ISSN 0280-5316
ISRN LUTFD2/TFRT--5792--SE

Modeling and Balancing of Spherical Pendulum using a Parallel Kinematic Manipulator

David Barrio Vicente

Department of Automatic Control
Lund University
April 2007

Department of Automatic Control **Lund Institute of Technology** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* April 2007 *Document Number* ISRNLUTFD2/TFRT--5792--SE
Author(s) David Barrio Vicente	*Supervisor* Anders Robertsson at Automatic Control in Lund. Rolf Johansson at Automatic Control in Lund (examiner) *Sponsoring organization*

Title and subtitle
Modeling and Balancing of Spherical Pendulum using a Parallel Kinematic Manipulator (Modellering och balansering av en sfärisk pendel med en parallellkinematisk manipulator)

Abstract
The balancing act of an inverted pendulum with a robotic manipulator is a classical benchmark for testing modern control strategies in conjunction with fast sensor-guided movements. From the control design perspective, it presents a challenging and difficult problem as the system is open-loop unstable and includes nonlinear effects in the actuators, such as friction, backlash, and elasticity. In addition, the necessity of a sensor system that can measure the inclination angles of the pendulum contributes to the complexity of the balancing problem. The pendulum is projected onto the xz and yz planes of the inertial coordinate system. These projections are controlled by a state-space controller.
A specially developed sensor system allows the contactless measurement of the inclination angles of the pendulum. This system consists of a small magnet, placed at the bottom of the pendulum and Hall-effect sensors placed below the end effector.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language english	*Number of pages* 104	*Recipient's notes*
Security classification		

Contents

1 Introduction

A robot balancing an inverted pendulum is an impressive demonstration object that shows how an intelligent combination of modern control algorithms, robotics, and electronics can lead to a high-performance dynamic system. As such, it constitutes a typical mechatronical system.

Its realization requires the design of a state-space controller, calculation of the forward and inverse kinematics and the development of specialized signal processing electronics, which are necessary for the measurement of the inclination angles of the pendulum.

The goal project is controlling the balance of an inverted pendulum. The pendulum is supported over a plate, which is joined to robot hand.

The parallel robot, which I will use in this master thesis, will be a Gantry-Tau structure but the pendulum and its plate will be possible to use with others robots.

The lower pendulum part is an iron cone. Over the cone there is a magnetic object. On the plate surface, two hall-sensors for magnetic field will be placed to take measurements of pendulum angle on at x-y axis.

The TCP-robot will be moved along x-y directions.

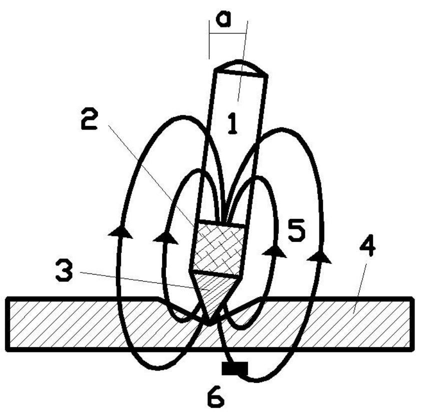


Figure 1.1 Cross section of pendulum and sensor.

Above picture shows the pendulum structure. It can be seen:

1. Pendulum (glass fibber tube).
2. Permanent magnet.
3. Cone (steel).
4. Mounting plate (plastic).
5. Magnetic flux.
6. Hall-effect sensor and (α) inclination angle of the pendulum.

My master thesis will have three stages:

- Simulation of pendulum dynamics: I will use MATLAB[1] and Dymola[2] software to do it.
- Control design: there are several possibilities (P, PI, PID, LQR, etc.). I will have to study everyone and to choose the most suitable.
- Design of electronics for angle measurements: hall-sensors give in a little signal which is in relation to magnetic flux that is in relation to pendulum angle. I will have to amplify that signal using operational amplifiers.

This master thesis will allow me to use and to extend my knowledge I learnt in Spain for last years about Electronics, Robotics and Automatic Control.

[1]MATLAB version 7.2.0.232 (R2006a) was used.

[2]Dymola version 6 was used.

2 PKMs

2.1 Robotics. Historical background

Today robots are natural components in the manufacturing industry and are even expanding to other fields. Robots are however a pretty young product compared to other equipment used today.

The robotic history began in the late 1950's in USA where George Devol and Joseph Engelberger started what was to become the Unimation. Joseph Engelberger is sometimes called "The Father of Robotics". Unimation was the first company who delivered robots to the American industry and General Motors was the first customer in 1961:

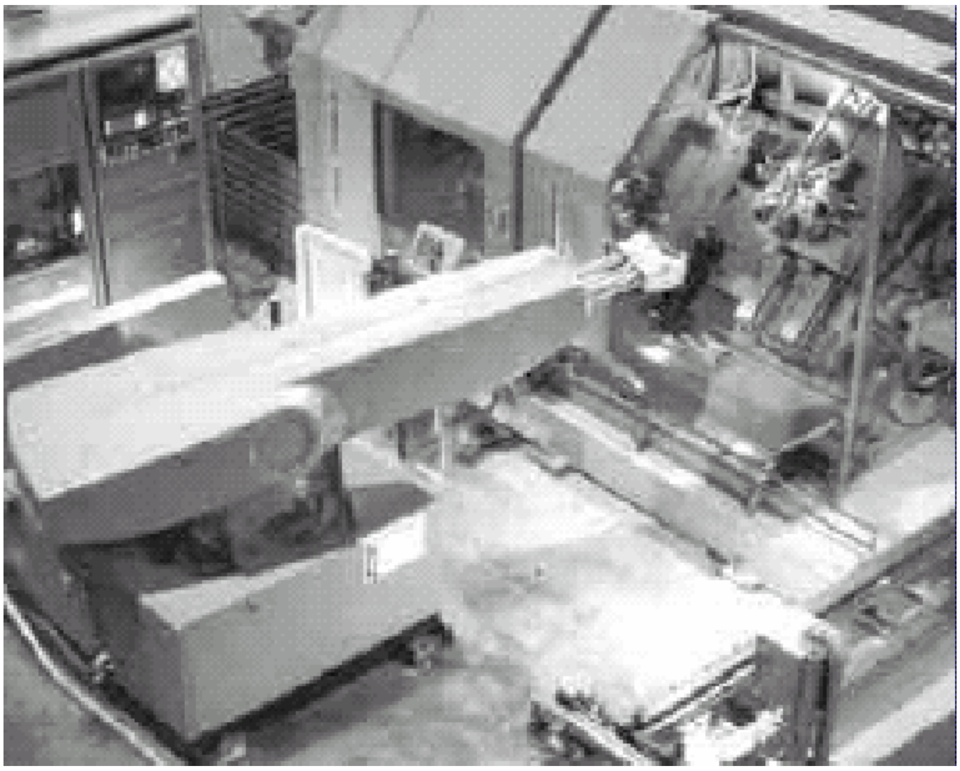


Figure 2.1 First industrial robot in a factory.

The word "Robot" comes from the Czech play "Rossums Universal Robots", performed in the 1920's. The big breakthrough came 1964 when General Motors ordered 66 Unimate robots from Unimation, to be installed in their new top modern factory in Ohio. Even though the industry was hard to convince the public was now very interested in the robots, and Unimate robots appear in commercials and talk shows. Soon several other companies followed. IBM, AMF, Hughes Aircraft and Western Electric are just some of many companies who started their own production of robots.

In Europe the Scandinavian countries were early to adopt the new invention and several companies in Sweden, Norway and Finland started to develop robots or produce the Unimate robot on licence. Among the pioneers in Sweden one can mention Roland Kaufeldt, founder of Kaufeldt AB, the kitchen appliances company Electrolux, Esab, manufacturer of welding products and Asea, electronics manufacturer. In Norway Trallfa, manufacturer of wheelbarrows, constructed a robot for painting which became a success story.

In 1971 Asea started to develop a robot, which would come to make ABB one of the main players on the market. The robot, which was to be called IRB 6, had a fully electronic control and power system, and was the first microprocessor controlled robot. It was also an anthropomorphic robot, i.e. it imitated the human anatomy. Using Harmonic Drives meant that it was much more compact than other robots. The production of IRB 6 started in 1973:

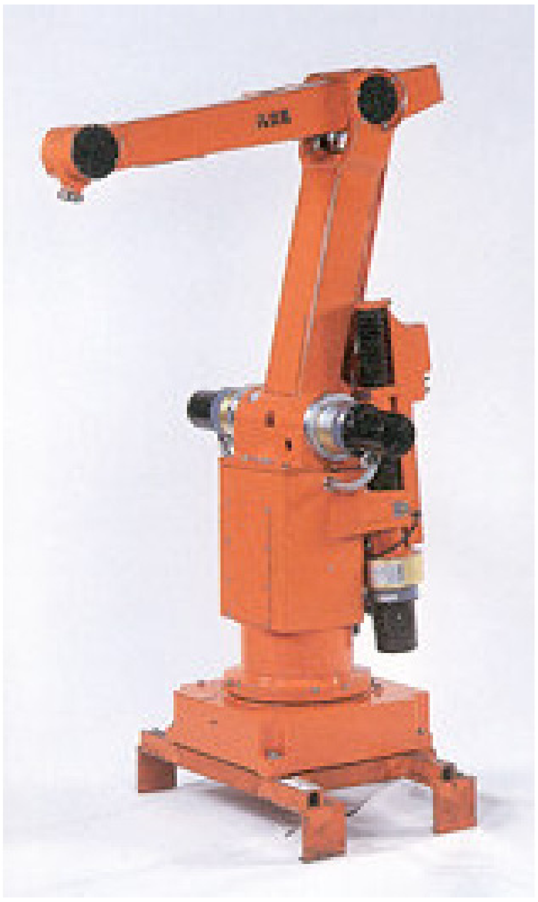


Figure 2.2 IRB 6 robot.

Other countries in Europe were not so eager to follow. Europe had a high unemployment rate and there was no need for robots since the pressure to raise productivity was relatively low. There was however some exceptions. The German company Kuka developed a robot used for welding, mainly sold to the European car industry. The European car industry also started to produce robots on their own.

While Europe had a high unemployment rate and no problem to get enough labour the situation was very much the opposite in Japan. The high economical growth in the 60’s resulted in a lack of labour. This meant that the companies were very open for new ideas and the robots were embraced as a way to increase the production. The industry was quick to apply the robots in the production and soon there were many

Japanese robot producers to compete on the growing market. In 1980 there were 150 Japanese robot producers and in 1988 nearly 70 % of the 256 000 robots in use all over the world were installed in Japan. Some large Japanese robot manufacturers today are Fanuc, Yaskawa and Kawasaki.

In Scandinavia Asea became the main robot producer in the mid 80's when both Electrolux robot production and Trallfa was incorporated. After Asea and the Swiss company Brown Boveri merged in 1988 and formed ABB, the robot production of Cincinnati Milicroms, Graco Robotics and Esab were also incorporated.

2.2 Parallel manipulators

Parallel kinematics manipulators (PKMs) have recently attracted a lot of interest in the robot community. The main reason for this is some inherited properties of the structure, mainly high stiffness and dynamical advantages.

A parallel mechanism can be defined as a closed-loop mechanism in which the end effector (mobile platform) is connected to the base by at least two independent kinematics chains. In other words, a parallel kinematics manipulator consists of several kinematics chains, in contrast to the serial that only consist of one. This is a very general definition that opens up for many different constructions, with very different properties.

There are already some PKMs on the market like *IRB 340 Flexpicker*, which is based on the Delta structure:

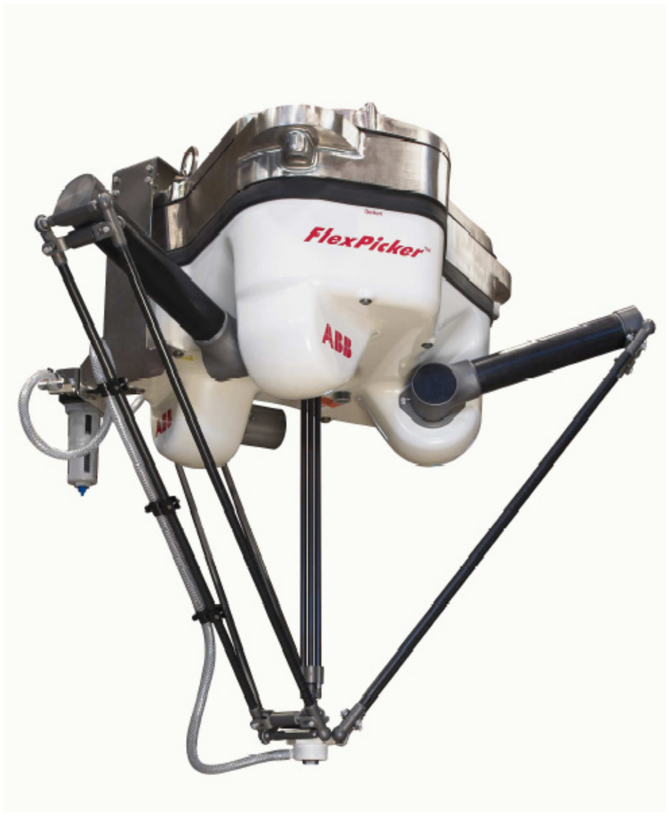


Figure 2.3 IRB 340 Flexpicker robot.

Other examples of parallel structures are the Hexaglide, the Triaglide, the I4 and the Orthoglide:

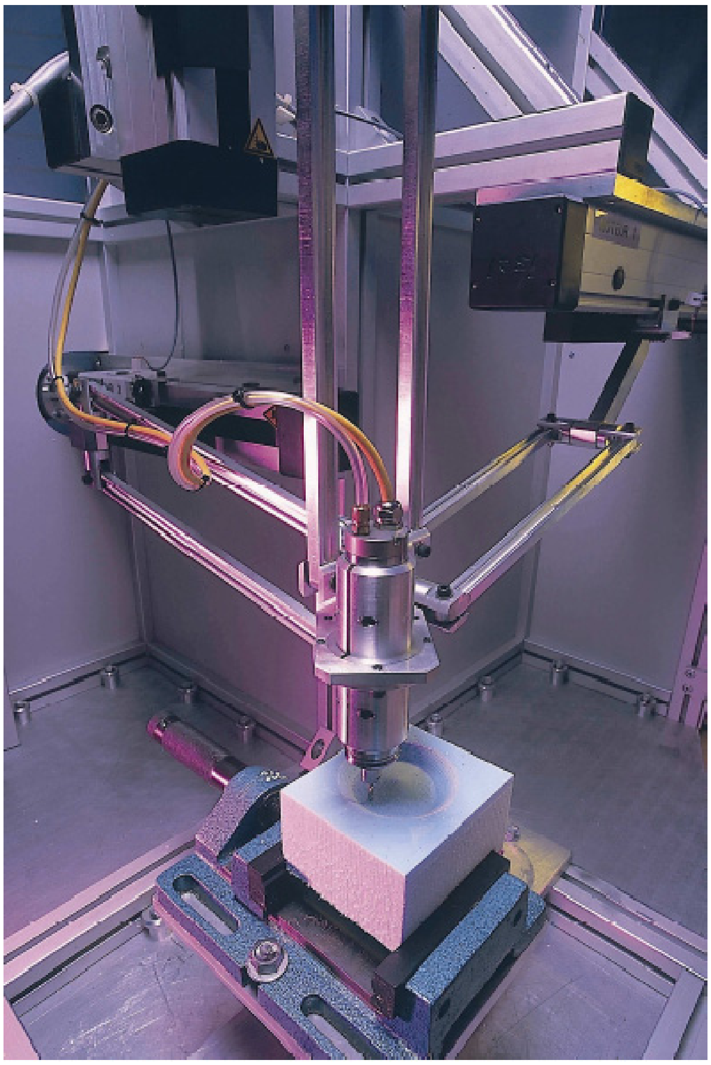


Figure 2.4 Orthoglide robot.

There is some common vocabulary that is used for parallel manipulators. The manipulator is said to consist of a mobile platform connected to a fixed base by several kinematics chains, called legs. If the number of legs is greater or equal to the degrees-of-freedom (DOFs) of the mobile platform and each arm having one actuated joint, the manipulator is called fully parallel.

2.3 Comparisons with serial structures

Same different properties of serial and parallel manipulators are workspace, payload, accuracy and dynamical behaviour. These are general properties, more or less true for different constructions, which give a background to raising interest in parallel robots and the problems inherited in the structure.

- Workspace: one of the main drawbacks with parallel robots is that they generally have a small workspace compared to the footprint of the robot.

- Payload: in a serial structure each actuator has to have the necessary power to move not only the manipulated object, but also the links and actuators located later in the kinematics chain. In a parallel structure the end-effector is directly supported by all actuators, and the actuators can be located close to the base, hence the payload can be much larger.
- Accuracy: in serial robots the errors from each link accumulate to a total error at the end-effector. An error in a joint closer to the base will also have a larger effect on the total error than an error in a joint closer to the end-effector. Parallel structures do not have these drawbacks at all, and are therefore remarkably rigid.
- Dynamical behaviour: the fact that the arm structure of a parallel robot can be made much lighter since the arms do not have to carry actuators, and the fact that errors do not accumulate, give them better dynamic performance than serial robots.

As seen in these comparisons, there are a lot of properties of a parallel structure that could make it interesting. The main drawback of the structure is the small workspace.

2.4 The Tau structure

As mentioned above the definition of a parallel manipulator opens up for a wide range of constructions. We will here study a special group of parallel manipulators based on a mobile platform, six arms and three actuators. Depending on where the arms are connected to the platform, how they are grouped and what kind of actuators used the performance will be very different even within this group of manipulators. One example of construction like this is the Orthoglide, (see before figure).

In this group one can arrange the structures according to how the arms are grouped. The Orthoglide has a structure that would be named 2/2/2, since the arms are grouped in pairs. If the arms are grouped as 3/2/1, 3/1/2, 2/3/1, 2/1/3, 1/3/2 or 1/2/3 the structure is referred to as a Tau-structure. One of the advantages of the Tau-structure is that the different configurations make it possible to get the highest stiffness in a desired direction. For a 2/2/2 structure there is only one configuration. ABB Robotics introduced the Tau family of parallel kinematics manipulators.

