntfernung { rechts }
       else phi := phi + DeltaWinkel;
  37 : if ssShift in Shift then center.x := center.x - DeltaEntfernung { links }
       else phi := phi - DeltaWinkel;
  38 : if ssShift in Shift then center.y := center.y + DeltaEntfernung { oben }
       else alpha := alpha + DeltaWinkel;
  40 : if ssShift in Shift then center.y := center.y - DeltaEntfernung { unten }
       else alpha := alpha - DeltaWinkel;
  33 : if ssShift in Shift then center.z := center.z - DeltaEntfernung { nhern }
       else radius := radius - DeltaRadius;
  34 : if ssShift in Shift then center.z := center.z + DeltaEntfernung { entfernen }
       else radius := radius + DeltaRadius;
  123 : begin { Vollbild Umschaltung }
{         If WindowState=wsMaximized then
```

```
          begin
           WindowState:=wsNormal;
           BorderStyle:=bsSizeable;
           width:=oldwidth;
           height:=oldheight;
           top:=oldtop;
           left:=oldleft;
          end
        else
          begin
           oldwidth:=width;
           oldheight:=height;
           oldtop:=top;
           oldleft:=left;
           WindowState:=wsMaximized;
           BorderStyle:=bsNone;
          end;
         Scene.ReInit(Handle);}
        end;
  else begin
//        ShowMessage(IntToStr(Key));
        exit;
       end;
 end;
 KameraPosChanged(Sender);
end;

procedure TVisualisierForm.FormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  if ssRight in Shift then with Cam do begin;
{    center.x:=center.x+(x-MouseAlt.x)/width *0.03;
    center.y:=center.y-(y-MouseAlt.y)/height*0.03;
    KameraPosChanged(Sender);}
  end;
  if ssLeft in Shift then
    with Cam do begin
     alpha := (MouseAlt.y - y)/height*pi + KameraAlt.alpha;
     phi   := -(MouseAlt.x - x)/width*pi  + KameraAlt.phi;
     KameraPosChanged(Sender);
    end;
end;

procedure TVisualisierForm.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
   MouseAlt.X := x;
   MouseAlt.y := y;
   KameraAlt.alpha := Cam.alpha;
   KameraAlt.phi := Cam.phi;
end;

end.
```