2.5 The Gantry-Tau structure

As stated before, the main drawback with parallel structures is the small workspace compared to the footprint of the robot. The Gantry-Tau [11] is constructed to overcome this limitation while retaining most of the parallel structures advantages.

The Gantry-Tau has a total workspace larger than for a serial gantry robot with the same footprint. The robot has been constructed for the assembly of aeroplane components. This is an application where very large and expensive machines are used today and a lighter and more cost efficient manipulator could compete.

The parallel robot used in this master thesis is a Gantry-Tau structure, with 3 translational DOF, that can allow a big working area compared to the other structures:

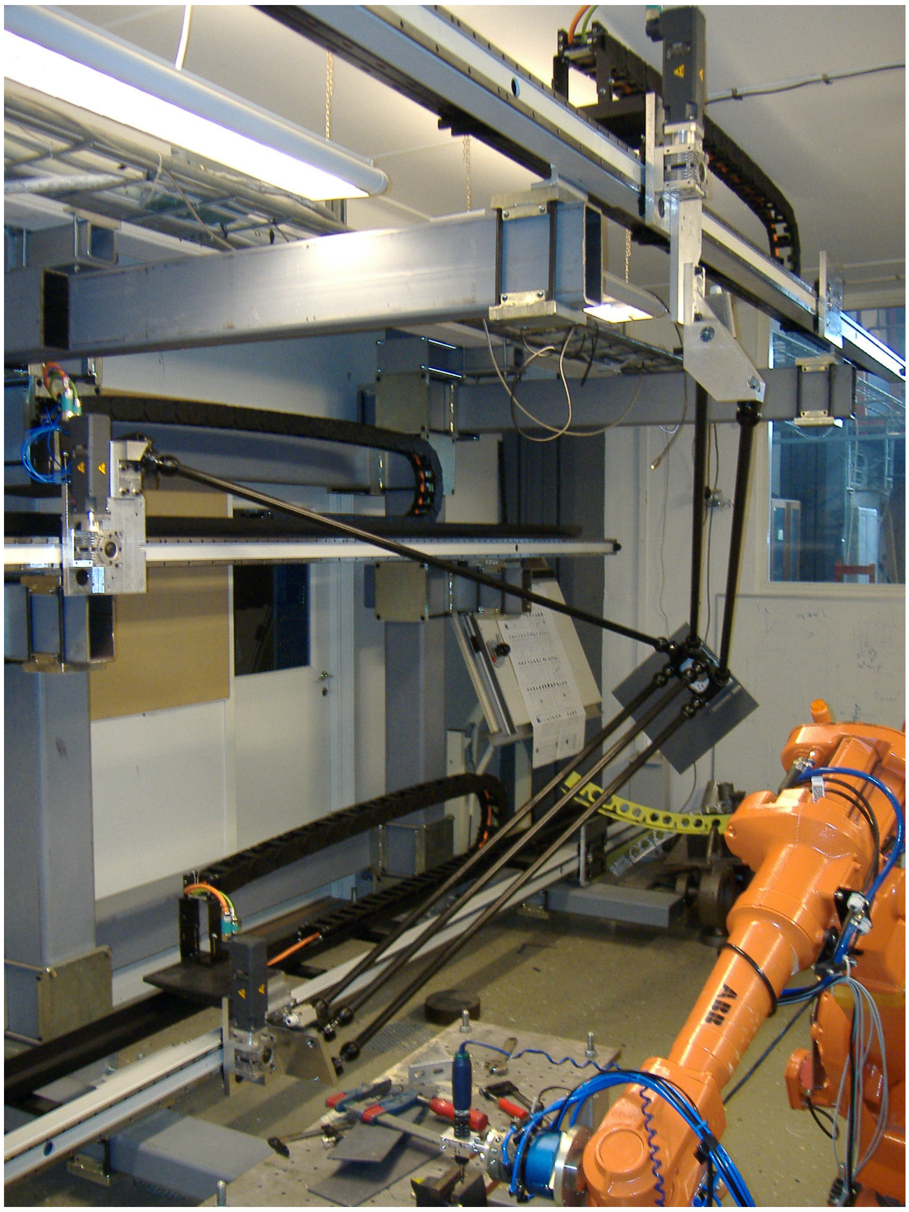


Figure 2.5 Gantry-Tau structure (robot on the left side).

This structure has 6 joints, in a 3-2-1 configuration, representing how many joints are on each kinematics group of the robot.

The table robot can became a demonstration model, easier to transport than the real size one.

The table robot consists of three parallel linear tracks, which are attached to a plate at each end. One of the end plates has been reduced to an L-shape so that the platform can move freely in this area. On the each track, the interface boards between tracks and PC are attached.

The picture below shows the table robot:

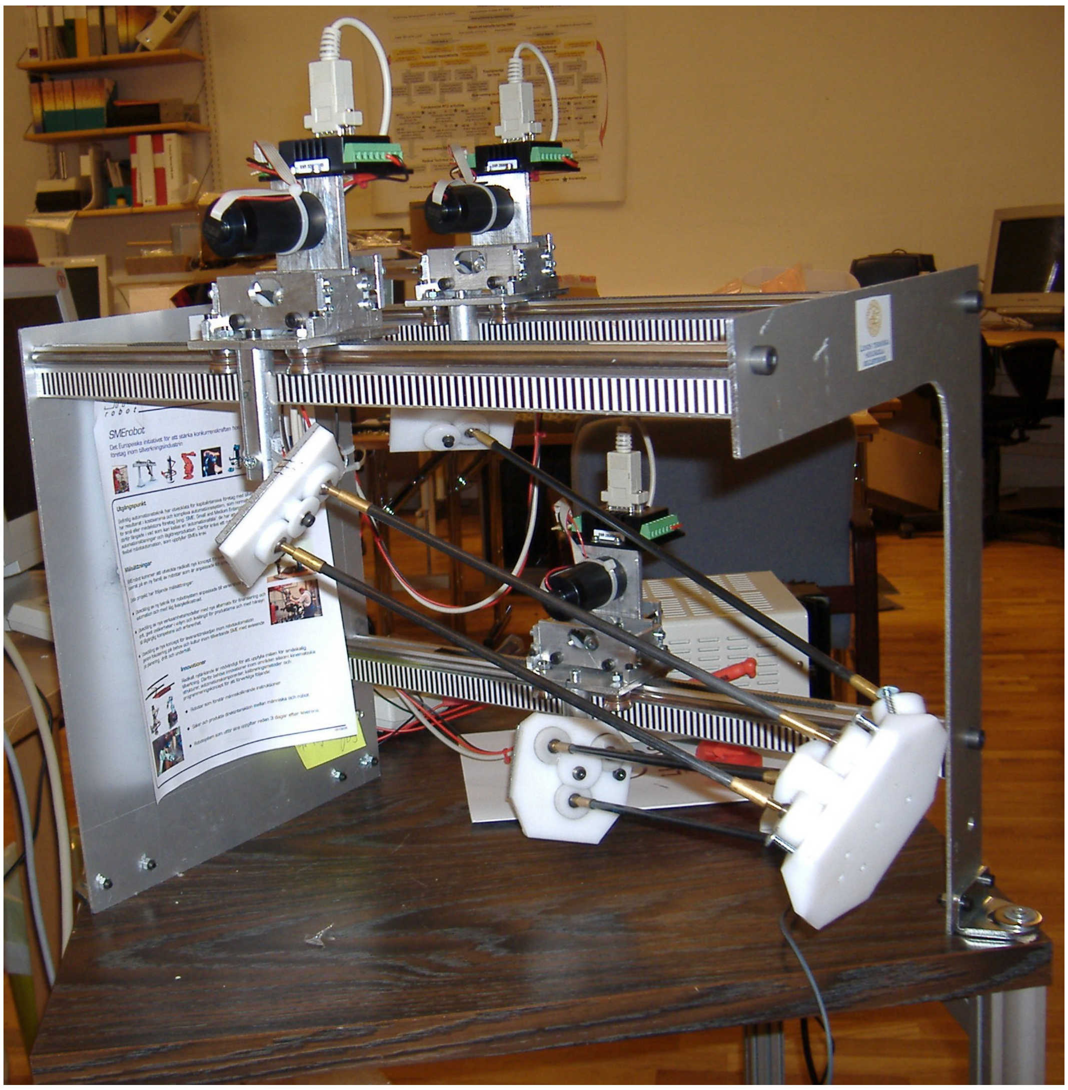


Figure 2.6 Table robot.

The cart moves on a toothed belt, driven by a DC motor that is coupled to a gear wheel. The tracks are about 50 cm long and fastened to the end plates with one screw at each end, so the robot is easy to transport and assemble. The bars have a diameter of 6 mm.

3 Modeling and simulation of pendulum dynamics

In this chapter I am going to explain how I have made the modeling and simulation of the pendulum dynamics.

In order to do that, Dymola simulation software [19] will be used. This software is based on Modelica language [23], and permits to build models easily using Drag and Drop, and simulate them as well as other many possibilities that will be seen later.

3.1 Modeling

In Dymola, I made a modeling of physic system. I just considered the motion of the hand robot. For it, I included:

- 2 prismatic actuators[3] (*motorX* and *motorZ*), which represent motions along X and Z-axis. These motions will be made by the robot hand.
- 2 accelerators[4] (*accelerateX* and *accelerateZ*), which represent input signal to every motor.
- 1 universal joint[5] (*cone*), which represents two DOFs of pendulum rotation.
- 1 body[6] (*bar*), which represents the pendulum.
- 1 absolute sensor[7] (*sensor*), which is able to give me every measure I need. These measures are: linear position and velocity of both motors, angular position and velocity of both degrees-of-freedom of the pendulum.
- 2 input ports[8] (*u1* and *u2*), to connect input signals to system.
- 8 output ports[9] (*x1_Xs, x2_Xv, x3_Zs, x4_Zv, x5_Aphi, x6_Aw, x7_Bphi* and *x8_Bw*), to read output signals from system.
- Of course, I need a reference system[10]: *world*.

[3]Included in Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic.

[4]Included in Modelica.Mechanics.Translational.Accelerate.

[5]Included in Modelica.Mechanics.MultiBody.Joints.Universal.

[6]Included in Modelica.Mechanics.MultiBody.Parts.BodyBox.

[7]Included in Modelica.Mechanics.MultiBody.Sensors.AbsoluteSensor.

[8]Included in Modelica.Blocks.Interfaces.RealInput.

[9]Included in Modelica.Blocks.Interfaces.RealOutput.

[10]Included in Modelica.Mechanics.MultiBody.World.

This one is the completed diagram:

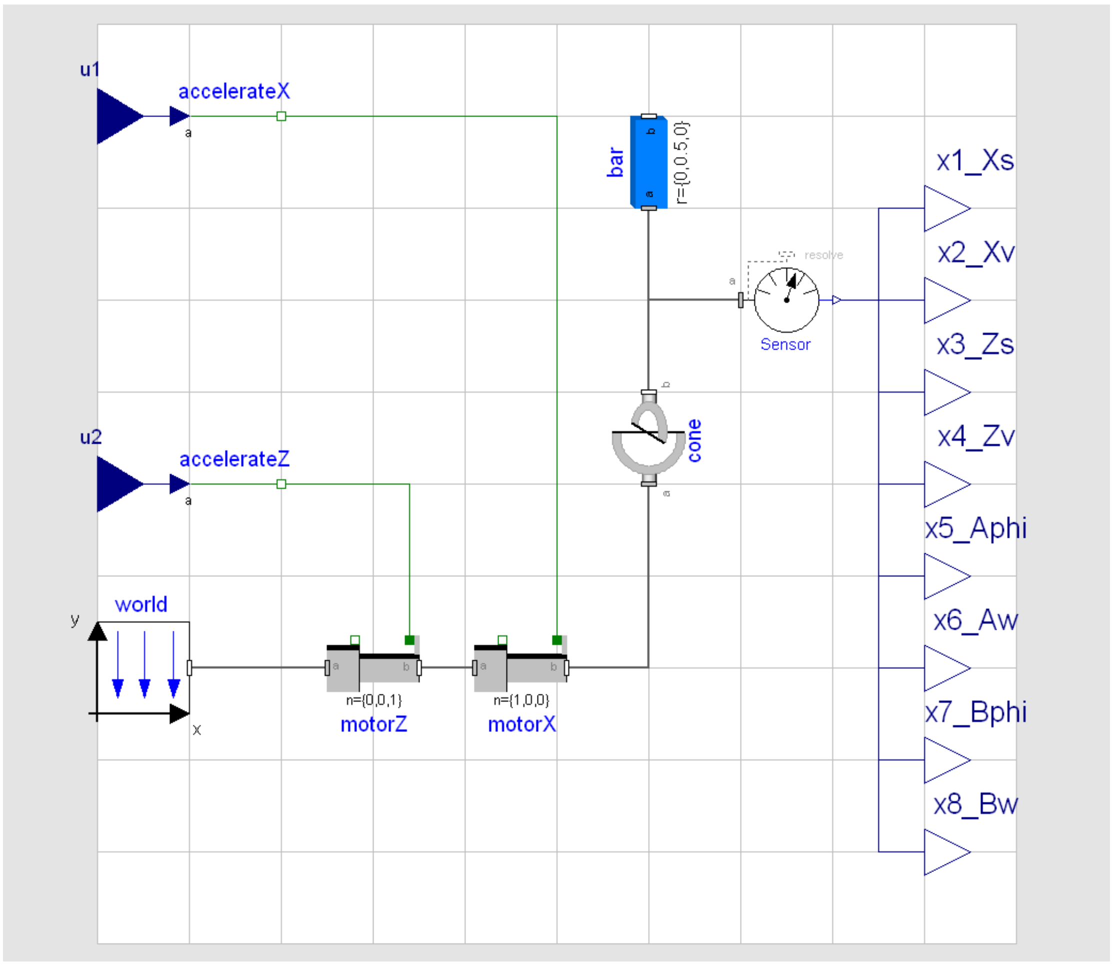
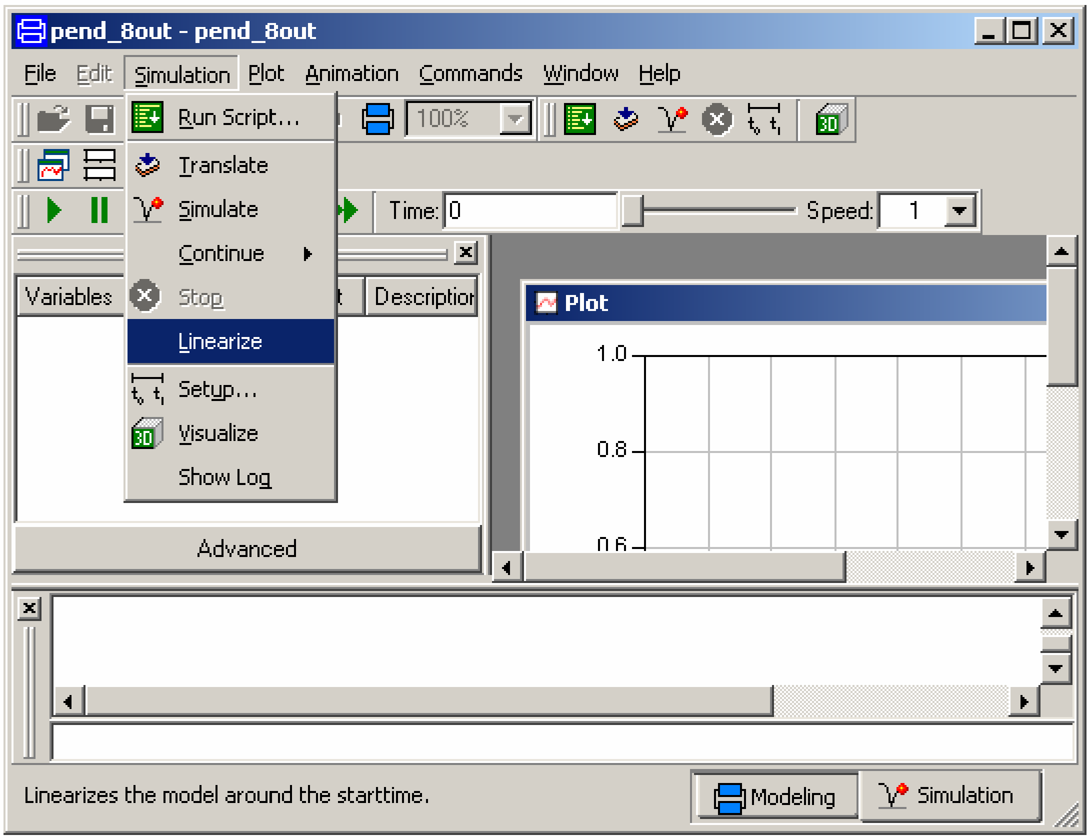


Figure 3.1 Dynamic model of the system.

Now I have a system and I can see its dynamic behaviour by means of Dymola. But that is not very useful. I already knew that the system would be unstable. For working with the system, for instance to calculate a controller, I need a mathematical model of it. Besides, that model must be linear. At the next step I will get a linear mathematical model.

3.2 Linearization of the dynamic model

Dymola has a tool to get a representation of a system by means of its state-space model. Besides, that model will be linear. It is possible to find that tool on the *Simulation* sheet, inside *Simulation* command:



Linearizes the model around the starttime.

Figure 3.2 *Linearize* tool in Dymola.

After executing this tool, a MATLAB file (*dslin.mat*) is made. This file stores all necessary information to make a state-space[11] model of the system using MATLAB software.

[11]More information about state-space representation in Appendix A.

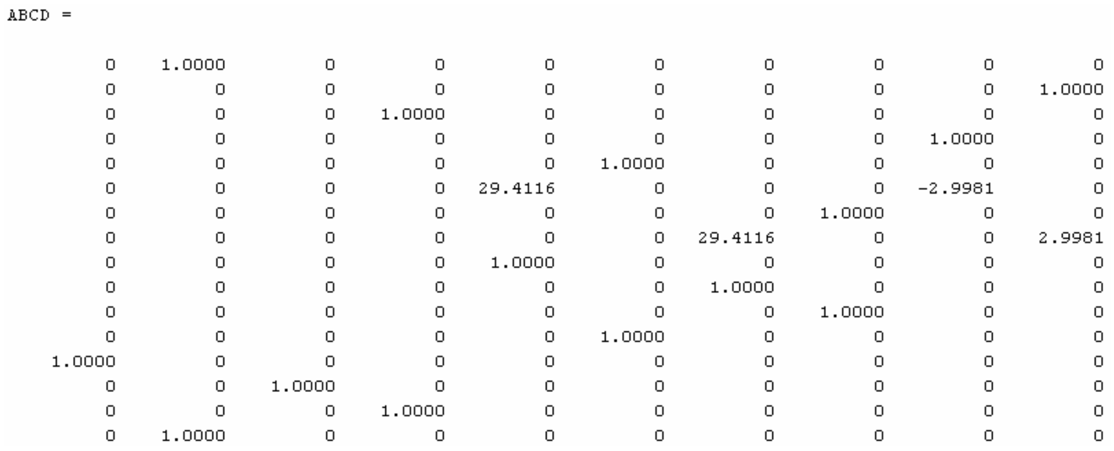
3.3 Mathematic model

The first step to extract a mathematic model from *dslin.mat* file is loading it in MATLAB: `load dslin_4outputs_ob.mat`

This command creates four variables. One of them (`xuyName`) contains information about inputs, outputs and states of the system:

```
xuyName =

motorX.s
motorX.v
motorZ.s
motorZ.v
cone.revolute_a.phi          states
cone.revolute_a.w
cone.revolute_b.phi
cone.revolute_b.w
u2
u1                           inputs
x5_Aphi
x7_Bphi
x8_Bw
x6_Aw                        outputs
x1_Xs
x3_Zs
x4_Zv
x2_Xv
```

Other variable contain *A*, *B*, *C* and *D* matrixes, that is, four matrixes to do a complete representation of a system by means of its state-space model:



```
ABCD =

         0    1.0000         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0    1.0000
         0         0         0    1.0000         0         0         0         0         0         0
         0         0         0         0         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000         0         0         0         0
         0         0         0         0   29.4116         0         0         0   -2.9981         0
         0         0         0         0         0         0         0    1.0000         0         0
         0         0         0         0         0         0   29.4116         0         0    2.9981
         0         0         0         0    1.0000         0         0         0         0         0
         0         0         0         0         0         0    1.0000         0         0         0
         0         0         0         0         0         0         0    1.0000         0         0
         0         0         0         0         0    1.0000         0         0         0         0
    1.0000         0         0         0         0         0         0         0         0         0
         0         0    1.0000         0         0         0         0         0         0         0
         0         0         0    1.0000         0         0         0         0         0         0
         0    1.0000         0         0         0         0         0         0         0         0
```

Figure 3.3 *ABCD* variable in MATLAB.

Next step is extracting *A*, *B*, *C* and *D* matrixes from *ABCD* variable.

$$ABCD = \begin{pmatrix} \begin{pmatrix} \boldsymbol{A} \\ 8\text{x}8 \end{pmatrix} & \begin{pmatrix} \boldsymbol{B} \\ 8\text{x}2 \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{C} \\ 8\text{x}8 \end{pmatrix} & \begin{pmatrix} \boldsymbol{D} \\ 8\text{x}2 \end{pmatrix} \end{pmatrix}_{16\text{x}10}$$

Which is very easy using MATLAB:

```
A=ABCD(1:8,1:8);
B=ABCD(1:8,9:10);
C=ABCD(9:16,1:8);
D=ABCD(9:16,9:10);
```

4 Control design

I checked in the before simulation that open-loop system is unstable. That means I need to introduce a control system to stabilizer it.

There are several possibilities to do it: proportional controller, proportional integral controller, proportional integral derivative controller, cascade controller, linear quadratic regulator…

Designing one of them (LQR) is very easy because I have state-space model in MATLAB. This software has a command (`lqr`), which give me *K* feedback matrix. I only need this matrix to control and stabilizer the closed-loop system.

4.1 LQR problem

The theory of optimal control is concerned with operating a dynamic system at minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic functional is called the LQ problem. One of the main results in the theory is that the solution is provided by the linear-quadratic regulator (LQR), a feedback controller whose equations are given below.

For a continuous-time linear system described by:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where:

x = state vector (n-dimension)

u = control vector (r-dimension)

y = output vector (m-dimension)

A = constant coefficient matrix (nxn dimension)

B = constant coefficient matrix (nxr dimension)

C = constant coefficient matrix (mxn dimension)

D = constant coefficient matrix (mxr dimension)

with a cost functional defined as:

$$J = \int_0^\infty (x^T(t)Qx(t) + u^T(t)Ru(t))dt$$

where the matrices Q and R are positive-semidefinite and positive-definite, respectively. Note that this cost functional is thought in terms of penalizing the control energy (measured as a quadratic form) and the time it takes the system to reach zero-state. This functional could seem rather useless since it assumes that the operator is driving the system to zero-state, and hence driving the output of the system to zero. This is indeed right, however the problem of driving the output to the desired level can be solved after the zero output one is. In fact, it can be proved that this secondary problem can be solved in a very straightforward manner. The optimal control problem defined with the previous functional is usually called the state regulator problem and its solution the linear quadratic regulator (LQR) which is no more than a feedback matrix gain of the form:

$$u(t) = -R^{-1}B^T Px(t) = -Kx(t)$$

where *K* is a rxn dimension matrix and *P* is found by solving the Riccati equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

This problem was elegantly solved by Rudolf Kalman (1960).

Therefore, the design of the systems of optimal control consists of calculating *K* matrix elements.

An advantage of using quadratic optimal control is that designed system will be stable, except in case of system is not controllable[12].

MATLAB has a command (`lqr`) which provides a solution Riccati equation in continuous time and it determines the optimal feedback gain matrix (*K*).

4.2 Design of the LQR controller

The first step is to check if the system is controllable. If the system is not controllable, when I connect the feedback by means of *K* matrix, the system may be not stable.

To check it, I have to calculate the matrix below:

$$CON = (B \quad AB \quad A^2B \quad A^3B \quad A^4B \quad A^5B \quad A^6B \quad A^7B)$$

System is controllable if and only if *rank*[13] *(CON) = n = 8.*

I checked using MATLAB that the system is controllable.

Before using `lqr` command in MATLAB, I have to determinate *Q* and *R* matrixes. Both matrixes must be positive-semidefinite. Then, the simplest form is identity matrix. So:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Now I can use `lqr` command to obtain *K* matrix: `K = lqr(A,B,Q,R)`

[12]More information about controllability in Appendix A.

[13]Rank is the number of linearly independent rows in a matrix.

4.3 Simulation of the designed controller

After executing `lqr` command in MATLAB, I have *K* matrix. To check if this matrix stabilizes the system, I have to simulate. At the beginning, I will simulate in MATLAB, and after that I will make a simulation in Dymola.

4.3.1 Simulation in MATLAB[14]

I will use `initial` command to do the simulation. This command allows to obtain initial condition response of state-space models. Before that, I have to close the loop. The only different matrix is *A*. Closed-loop *A* matrix (*Acl*) is $Acl = A - B*K$

Results for this simulation are shown below: initial conditions are: *Aphi* $= 10^o = 0.17$ *rad. and Bphi* $= 4^o = 0.07$ *rad.:*

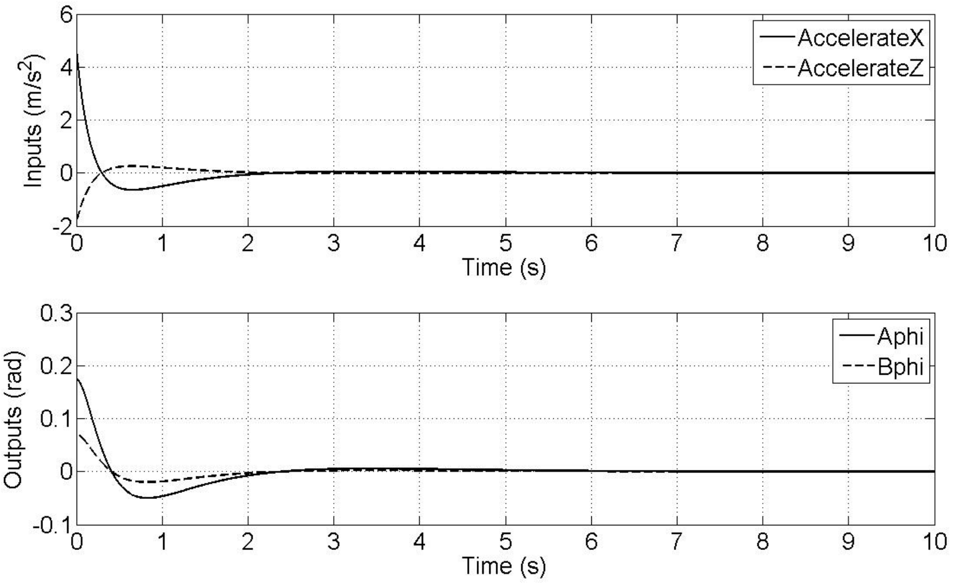


Figure 4.1 *Simulation obtained in MATLAB.*

I checked that results were like expected and that the response was stable.

4.3.2 Simulation in Dymola

The first step is creating a model with the system and the feedback. For it, I included a gain matrix[15] (*K_matrix*), which represents the optimal feedback gain matrix (*K*). Of course, I included a pendulum model.

[14]MATLAB file can be seen in Appendix C.

[15]Included in Modelica.Blocks.Math.MatrixGain.

I introduced values obtained from MATLAB in *K_matrix* block

Modeling of control system is shown below:

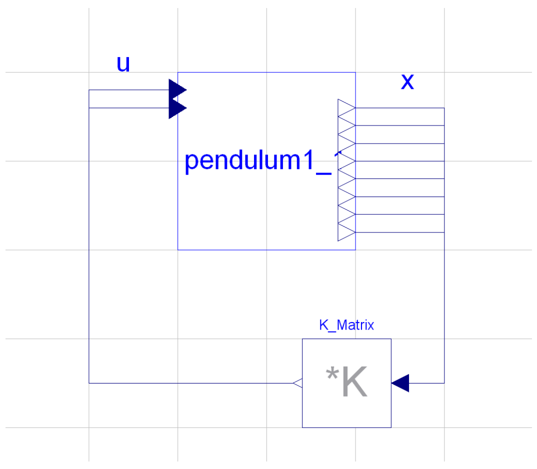


Figure 4.2 Dynamic model of the controlled system.

The follow graphs show results (initial conditions are the same that before one):

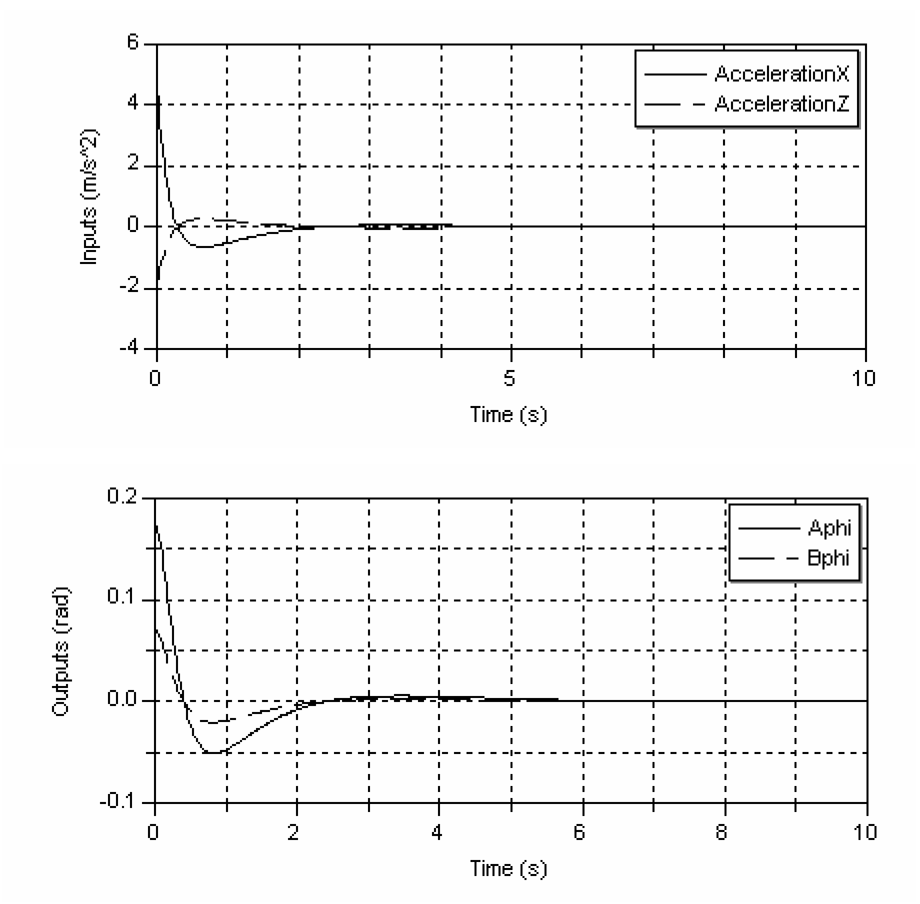


Figure 4.3 *Simulation obtained in Dymola.*

Results in Dymola were similar that MATLAB, like expected.

5 Modeling of pendulum dynamic (4 outputs)

In the real system, I cannot measure eight states (position and velocity along X and Z axis, and angle and angular velocity around two directions). So I have to do a modeling only with states I can measure (outputs). These states are position along X and Z-axis, and pendulum angle around two directions (four outputs).

5.1 Modeling

The new model is below:

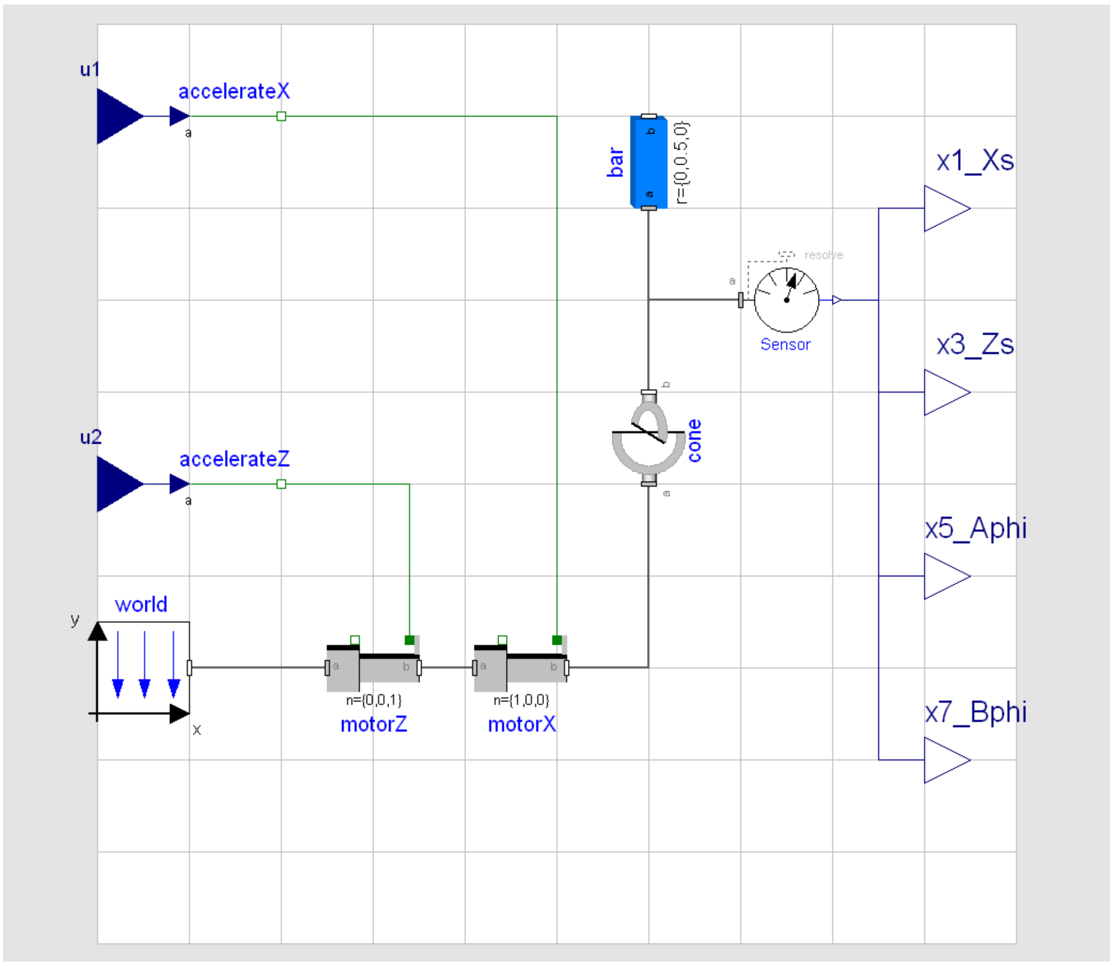


Figure 5.1 Dynamic model of the system.

This model is similar with previous model but now I have only four states like outputs. I have removed states I cannot measure such as velocity along X and Z-axis, and angular velocity around two directions. Hence I have held states which I can measure such as position along X and Z-axis, and pendulum angle around two directions.

5.2 Linearization of the dynamic model

The next step is to execute *Linearize* tool in Dymola. So I obtain MATLAB file (*dslin.mat*), which stores all necessary information to make a state-space model of the system using MATLAB software.

5.3 Mathematic model

Procedure to extract a mathematic model from *dslin.mat* file is similar to chapter 3.1.

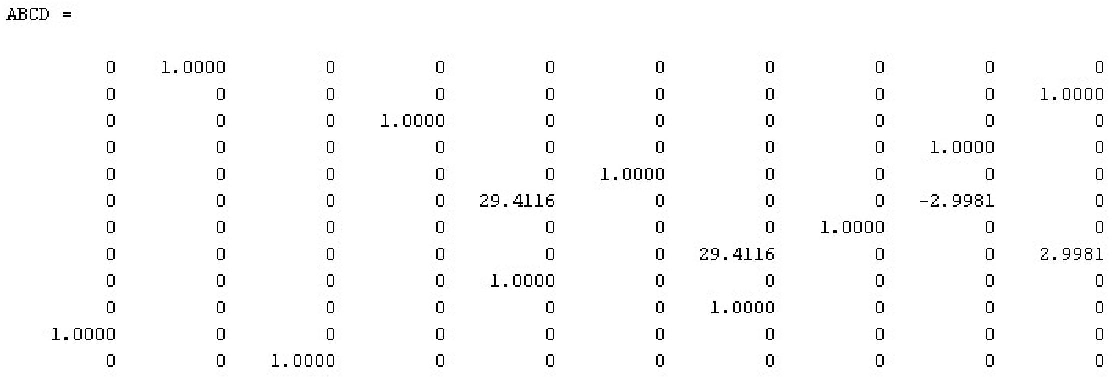
The first step is loading it in MATLAB: `load dslin_4outputs_ob.mat`

This command creates four variables. One of them (`xuyName`) contains information about inputs, outputs and states of the system:

```
xuyName =

motorX.s
motorX.v
motorZ.s
motorZ.v
cone.revolute_a.phi          states
cone.revolute_a.w
cone.revolute_b.phi
cone.revolute_b.w
u2
u1                           inputs
x5_Aphi
x7_Bphi
x1_Xs                        outputs
x3_Zs
```

Like it can be seen, now there are four outputs.

Other variable contain *A*, *B*, *C* and *D* matrixes, that is, four matrixes to do a complete representation of a system by means of its state-space model:



```
ABCD =

        0   1.0000        0        0        0        0        0        0        0        0
        0        0        0        0        0        0        0        0        0   1.0000
        0        0        0   1.0000        0        0        0        0        0        0
        0        0        0        0        0        0        0        0   1.0000        0
        0        0        0        0        0   1.0000        0        0        0        0
        0        0        0        0  29.4116        0        0        0  -2.9981        0
        0        0        0        0        0        0        0   1.0000        0        0
        0        0        0        0        0        0  29.4116        0        0   2.9981
        0        0        0        0   1.0000        0        0        0        0        0
        0        0        0        0        0        0   1.0000        0        0        0
   1.0000        0        0        0        0        0        0        0        0        0
        0        0   1.0000        0        0        0        0        0        0        0
```

Figure 5.2 *ABCD* variable in MATLAB.

Next step is extracting *A*, *B*, *C* and *D* matrixes from *ABCD* variable.

$$ABCD = \begin{pmatrix} \begin{pmatrix} \boldsymbol{A} \\ 8\text{x}8 \end{pmatrix} & \begin{pmatrix} \boldsymbol{B} \\ 8\text{x}2 \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{C} \\ 4\text{x}8 \end{pmatrix} & \begin{pmatrix} \boldsymbol{D} \\ 4\text{x}2 \end{pmatrix} \end{pmatrix}_{12\text{x}10}$$

Which is very easy using MATLAB:

```
A=ABCD(1:8,1:8);
B=ABCD(1:8,9:10);
C=ABCD(9:12,1:8);
D=ABCD(9:12,9:10);
```

6 Control design: controller + observer

In order to control the system, I have to observer four states, which I cannot measure.

Designing the observer is very easy because I have state-space model in MATLAB. This software has a command (`lqe`), which give me *L* observer gain matrix. I only need this matrix and *K* matrix to control and stabilizer the closed-loop system.

6.1 State observer

A state observer is an extension to a state-space model that provides feedback to control a system. A state observer is used on a system where direct access to the state is not possible. If the system is observable[16], then state observers can be designed to estimate the signals that cannot be measured. These signals are estimated with base in the outputs measures and the control.

Two of the basic parts of a control system are the plant and the controller. The "plant" is the black box model of the system that is to be controlled. The name originates from systems used to control factories or "plants." The controller is the subsystem designed to control the plant. The usual state space model for a plant can be written as:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

If this system is observable then the output, $y(t)$, can be used to steer the state of another state space model. This observer system is commonly denoted with a "hat": $\hat{x}(t)$ and $\hat{y}(t)$. The output of the observer system is subtracted from the output of the plant system; multiplied by a matrix L; and added to the state equation.

$$\dot{\hat{x}}(t) = A\hat{x}(t) - L[y(t) - \hat{y}(t)] + B\hat{u}(t)$$
$$\hat{y}(t) = C\hat{x}(t) + D\hat{u}(t)$$

For control purposes the output of the observer system is fed back to the input of both the observer and the plant:

$$\hat{u}(t) = u(t) = -K\hat{x}(t)$$

for some matrix *K*.

The observer equations become:

$$\dot{\hat{x}}(t) = A\hat{x}(t) - L[y(t) - \hat{y}(t)] - BK\hat{x}(t)$$
$$\hat{y}(t) = C\hat{x}(t) - DK\hat{x}(t)$$

or

$$\dot{\hat{x}}(t) = [A - BK]\hat{x}(t) - L[y(t) - \hat{y}(t)]$$
$$\hat{y}(t) = [C - DK]\hat{x}(t)$$

[16]More information about observability in Appendix A.

This is a diagram of the controlled system:

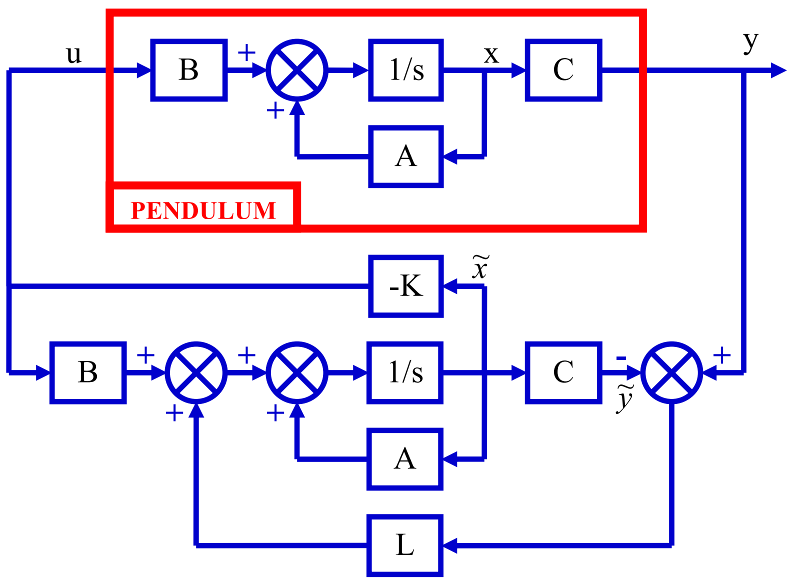


Figure 6.1 Control system by means of observed state feedback.

6.2 Design of the observer state

The first step is to check if the system is observable. If the system is not observable, when I connect the feedback by means of *K* and L matrix, the system may be not stable.

To check it, I have to calculate the matrix below:

$$OB = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \\ CA^4 \\ CA^5 \\ CA^6 \\ CA^7 \end{bmatrix}$$

System is controllable if and only if *rank (OB) = n = 8.*

I checked using MATLAB that the system is observable.

Before using `lqe` command in MATLAB, I have to determinate *Q, R* and *G* matrixes. These matrixes must be positive-semidefinite. Then, the simplest form is identity matrix. So:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now I can use `lqe` command to obtain *L* matrix: `L = lqe(A,G,C,Q,R)`

6.3 Simulation of the designed controller

After executing `lqe` command in MATLAB, I have *L* matrix. To check if this matrix stabilizes the system, I have to simulate. At the beginning, I will simulate in MATLAB, and after that I will make a simulation in Dymola.

6.3.1 Simulation in MATLAB[17]

I will use initial command to do the simulation. This command allows to obtain initial condition response of state-space models. Before that, I have to close the loop. The only different matrix is *A*. Closed-loop *A* matrix (*Aob*) is $Aob = A - L*C - B*K$

[17]MATLAB file can be seen in Appendix C.

Results for this simulation are shown below: initial conditions are: *Aphi = 10º = 0.17 rad. and Bphi = 4º = 0.07 rad.:*

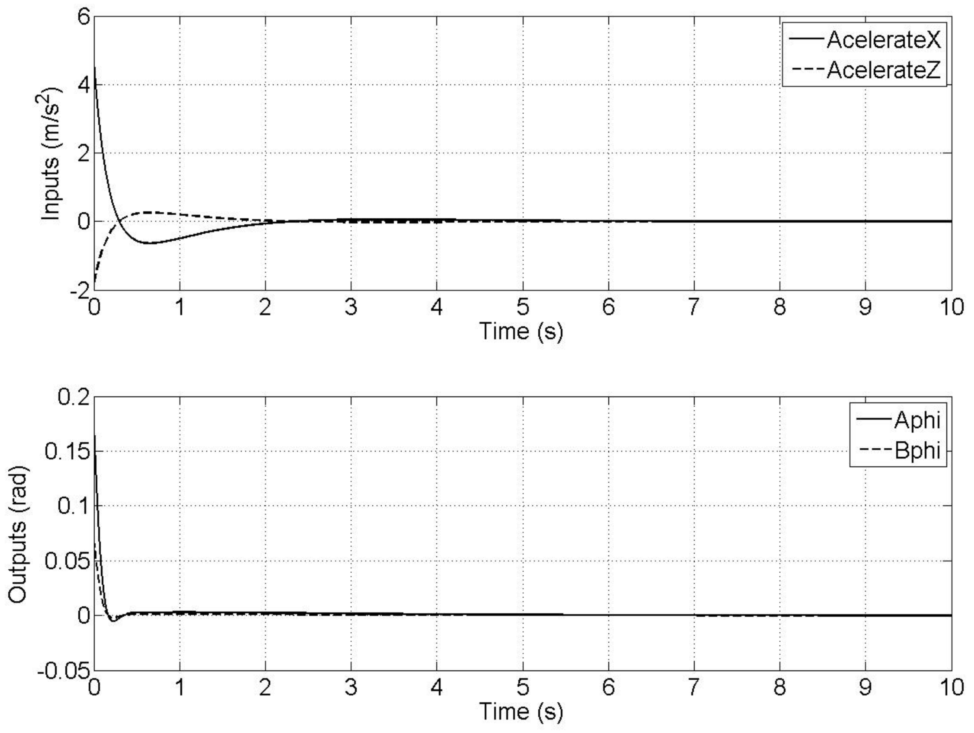


Figure 6.2 *Simulation obtained in MATLAB.*

I checked that results were like expected and that the response was stable.

6.3.2 Simulation in Dymola

The first step is creating a model with the system and the feedback. For it, I included:

- A pendulum model[18] (*pend_4output*), which represents pendulum dynamics.
- 8 sums[19] (*sum1…sum8*).
- 8 integrators[20] (*state1…state8*).
- 4 feedback[21] (*feedback1…feedback4*).
- 5 gain matrix[22] (*A_Matrix*, *B_Matrix, C_Matrix, K_Matrix, L_Matrix*), which represent matrixes of state-space model (*A*, *B* and *C*), feedback gain matrix (*K*) and observer gain matrix (*L*).

[18]This block was described at the paragraph 5.1.

[19]Included in Modelica.Blocks.Math.Sum.

[20]Included in Modelica.Blocks.Continuous.Integrator.

[21]Included in Modelica.Blocks.Math.Feedback.

[22]Included in Modelica.Blocks.Math.MatrixGain.

I introduced values obtained from MATLAB in *K_Matrix* and *L_Matrix* blocks
Modeling of control system + observer is shown below:

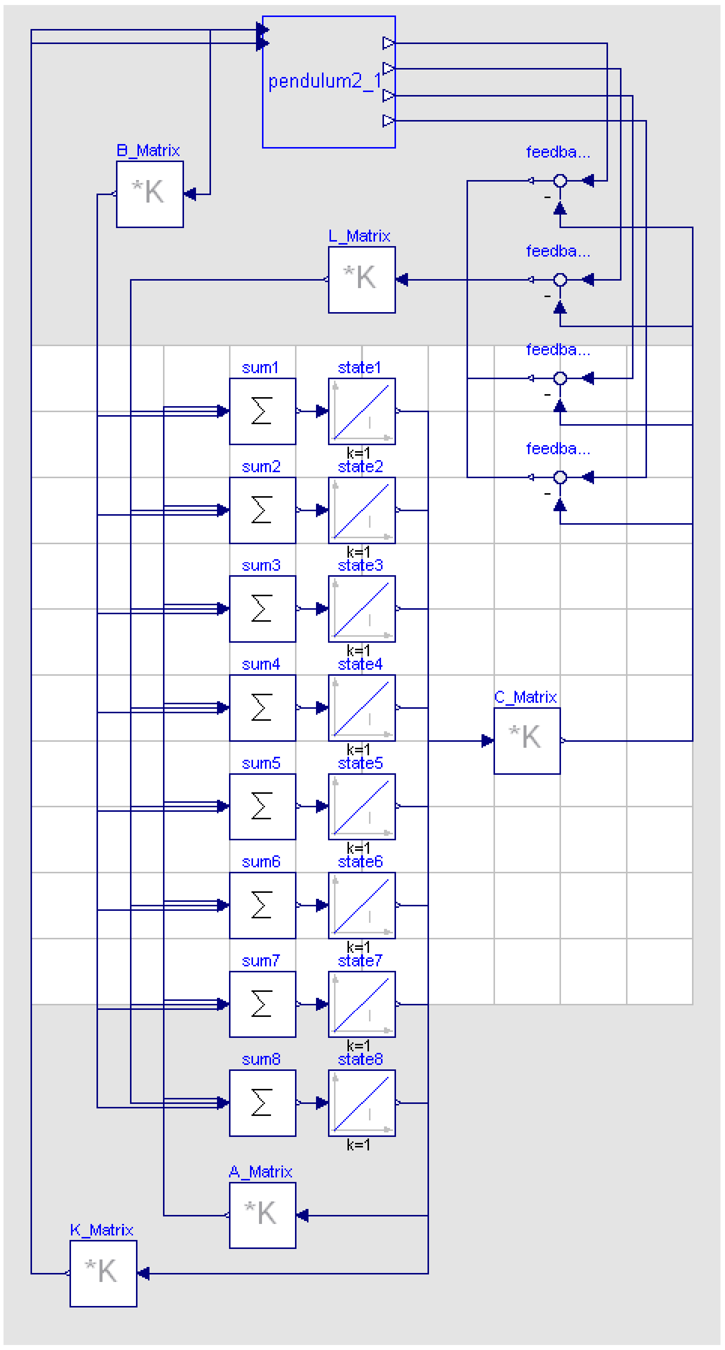


Figure 6.3 Dynamic model of the system + control + observer.

Previous model is similar to diagram shown in figure 6.1.

And the follow graphs show results (initial conditions are the same that before simulation with MATLAB):

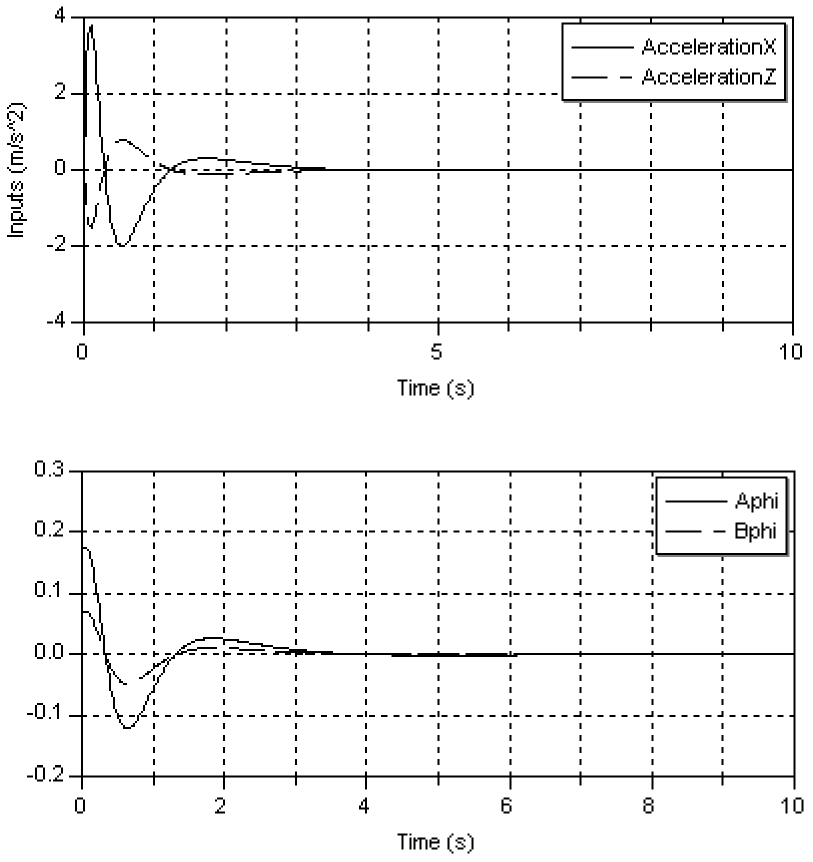


Figure 6.4 *Simulation obtained in Dymola.*

Now results are a little different with reference to MATLAB. Perhaps that is so because calculates in MATLAB were made with linear system and system is not linearized in Dymola. Nevertheless, results are correct because response is stable.

7 Kinematics for a Gantry-Tau parallel robot

Robot kinematics is the study of the motion (kinematics) of robots. In a kinematic analysis the position, velocity and acceleration of all the links are calculated without considering the forces that cause this motion. The relationship between motion, and the associated forces and torques is studied in robot dynamics.

Robot kinematics deals with aspects of redundancy, collision avoidance and singularity avoidance. While dealing with the kinematics used in the robots we deal each parts of the robot by assigning a frame of reference to it and hence a robot with many parts may have many individual frames assigned to each movable parts. For simplicity we deal with the single manipulator arm of the robot. Each frame is named systematically with numbers, for example the immovable base part of the manipulator is numbered 0, the first link joined to the base is numbered 1 and the next link 2 and similarly till n for the last nth link.

Robot kinematics is mainly of the following two types: forward kinematics and inverse kinematics. Forward kinematics is also known as direct kinematics. In forward kinematics, the length of each link and the angle of each joint is given and we have to calculate the position of any point in the work volume of the robot. In inverse kinematics, the length of each link and position of the point in work volume is given and we have to calculate the angle of each joint.

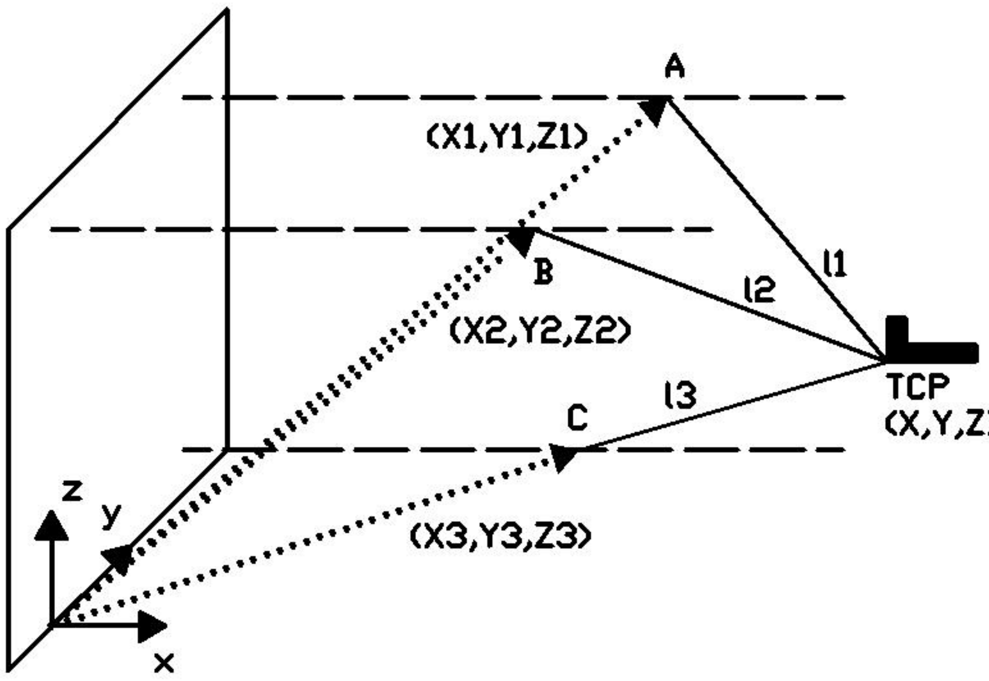


Figure 7.1 *Schematic Gantry-Tau structure.*

7.1 Inverse kinematics[23]

The inverse kinematics solves the following problem: "given the actual end effector pose, what are the corresponding joint positions?" The solution of the inverse problem is not always unique: the same end effector pose can be reached in several configurations, corresponding to distinct joint position vectors.

For the considered parallel robot the inverse kinematics problem is formulated as follow [11]. Calculate the location of points *A*, *B* and *C* along the linear tracks for a given *TCP* location. Let:

$A = X_1$

$B = X_2$

$C = X_3$

$TCP = [X\ Y\ Z]^T$

Here the parameters X_1, X_2 and X_3 are to be determined and can be found as the intersection between spheres with midpoints at $TCP\text{-}mp\_d_1$, $TCP\text{-}mp\_d_2$ and $TCP\text{-}mp\_d_3$ and the respective linear track.

The spherical equations can be written as follows:

$$(X_1 - X - mp\_d_{1x})^2 + (Y_1 - Y - mp\_d_{1y})^2 + (Z_1 - Z - mp\_d_{1z})^2 = l_1^2$$

$$(X_2 - X - mp\_d_{2x})^2 + (Y_2 - Y - mp\_d_{2y})^2 + (Z_2 - Z - mp\_d_{2z})^2 = l_2^2$$

$$(X_3 - X - mp\_d_{3x})^2 + (Y_3 - Y - mp\_d_{3y})^2 + (Z_3 - Z - mp\_d_{3z})^2 = l_3^2$$

where *mp_d* collects offsets in cart and plate.

Then we can determine the parameters:

$$X_1 = X + mp\_d_{1x} \pm \sqrt{l_1^2 - (Y_1 - Y - mp\_d_{1y})^2 - (Z_1 - Z - mp\_d_{1z})^2}$$

$$X_2 = X + mp\_d_{2x} \pm \sqrt{l_2^2 - (Y_2 - Y - mp\_d_{2y})^2 - (Z_2 - Z - mp\_d_{2z})^2}$$

$$X_3 = X + mp\_d_{3x} \pm \sqrt{l_3^2 - (Y_3 - Y - mp\_d_{3y})^2 - (Z_3 - Z - mp\_d_{3z})^2}$$

The sign before the root expression decides the configuration of the robot.

[23]MATLAB file can be seen in Appendix D.

7.2 Forward kinematics[24]

The forward kinematics solves the following problem: "given the joint positions, what is the corresponding end effector pose?" With Gantry-Tau structure, there are two possibilities for each given joint positions.

For the considered parallel robot the forward kinematics problem can be formulated as follows [11]. Calculate the location of the *TCP* for given *A*, *B* and *C*.

Three spheres with radius l_1, l_2 and l_3 describe all possible location for the *TCP* for *A*, *B* and *C*. The intersection points between the spheres describe the location of the *TCP*.

The midpoints of the spheres are:

$$P_{1c} = [X_1 \quad Y_1 \quad Z_1]^T - mp\_d_1$$

$$P_{2c} = [X_2 \quad Y_2 \quad Z_2]^T - mp\_d_2$$

$$P_{3c} = [X_3 \quad Y_3 \quad Z_3]^T - mp\_d_3$$

and the spherical equations are:

$$(X_1 - X)^2 + (Y_1 - Y)^2 + (Z_1 - Z)^2 = l_1^2$$

$$(X_2 - X)^2 + (Y_2 - Y)^2 + (Z_2 - Z)^2 = l_2^2$$

$$(X_3 - X)^2 + (Y_3 - Y)^2 + (Z_3 - Z)^2 = l_3^2$$

Mathematical symbolic software can solve the spherical equations, but produces a rather extensive solution. Proficient use of simplification rules is needed in order to simplify the solution. This problem is avoided by solving the equations in two steps. First find the intersection between two of the spheres. The intersection is either a circle or a point. Ignore the point case for now. The intersection between the third sphere and one of the other forms of course also a circle. Derive the plane where this circle is located. Secondly the intersections of this plane and the first circle describe the possible location for the TCP.

In the solution below the intersection circle between spheres with midpoints at *B* and *A* is calculated. All calculations are then done in a coordinate system with the z-axis pointing from *B* to *A*.

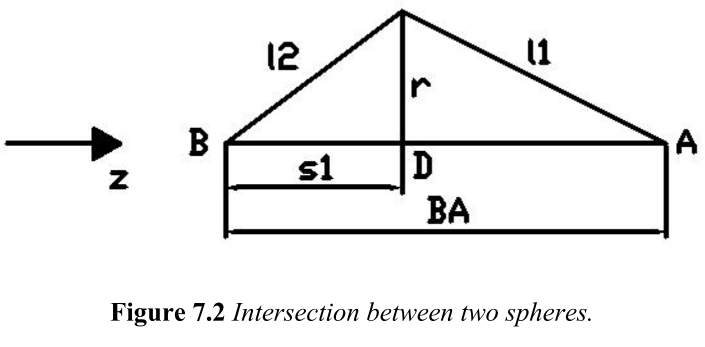


Figure 7.2 *Intersection between two spheres.*

[24]MATLAB file can be seen in Appendix E.

$$s_1 = \frac{l_2^2 + \left|\overrightarrow{BA}\right|^2 - l_1^2}{2\left|\overrightarrow{BA}\right|}, \quad r = \sqrt{l_2^2 - s_1^2}$$

Midpoint for the circle:

$$D = B + s_1 \frac{\overrightarrow{BA}}{\left|\overrightarrow{BA}\right|}, \quad s_2 = \frac{l_3^2 + \left|\overrightarrow{CA}\right|^2 - l_1^2}{2\left|\overrightarrow{CA}\right|}$$

A point on the plane:

$$E = C + s_2 \frac{\overrightarrow{CA}}{\left|\overrightarrow{CA}\right|}$$

The normal vector for the plane:

$$N = \overrightarrow{CA}$$

Deriving the rotation matrix:

$$\theta = \tan^{-1}\left(\frac{X_2 - X_1}{Y_1 - Y_2}\right), \quad \beta = \cos^{-1}\left(\frac{Z_1 - Z_2}{\left|\overrightarrow{BA}\right|}\right)$$

$$Rot_z = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad Rot_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{pmatrix}$$

$$Rot_{xz} = Rot_x Rot_z$$

The normal vector for the plane, *N*, and points *D* and *E* are transformed into a coordinate system with the z-axis pointing from *B* to *A*:

$$N_1 = \begin{pmatrix} N_x & N_y & N_z \end{pmatrix} = Rot_{xz} N$$

$$D_1 = \begin{pmatrix} x_d & y_d & z_d \end{pmatrix} = Rot_{xz} D$$

$$E_1 = \begin{pmatrix} x_e & y_e & z_e \end{pmatrix} = Rot_{xz} E$$

The spherical equations can now be written in the new coordinate system as the intersection between a circle and a sphere:

$$(x_r - x_d)^2 + (y_r - y_d)^2 = r^2$$

$$N_x(x_r - x_e) + N_y(y_r - y_e) + N_z(z_r - z_e) = 0$$

$$z_r = z_d$$

where:

$$\begin{pmatrix} x_r & y_r & z_r \end{pmatrix}^T = Rot_{xz} TCP$$

There are two solutions:

$$x_{r1} = \frac{N_y^2 x_d + N_x^2 x_e - N_{xy} T_2 - N_{xz} T_3 - N_y S}{N_q}$$

$$y_{r1} = \frac{N_y^2 x_e + N_x^2 x_d - N_{xy} T_1 - N_{yz} T_3 + N_x S}{N_q}$$

$$z_{r1} = z_d$$

$$x_{r2} = \frac{N_y^2 x_d + N_x^2 x_e - N_{xy} T_2 - N_{xz} T_3 + N_y S}{N_q}$$

$$y_{r2} = \frac{N_y^2 x_e + N_x^2 x_d - N_{xy} T_1 - N_{yz} T_3 - N_x S}{N_q}$$

$$z_{r2} = z_d$$

where:

$$T_1 = x_d - x_e, \quad T_2 = y_d - y_e, \quad T_3 = z_d - z_e$$

$$N_{xy} = N_x N_y, \quad N_{yz} = N_y N_z, \quad N_{xz} = N_x N_z$$

$$N_q = N_x^2 + N_y^2, \quad Q = N_y r, \quad R = N_y T_2 + N_z T_3$$

$$S = \sqrt{N_x^2 (r + T_1)(r - T_1) - 2 N_x T_1 R + Q^2 - R^2}$$

And the final solutions are:

$$TCP_1 = Rot_{xz}^{-1} \begin{pmatrix} x_{r1} \\ y_{r1} \\ z_{r1} \end{pmatrix}, \quad TCP_2 = Rot_{xz}^{-1} \begin{pmatrix} x_{r2} \\ y_{r2} \\ z_{r2} \end{pmatrix}$$

The configuration of the robot decides which solution is valid.

7.3 Velocity Jacobian

Reference [6] does a study about kinematics and dynamics of robot used in this paper. From it I am going to derive some equations to get Jacobian matrix.

Thanks to the Tau-configuration, the orientation of the end-effector plate is constant and the three DOFs of the robot are completely translational, so it is sufficient to consider one link per link cluster. The closure equation for link *i* is then:

$$L_i^2 - (\Delta X_i^2 + \Delta Y_i^2 + \Delta Z_i^2) = 0$$

Where $(\Delta X_i, \Delta Y_i, \Delta Z_i)^T$ is the vector along link *i*:

$$\Delta X_i = X + mp\_d_{ix} - X_i$$

$$\Delta Y_i = Y + mp\_d_{iy} - Y_i$$

$$\Delta Z_i = Z + mp\_d_{iz} - Z_i$$

Relating robot geometry with linear velocity of end-effector and joint velocity of carts:

$$\begin{pmatrix} \Delta X_1 & \Delta Y_1 & \Delta Z_1 \\ \Delta X_2 & \Delta Y_2 & \Delta Z_2 \\ \Delta X_3 & \Delta Y_3 & \Delta Z_3 \end{pmatrix}\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = \begin{pmatrix} \Delta X_1 & 0 & 0 \\ 0 & \Delta X_2 & 0 \\ 0 & 0 & \Delta X_3 \end{pmatrix}\begin{pmatrix} \dot{X}_1 \\ \dot{X}_2 \\ \dot{X}_3 \end{pmatrix} \equiv \Delta V = D\dot{Q}$$

Relating before equation with $V = J\dot{Q}$, we can deduce:

$$J^{-1} = D^{-1}\Delta = \begin{pmatrix} 1 & \Delta Y_1/\Delta X_1 & \Delta Z_1/\Delta X_1 \\ 1 & \Delta Y_2/\Delta X_2 & \Delta Z_2/\Delta X_2 \\ 1 & \Delta Y_3/\Delta X_3 & \Delta Z_3/\Delta X_3 \end{pmatrix}$$

(For $\Delta X_i \neq 0$, that is, not on the edge of workspace.)

The end-effector velocity can be obtained by inverting the inverse Jacobian matrix:

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = J\begin{pmatrix} \dot{X}_1 \\ \dot{X}_2 \\ \dot{X}_3 \end{pmatrix}$$

7.4 Acceleration

From [6] we can take some equations about acceleration of the parallel robot.

The end-effector acceleration is obtained by differentiating the closure equations twice by time:

$$\begin{pmatrix} \Delta X_1 & \Delta Y_1 & \Delta Z_1 \\ \Delta X_2 & \Delta Y_2 & \Delta Z_2 \\ \Delta X_3 & \Delta Y_3 & \Delta Z_3 \end{pmatrix}\begin{pmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{pmatrix} = \begin{pmatrix} \Delta X_1 & 0 & 0 \\ 0 & \Delta X_2 & 0 \\ 0 & 0 & \Delta X_3 \end{pmatrix}\begin{pmatrix} \ddot{X}_1 \\ \ddot{X}_2 \\ \ddot{X}_3 \end{pmatrix} - \begin{pmatrix} (\dot{X} - \dot{X}_1)^2 + \dot{Y}^2 + \dot{Z}^2 \\ (\dot{X} - \dot{X}_2)^2 + \dot{Y}^2 + \dot{Z}^2 \\ (\dot{X} - \dot{X}_3)^2 + \dot{Y}^2 + \dot{Z}^2 \end{pmatrix}$$

$$\begin{pmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{pmatrix} = J\begin{pmatrix} \ddot{X}_1 \\ \ddot{X}_2 \\ \ddot{X}_3 \end{pmatrix} - \Delta^{-1}\begin{pmatrix} (\dot{X} - \dot{X}_1)^2 + \dot{Y}^2 + \dot{Z}^2 \\ (\dot{X} - \dot{X}_2)^2 + \dot{Y}^2 + \dot{Z}^2 \\ (\dot{X} - \dot{X}_3)^2 + \dot{Y}^2 + \dot{Z}^2 \end{pmatrix}$$

(Vectors along links not linearly dependent for second equation.)

8 Simulation of full system (robot + controller + pendulum)

After getting forward and inverse kinematics in MATLAB, I had to translate these functions to Modelica language in order to make a model of the full system in Dymola.

The included blocks in the model are:

- A block[25], which represents pendulum dynamics and motion of robot hand (*pendulum_4out_pos*).
- A block[26], which represents controller + observer system (*ctrl_feed_ob*).
- A block[27], which represents forward kinematics of PKM (*ForwardKinematicsPKM*).
- A block[28], which represents inverse kinematics of PKM (*InverseKinematicsPKM*).
- 4 integrators[29] (*integrator1…integrator4*). In the *pendulum_4out_pos* block, I changed two acceleration actuators by two position actuators because at the robot, I will control the position instead of the acceleration.
- A constant[30] (*Z_desired*), which represents desired Z-coordinate.

Modeling of control system + pendulum + robot is shown below:

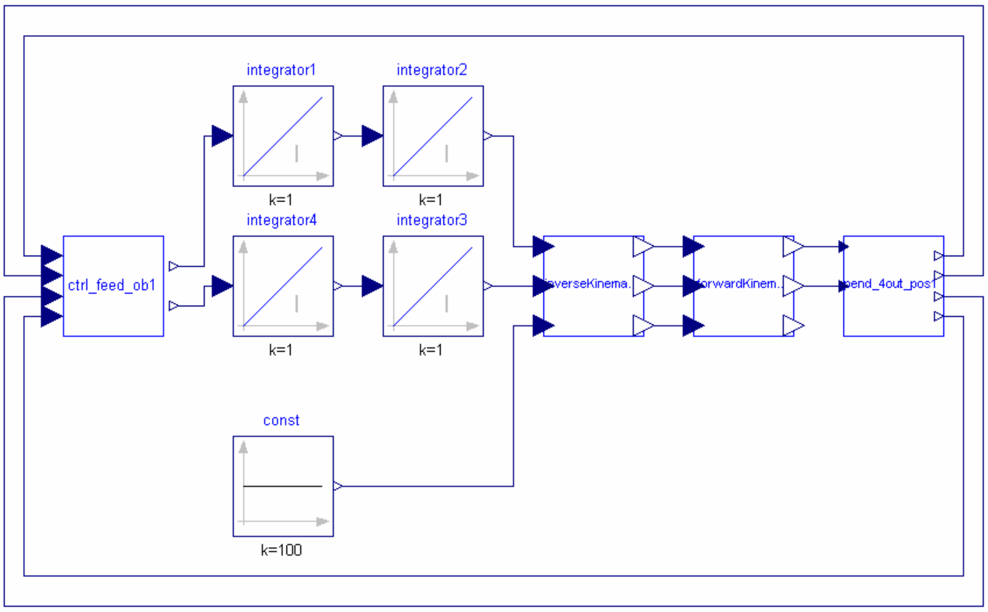


Figure 8.1 *Dynamic model of the system.*

[25]This block was described at the paragraph 5.1.

[26]This block was described at the paragraph 6.3.2.

[27]This block was described at the paragraph 7.2.

[28]This block was described at the paragraph 7.1.

[29]Included in Modelica.Blocks.Continuous.Integrator.

[30]Included in Modelica.Blocks.Sources.Constant.

And the follow graphs show results (initial conditions are: *Aphi = 10º = 0.17 rad. and Bphi = 4º = 0.07 rad.*):

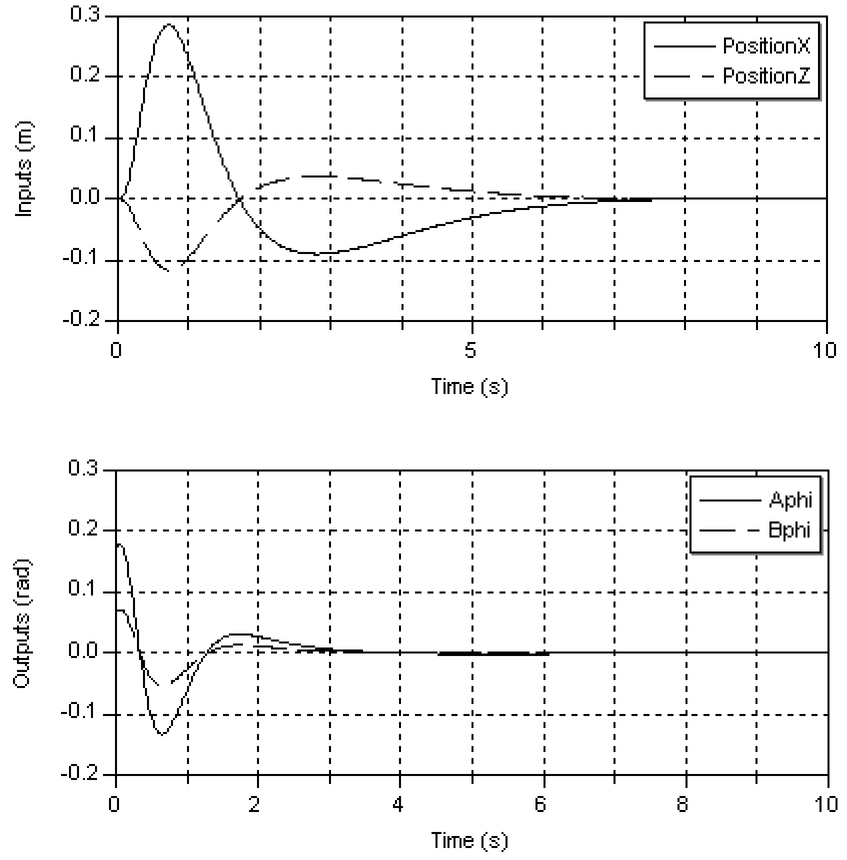


Figure 8.2 *Simulation obtained in Dymola.*

We can see results are correct. Response was similar to before response without including the robot.

8.1 Improving simulation. Motion in Z-coordinate

The aim of this step is testing how the system works if there is a motion along Z-coordinate. For doing it, I took the dynamic model that was explained at the paragraph 5.1. I replace two acceleration actuators by two position actuators (at the robot, I will control position instead of acceleration).

Besides, I added:

- A prismatic actuators[31] (*motorY*), which represents motions along Y-axis (vertical direction).
- A positioner[32] (*positionY*), which represents input signal to motion along vertical direction.
- A sine signal input[33] (*motion_Z_coordinate*), which represents the motion along vertical direction.

Diagram below shows the dynamic model of the system:

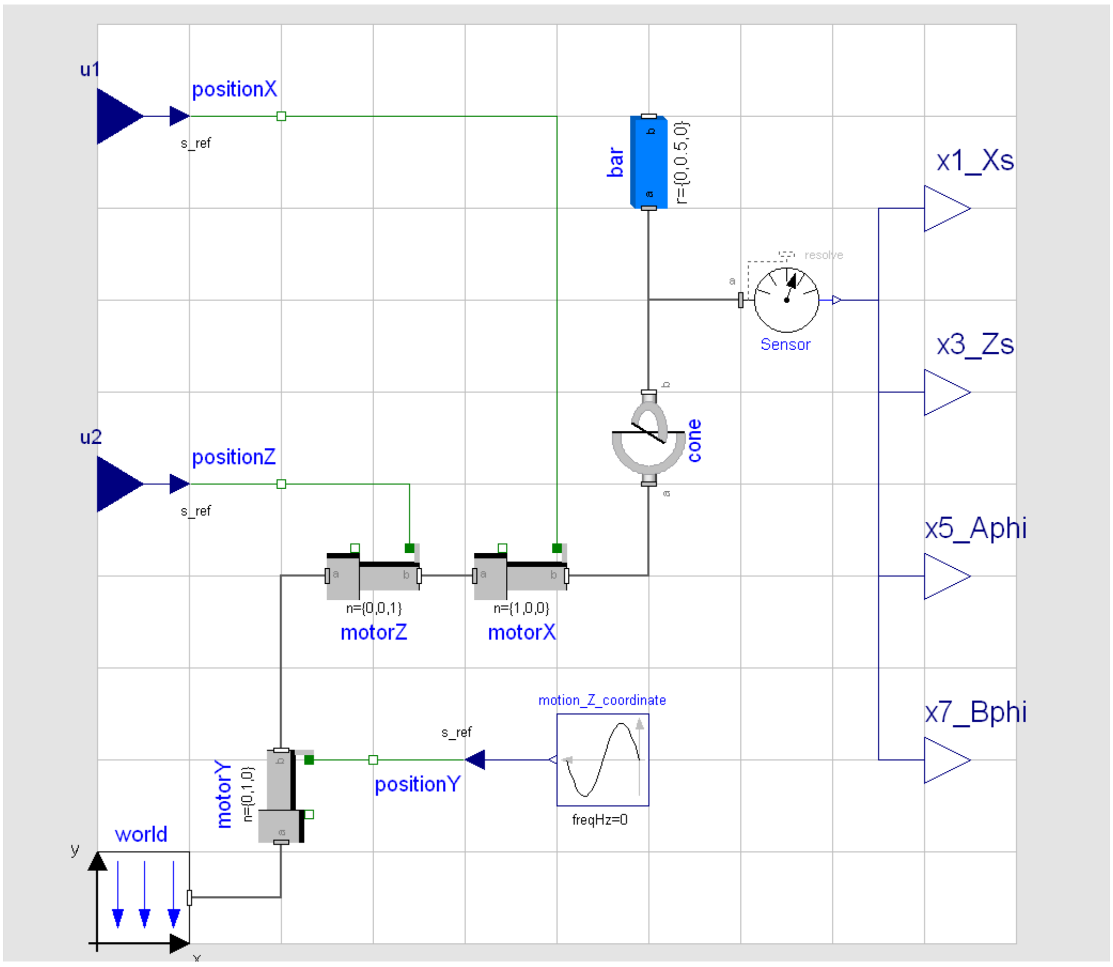


Figure 8.3 *Dynamic model of the system.*

[31]Included in Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic.

[32]Included in Modelica.Mechanics.Translational.Position.

[33]Included in Modelica.Mechanics.Blocks.Sources.Sine.

The input for motion along vertical direction was a sine signal like this (*amplitude = 0.4 m, frequency = 0.3 Hz*):

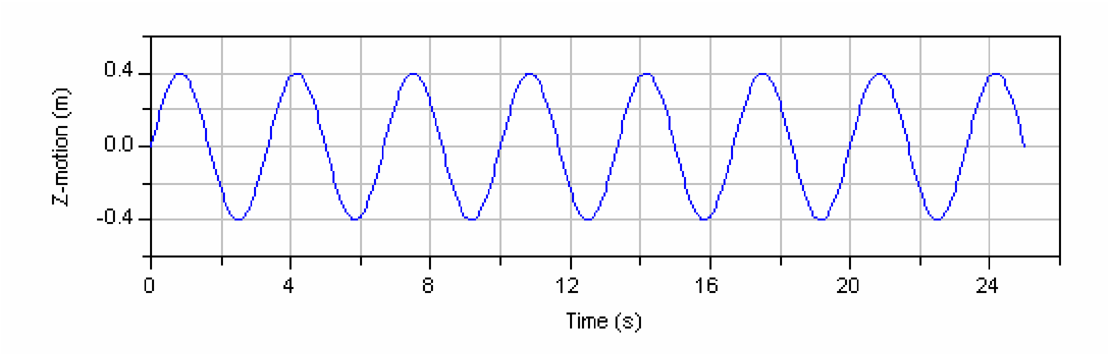


Figure 8.4 *Input for motion along Z-coordinate.*

And the follow graphs show results (initial conditions are: *Aphi = 10º = 0.17 rad. and Bphi = 4º = 0.07 rad.*):

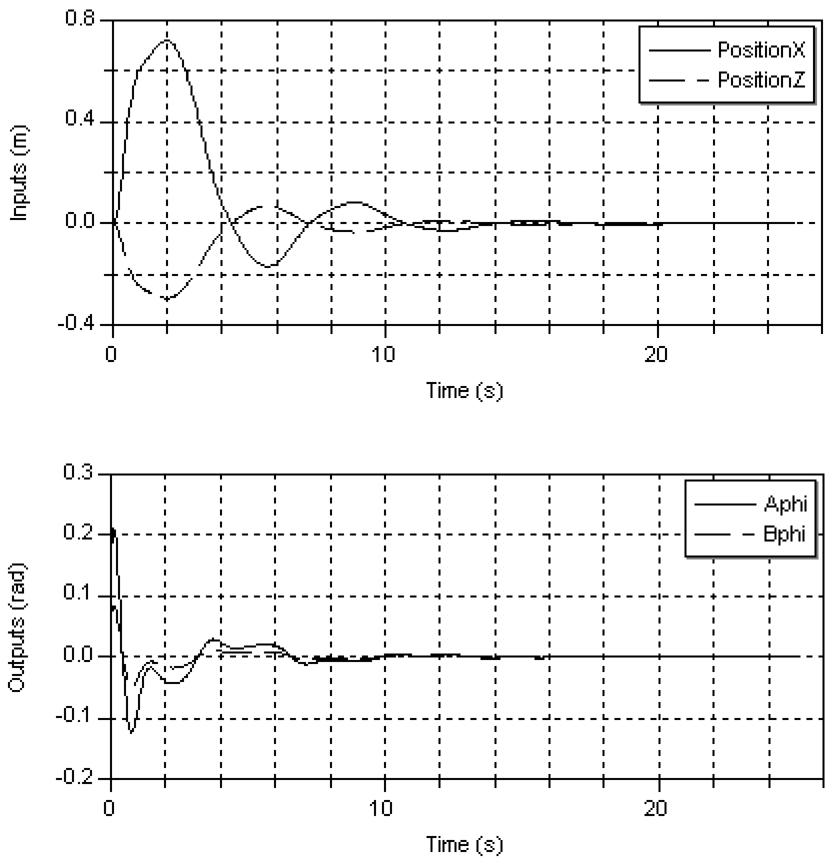


Figure 8.5 *Result of the simulation.*

You can observer how the system become stable.

8.2 Improving simulation. Add offsets at X and Y coordinates

The aim of this step is testing how the system works if I add an offset along X and/or Y-coordinate. For doing it, I took the dynamic model which was explained at the paragraph 8.

Besides, I added:

- 2 feedback[34] (*feedbackX* and *feedbackZ*).
- 2 constant[35] (*ofssetX* and *offsetZ*), which represents input signal to motion along vertical direction

Diagram below shows the dynamic model of the system:

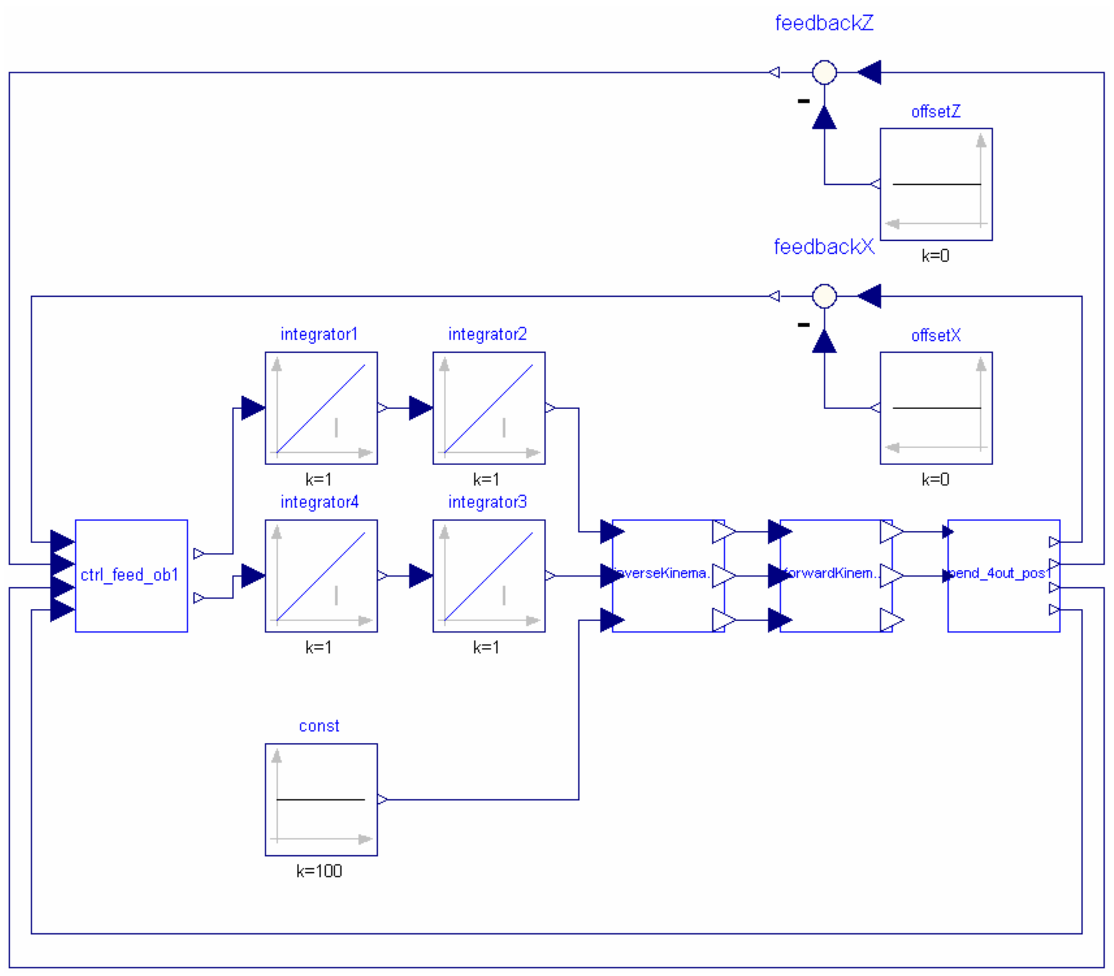


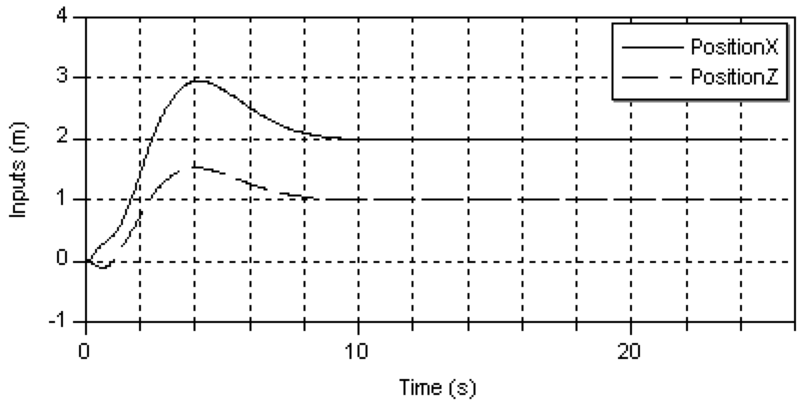
Figure 8.6 *Dynamic model of the system.*

[34]Included in Modelica.Blocks.Math.Feedback.

[35]Included in Modelica.Blocks.Sources.Constant.

Offsets I added for doing the simulation were 1 and 2.

And the follow graphs show results (initial conditions are: *Aphi = 10º = 0.17 rad. and Bphi = 4º = 0.07 rad.*):



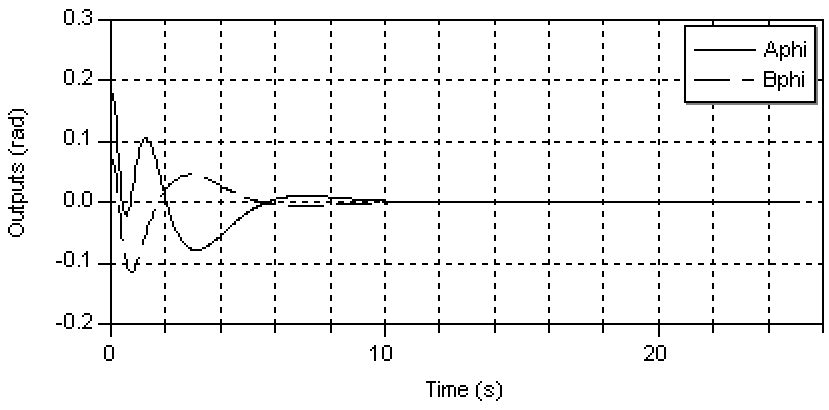


Figure 8.7 *Result of the simulation.*

You can observer how the system become stable.

9 Design of electronics for angle measurements

Hall-sensors are based in Hall effect. The Hall effect refers to the potential difference (Hall voltage) on opposite sides of a thin sheet of conducting or semiconducting material in the form of a 'Hall bar' (or a Van der Pauw element) through which an electric current is flowing, created by a magnetic field applied perpendicular to the Hall element. Edwin Hall discovered this effect in 1879.

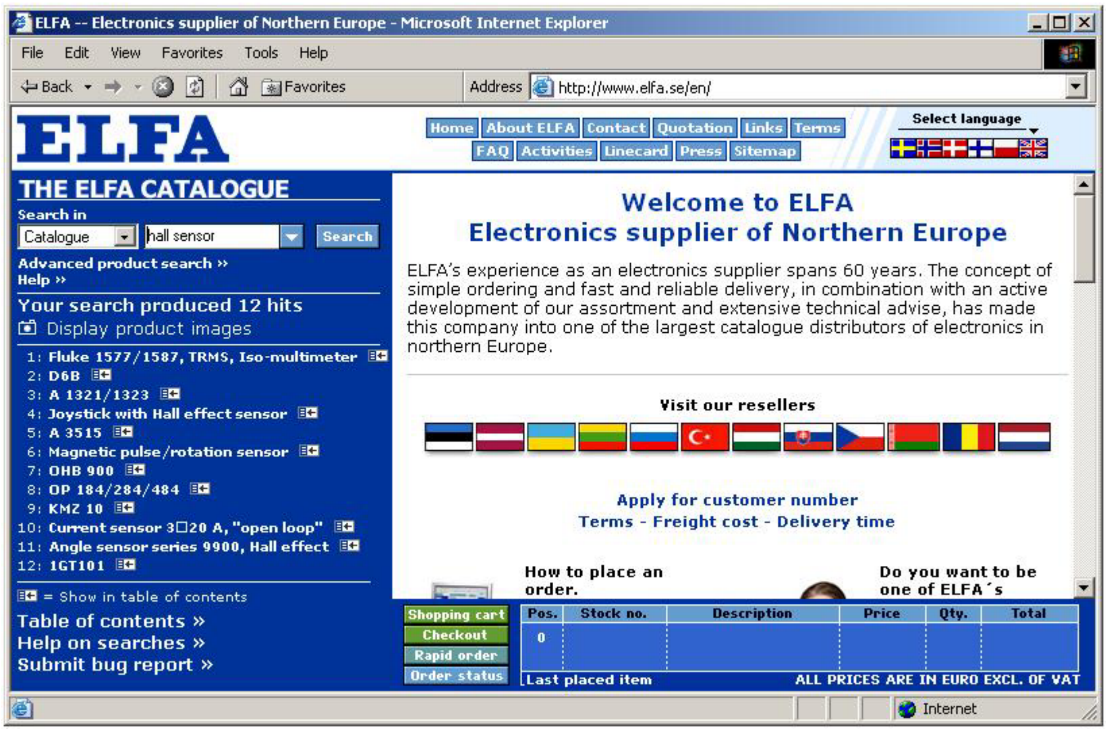
The ratio of the voltage created to the product of the amount of current and the magnetic field divided by the element thickness is known as the Hall coefficient and is a characteristic of the material of which the element is composed.

However, Hall-sensors give in a little signal, which is in relation to magnetic flux that is in relation to pendulum angle.

I will have to chose available sensor and test it. After that, perhaps I will have to amplify that signal using operational amplifiers.

9.1 Select a suitable Hall-sensor

The first step was searching a hall-sensor in ELFA [20], which is an electronics supplier. For doing it, I wrote *hall sensor* in its searcher. I got several results:


ELFA -- Electronics supplier of Northern Europe - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Favorites Address http://www.elfa.se/en/
ELFA
Home About ELFA Contact Quotation Links Terms FAQ Activities Linecard Press Sitemap
Select language
THE ELFA CATALOGUE
Search in Catalogue hall sensor Search
Advanced product search »
Help »

Your search produced 12 hits
Display product images

1: Fluke 1577/1587, TRMS, Iso-multimeter
2: D6B
3: A 1321/1323
4: Joystick with Hall effect sensor
5: A 3515
6: Magnetic pulse/rotation sensor
7: OHB 900
8: OP 184/284/484
9: KMZ 10
10: Current sensor 3□20 A, "open loop"
11: Angle sensor series 9900, Hall effect
12: 1GT101

= Show in table of contents

Table of contents »
Help on searches »
Submit bug report »

Welcome to ELFA
Electronics supplier of Northern Europe

ELFA's experience as an electronics supplier spans 60 years. The concept of simple ordering and fast and reliable delivery, in combination with an active development of our assortment and extensive technical advise, has made this company into one of the largest catalogue distributors of electronics in northern Europe.

Visit our resellers
Apply for customer number
Terms - Freight cost - Delivery time
How to place an order.
Do you want to be one of ELFA´s
Shopping cart
Checkout
Rapid order
Order status

Pos.	Stock no.	Description	Price	Qty.	Total
0					

Figure 9.1 Results of the Hall-sensor search.

After checking every possibility, I took two sensors: *A 3515* [3] and *KMZ 10* [4]. Besides, both sensors were available in the laboratory, so I could test their way of working.

Finally I selected *A 3515* Hall-sensor[36] due to its characteristics of working and its simplicity for adapting its voltage output by means of only operational amplifiers.

9.2 Behaviour of *A 3515* Hall-sensor

Picture below shows set up I made to test working of the Hall-sensor:

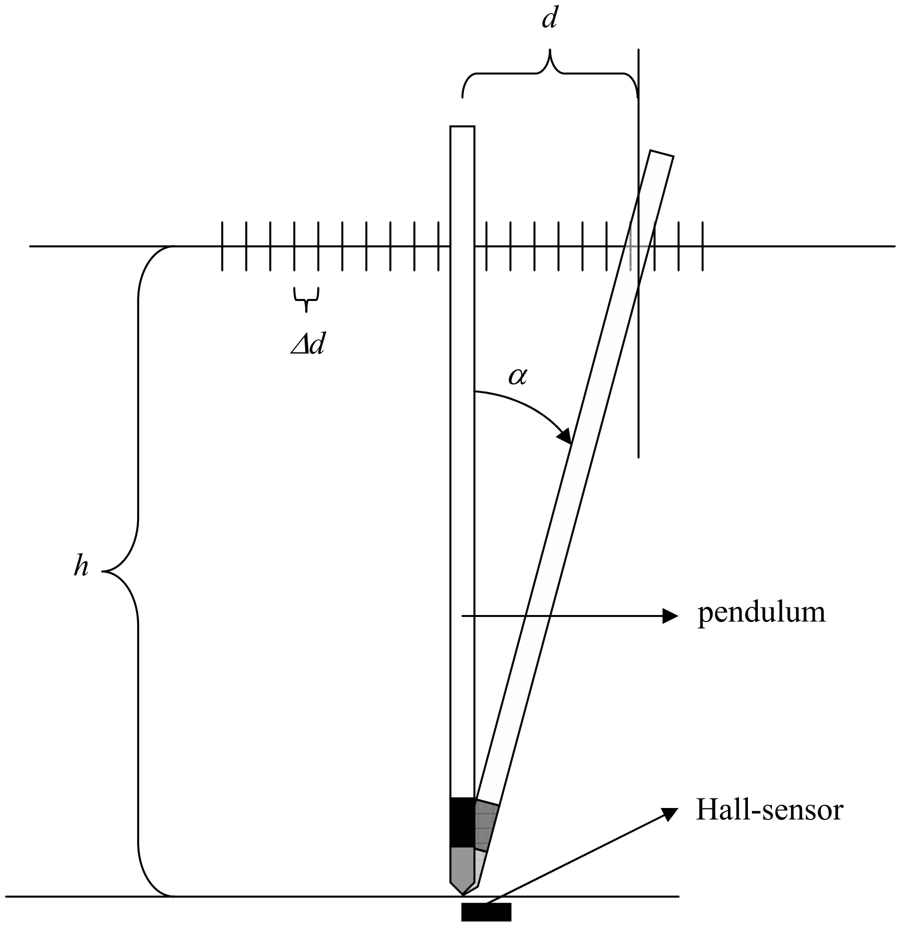


Figure 9.2 Set up for testing Hall-sensor.

After that, I followed below steps:

- I measured length of the pendulum (h = 235 mm).
- I connected 5 V DC to the supply voltage of the Hall-sensor.
- I connected a voltmeter at its output.
- I was taking measures while I moved pendulum with Δd = 5 mm.

[36]More information about *A 3515* Hall-sensor in Appendix B.

After taking every measure between –100 mm < α < +100 mm (–23° < α < +23°), I calculated the corresponding angle $\alpha = arctg\left(\frac{d}{h}\right)$ and I drew a graphic with results:

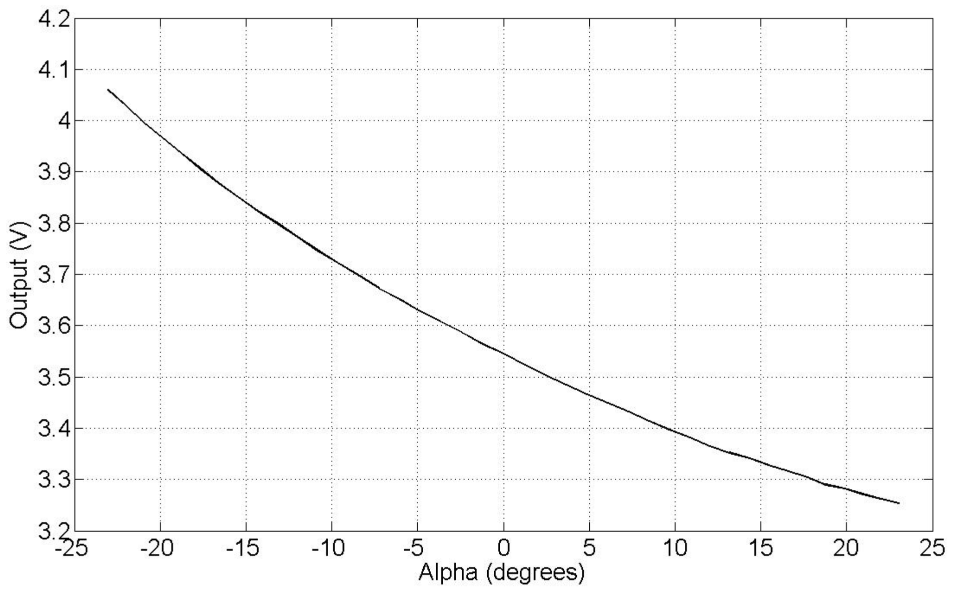


Figure 9.3 Result of Hall-sensor operating.

You can see on above picture that sensor response is nearly linear.

Now, I have to adapt that response to limits of date-acquisition system, that is, adjust sensor output to ±10 V, and 0 V when α = 0 V.

For doing it, I will place an amplifier stage next to Hall-sensor. Something like this:

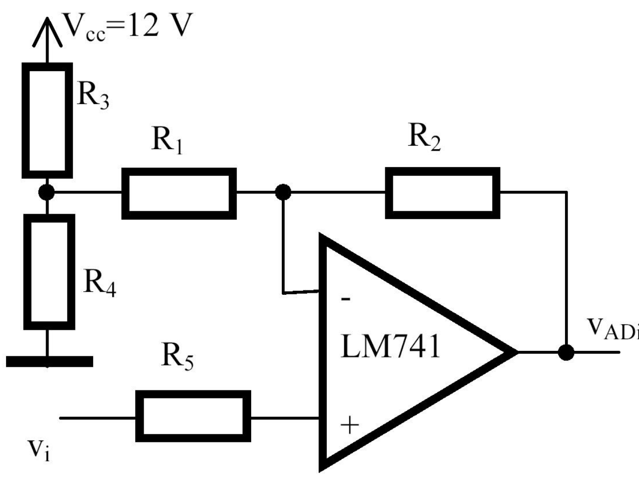


Figure 9.4 Stage with operational amplifier.

I used an amplifier with zero correction available in the laboratory. After adjusting offset and gain, I obtained this result:

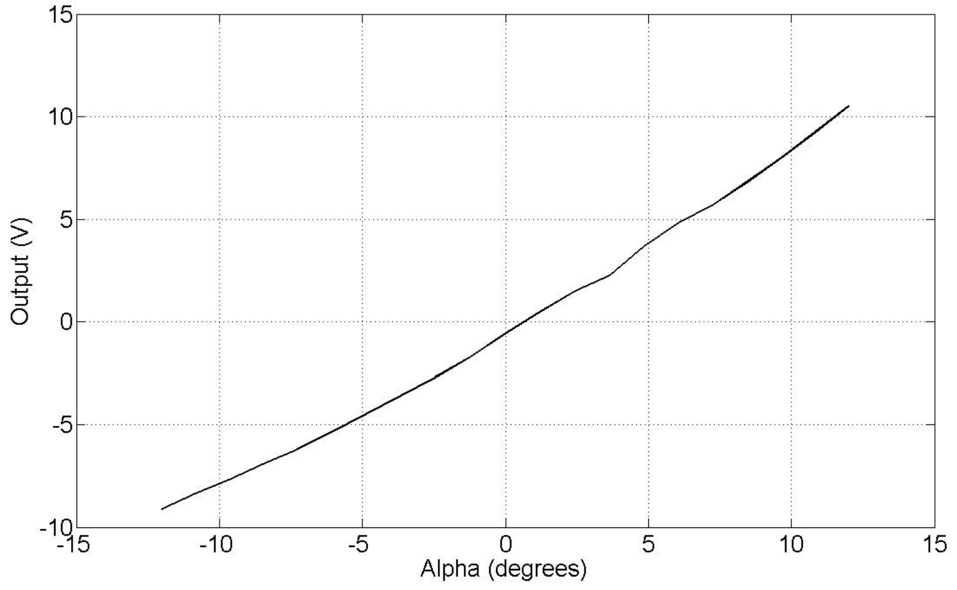


Figure 9.5 Result of Hall-sensor operating + operational amplifier.

Results were like expected and output is almost perfect. I will be able to correct some little deviations by means of computer.

Now, I obtain ±10 V when pendulum angle is ±12° as needed.

10 Experiment 1: balancing using serial robot (IRB140)

First experiment was balancing my pendulum with IRB140 robot, which is in the robotics laboratory.

Before doing experiment, I simulated the system using Dymola. In order to do it, I built a model of the manipulator + pendulum and I also obtained inverse kinematics model from it.

10.1 IRB 140

IRB 140 is a typical industrial serial robot with six DOFs. It is the most compact robots that ABB makes.

Its most important technical data are:

- Handing capacity: 5 kg.
- Number of axis: 6.
- Working range of axis movement:
    - Axis 1: 360°.
    - Axis 2: 200°.
    - Axis 3: 280°.
    - Axis 4: Unlimited (400° default).
    - Axis 5: 240°.
    - Axis 6: Unlimited (800° default).

Picture below show the robot structure:

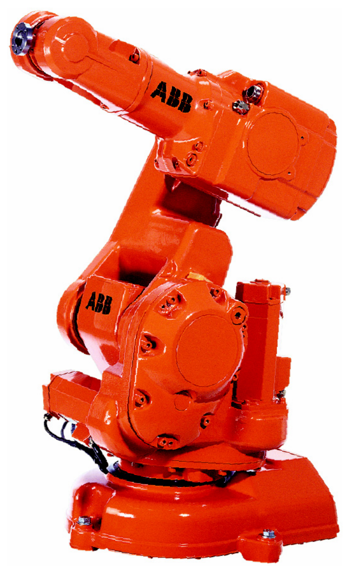


Figure 10.1 IRB 140.

10.2 Modeling of robot + pendulum

In Dymola, I made a modeling of physic system, that is, a model of robot + pendulum. For it, I included:

- 6 rotational actuators[37] (*J1 ... J6*), which represent motions of every joint.
- 6 positioners[38] (*position1 ... position6*), which represent input signal to every joint.
- 6 links[39] (*link1 ... link6*), which represent links of the robot.
- 1 universal joint[40] (*cone*), which represents two DOFs of the pendulum rotation.
- 1 body[41] (*pendulum*), which represents the pendulum.
- 1 absolute sensor[42] (*sensor*), which is able to give me every measure I need. These measures are: X and Y TCP-coordinates and angular position of both DOFs of the pendulum.
- 6 input ports[43] (*u1 … u6*), to connect input signals to system.
- 6 output ports[44] (*x1_Xs, x3_Zs, x5_Aphi* and *x7_Bphi*), to read output signals from system.
- Of course, I need a reference system[45]: *world*.

[37]Included in Modelica.Mechanics.MultiBody.Joints.ActuatedRevolute.

[38]Included in Modelica.Mechanics.Rotational.Position.

[39]Included in Modelica.Mechanics.MultiBody.Parts.BodyShape.

[40]Included in Modelica.Mechanics.MultiBody.Joints.Universal.

[41]Included in Modelica.Mechanics.MultiBody.Parts.BodyBox.

[42]Included in Modelica.Mechanics.MultiBody.Sensors.AbsoluteSensor.

[43]Included in Modelica.Blocks.Interfaces.RealInput.

[44]Included in Modelica.Blocks.Interfaces.RealOutput.

[45]Included in Modelica.Mechanics.MultiBody.World.

This one is the completed diagram:

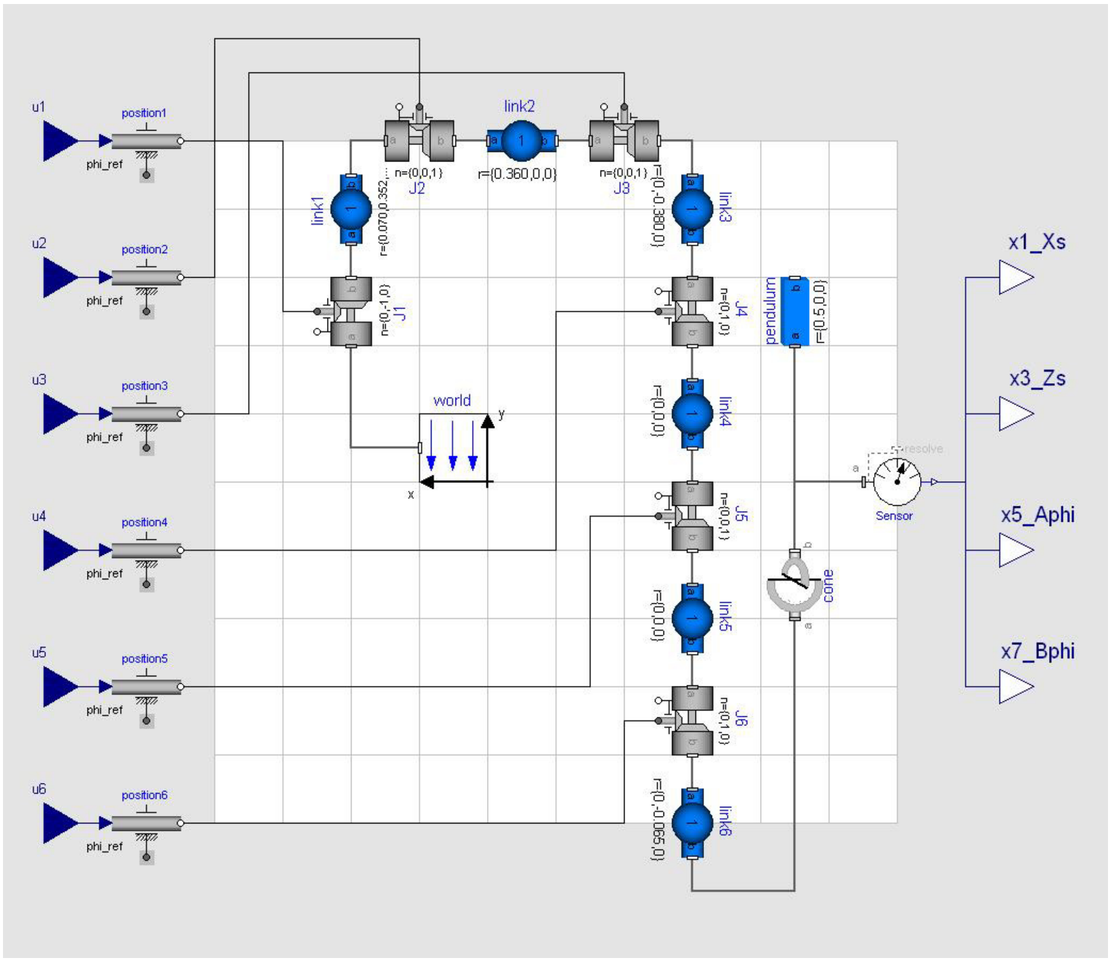


Figure 10.2 Dynamic model of the system.

10.3 Inverse kinematics of a serial robot

Unlike simulation of parallel robot, at this experiment I have built a modelling of the robot, so I do not need to derive forward kinematics model.

The inverse kinematics solves the following problem: "given the actual end effector pose, what are the corresponding joint positions?" The solution of the inverse problem is not always unique: the same end effector pose can be reached in several configurations, corresponding to distinct joint position vectors.

Although the general problem of inverse kinematics is quite difficult, it turns out that for manipulators having six joints, with the last three joints intersecting at a point, it is possible to decouple the inverse kinematics problem into two simpler problems, know respectively, as inverse position kinematics, and inverse orientation kinematics. To put it another way, for a six-DOF manipulator with a spherical wrist, the inverse kinematics problem may be separated into two simpler problems, namely first finding the position or the intersection of the wrist axes, hereafter called the wrist centre, and then finding the orientation of the wrist.

The IRB140 has exactly six DOFs and its last three joints axes intersect at a point *o*. We express follow equations as two sets of equations representing the rotational and positional equations:

$$R_6^0(q_1,...,q_6) = R$$

$$d_6^0(q_1,...,q_6) = d$$

where *d* and *R* are the given position and orientation of the tool frame.

Tool centre point is defined so:

$$TCP = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now assumption of a spherical wrist means that the axes z_4, z_5 and z_6 intersect at o and hence the origins o_4 and o_5 assigned by the D-H convention will always be at the wrist centre *o*. Often o_3 will be at *o* as well but this is not necessary for our subsequent development. The important point for this assumption for the inverse kinematics is that motion of the final three links about these axes will not change the position of *o*. The position of the wrist centre is thus a function of only the first three joint variables. Since the origin of the tool frame o_6 is simply a translation by a distance d_6 along the z_5 axis from *o*, the vector o_6 in the frame $o_0x_0y_0z_0$ are just:

$$o_6 - o = -d_6 Ra$$

Let p_c denote the vector from the origin of the base frame to the wrist centre. Thus in order to have the end-effector of the robot at the point *d* with the orientation of the end-effector given by $R = (r_{ij})$, it is necessary and sufficient that the wrist centre *o* be located at the point:

$$p_w = d - d_6 Ra$$

and that the orientation of the frame $o_0x_0y_0z_0$ with respect to the base be given by *R*. If the components of the end-effector position *d* are denoted d_x, d_y, d_z and the components of the wrist centre p_w are denoted by p_{xw}, p_{yw}, p_{zw} then the relationship is:

$$\begin{bmatrix} p_{xw} \\ p_{yw} \\ p_{zw} \end{bmatrix} = \begin{bmatrix} d_x - d_6 a_x \\ d_y - d_6 a_y \\ d_z - d_6 a_z \end{bmatrix}$$

Using before equation we may find the values of the first three joint variables. This determines the orientation transformation R_0^3, which depends only on these first three joint variables. We can now determine the orientation of the end-effector relative to the frame $o_3x_3y_3z_3$ from the expression:

$$R = R_0^3 R_3^6$$

as:

$$R_3^6 = (R_0^3)^{-1} R$$

The final three joints angles are then found as a set of Euler angles corresponding to R_3^6. Note that the right hand side of before equation is completely known since R is given and R_0^3 can be calculated once the first three joint variables are known.

The idea of kinematics decoupling is illustrated in figure below:

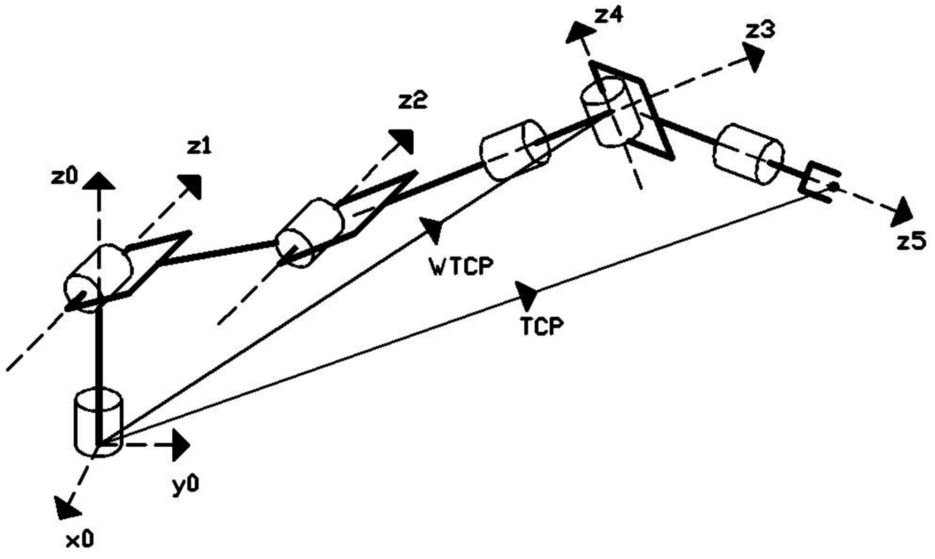


Figure 10.3 Kinematics decoupling.

10.4 Inverse kinematics of IRB140

I will divide the problem in two parts: at the beginning I will find q_1, q_2 and q_3, which will give me the position of *WCP*, and after that I will find q_4, q_5 and q_6, which will give me the orientation of *TCP*.

10.4.1 Find q_1, q_2 and q_3

I will use a geometric approach to find the variables q_1, q_2 and q_3, corresponding to wrist centre point (p_w).

Consider the manipulator shown in figure below, which represent the three first joints of the IRB140:

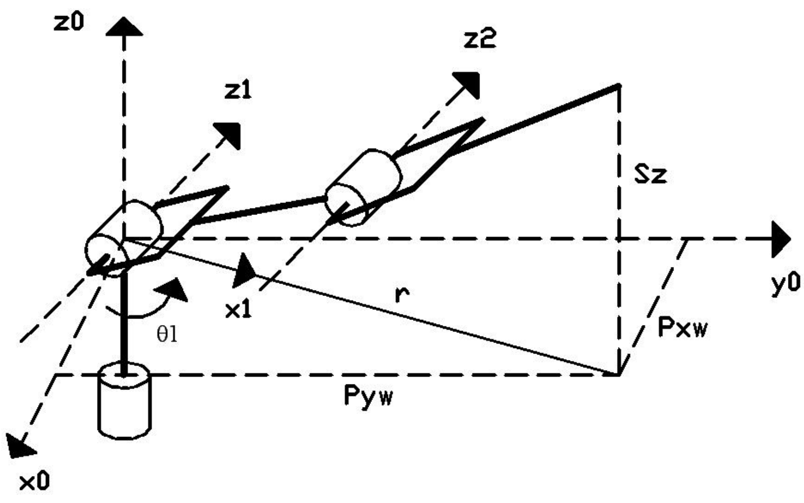


Figure 10.4 Elbow manipulator.

With the components of p_w denoted by p_{xw}, p_{yw}, p_{zw}, the project p_w onto the x_0y_0 plane as shown in figure below:

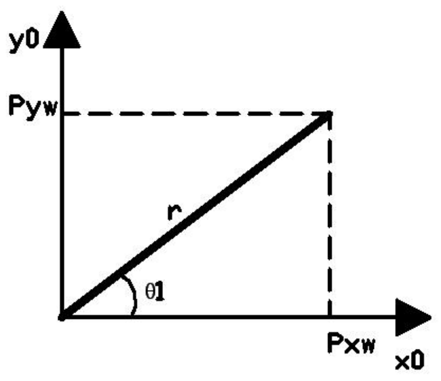


Figure 10.5 Projection of the wrist centre onto x_0-y_0 plane.

We see from this projection that:

$$\theta_1 = \tan^{-1}\left(\frac{p_{yw}}{p_{xw}}\right) = a\tan(p_{yw}, p_{xw})$$

where *atan(p_{yw},p_{xw})* denote the two argument arctangent function.

Note that a valid second solution for θ_1 is:

$$\theta_1 = \pi + a\tan(p_{yw}, p_{xw}) = a\tan(-p_{yw}, -p_{xw})$$

To find the angles θ_2 and θ_3, given θ_1, we consider the plane formed by the second and third links as shown in figure below:

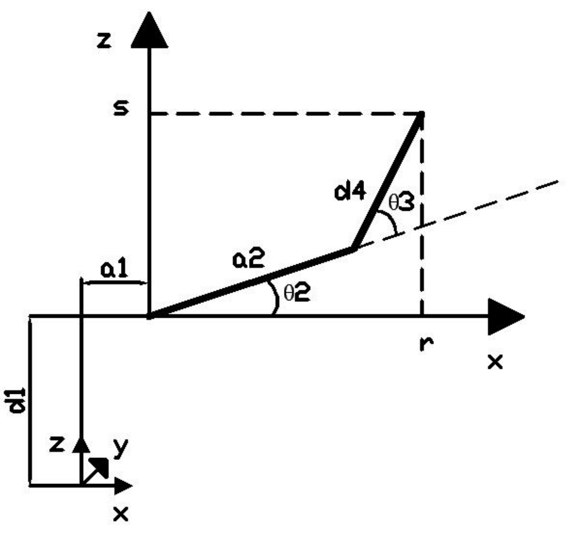


Figure 10.6 Projecting onto the plane formed by links 2 and 3.

Since the motion of links two and three is planar, the solution can be found as I explain following.

Using the Law of Cosines we see that the angle θ_3 is given by:

$$\cos\theta_3 = \frac{r^2 + s^2 - a_2^2 - d_4^2}{2a_2 d_4}$$

If I consider offset which is introduced by Denavit-Hartenberg parameters d_1 and a_1:

$$\cos\theta_3 = \frac{(p_{xw} + a_1\cos(\theta_1))^2 + (p_{yw} + a_1\sin(\theta_1))^2 + (P_{zw} - d_1)^2 - a_2^2 - d_4^2}{2a_2 d_4} = D$$

We could now determine θ_3 as:

$$\cos(\theta_3) = D \Rightarrow \theta_3 = \cos^{-1}(D)$$

However, a better way to find θ_3 is to notice that if:

$$\cos^2(\theta_3) + \sin^2(\theta_3) = 1 \Rightarrow D^2 + \sin^2(\theta_3) = 1 \Rightarrow \sin(\theta_3) = \pm\sqrt{1-D^2}$$

And, hence, θ_3 can be found by:

$$\tan(\theta_3) = \frac{\sin(\theta_3)}{\cos(\theta_3)} = \frac{\pm\sqrt{1-D^2}}{D} \Rightarrow$$

$$\theta_3 = \tan^{-1}\left(\frac{\pm\sqrt{1-D^2}}{D}\right) = a\tan(\pm\sqrt{1-D^2}, D)$$

The advantage of this latter approach is that both the elbow-up and elbow-down solutions are recovered by choosing the positive and negative signs respectively.

Similary θ_2 is given as:

$$\theta_2 = a\tan(s, r) - a\tan\left(d_4\sin(\theta_3), \quad a_2 + d_4\cos(\theta_3)\right) \Rightarrow$$

$$\theta_2 = a\tan\left(p_{zw} - d_1, \quad \sqrt{(p_{xw} + a_1\cos(\theta_1))^2 \quad + \quad (p_{yw} + a_1\sin(\theta_1))^2} \right) -$$

$$- a\tan\left(d_4\sin(\theta_3), \quad a_2 + d_4\cos(\theta_3) \right)$$

10.4.2 Find *q₄*, *q₅* and *q₆*

In the previous paragraph I used a geometric approach to solve the inverse position problem. This gives the values of the first three joint variables corresponding to a given position of the wrist origin. The inverse orientation problem is now one on finding the values of the final three joint variables corresponding to a given orientation with respect to the frame $o_3x_3y_3z_3$. For a spherical wrist this can be interpreted as the problem of finding a set of Euler angles corresponding to a given rotation matrix *R* as I pointed out in paragraph 10.3.

Equations to calculate q_4, q_5 and q_6 are:

$$R = R_0^3 R_3^6 \Rightarrow R_3^6 = (R_0^3)^{-1} R$$

where:

- R_3^6 is a matrix 3x3, which is a function of θ_4, θ_5 and θ_6.
- R_0^3 can be obtained from θ_1, θ_2 and θ_3 and Denavit-Hartenberg transformation matrixes A_0^1, A_1^2 and A_2^3:

$$T_0^3 = \begin{pmatrix} R_0^3 & d_0^3 \\ 0 & 1 \end{pmatrix} = A_0^1 * A_1^2 * A_2^3 \quad or \quad R_0^3 = R_0^1 * R_1^2 * R_2^3$$

- *R* is the rotation matrix of *TCP*:

$$TCP = \begin{pmatrix} R & d \\ 0 & 1 \end{pmatrix}$$

$(R_0^3)^{-1} R$ is a matrix 3x3 (as R_3^6). So, I have 9 equations and 9 unknown quantities to find θ_4, θ_5 and θ_6.

Following with the calculations, I obtain:

- $$R_3^6 = R_3^4 * R_4^5 * R_5^6 = \begin{pmatrix} C_4C_5C_6 - S_4S_6 & -C_4C_5S_6 - S_4C_6 & C_4S_5 \\ S_4C_5C_6 + C_4S_6 & -S_4C_5S_6 + C_4C_6 & S_4S_5 \\ -S_5C_6 & S_5S_6 & C_5 \end{pmatrix}$$

- $$(R_0^3)^{-1} R = B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

Therefore 9 equations are:

- $b_{11} = C_4C_5C_6 - S_4S_6$ *(Equation 1)*

- $b_{12} = -C_4 C_5 S_6 - S_4 C_6$ *(Equation 2)*
- $b_{13} = C_4 S_5$ *(Equation 3)*
- $b_{21} = S_4 C_5 C_6 + C_4 S_6$ *(Equation 4)*
- $b_{22} = -S_4 C_5 S_6 + C_4 C_6$ *(Equation 5)*
- $b_{23} = S_4 S_5$ *(Equation 6)*
- $b_{31} = -S_5 C_6$ *(Equation 7)*
- $b_{32} = S_5 S_6$ *(Equation 8)*
- $b_{33} = C_5$ *(Equation 9)*

From *equation 3 y 6* I find θ_4:

$$\left.\begin{aligned} b_{13} &= C_4 S_5 \\ b_{23} &= S_4 S_5 \end{aligned}\right\} \quad \frac{b_{23}}{b_{13}} = \frac{S_4}{C_4} = \tan(\theta_4) \quad \Rightarrow \quad \theta_4 = a\tan 2(b_{23}, b_{13})$$

From *equation 7 y 8* I find θ_6:

$$\left.\begin{aligned} b_{31} &= -S_5 C_6 \\ b_{32} &= S_5 S_6 \end{aligned}\right\} \quad \frac{b_{32}}{b_{31}} = -\frac{S_6}{C_6} = -\tan(\theta_6) \quad \Rightarrow \quad \theta_6 = a\tan 2(b_{32}, -b_{31})$$

From *equation 3 y 9* I find θ_6:

$$\left.\begin{aligned} b_{13} &= C_4 S_5 \\ b_{33} &= C_5 \end{aligned}\right\} \quad \frac{b_{13}}{b_{33}} = C_4 \tan(\theta_5) \quad \Rightarrow \quad \tan(\theta_5) = \frac{b_{13}}{b_{33} C_4} \quad \Rightarrow \quad \theta_5 = a\tan 2(b_{13}, b_{33} C_4)$$

11 Experiment 2: checking behaviour of angle measurement system

Second experiment was checking behaviour of the actual angle measurement system. For doing it, I did the same assemble that I described in part 9.2 of this report.

I checked both sensors (a sensor measures angle at X-direction and other one measures angle at Y-direction).

Results of both sensors were similar. These graphs show results of one of them:

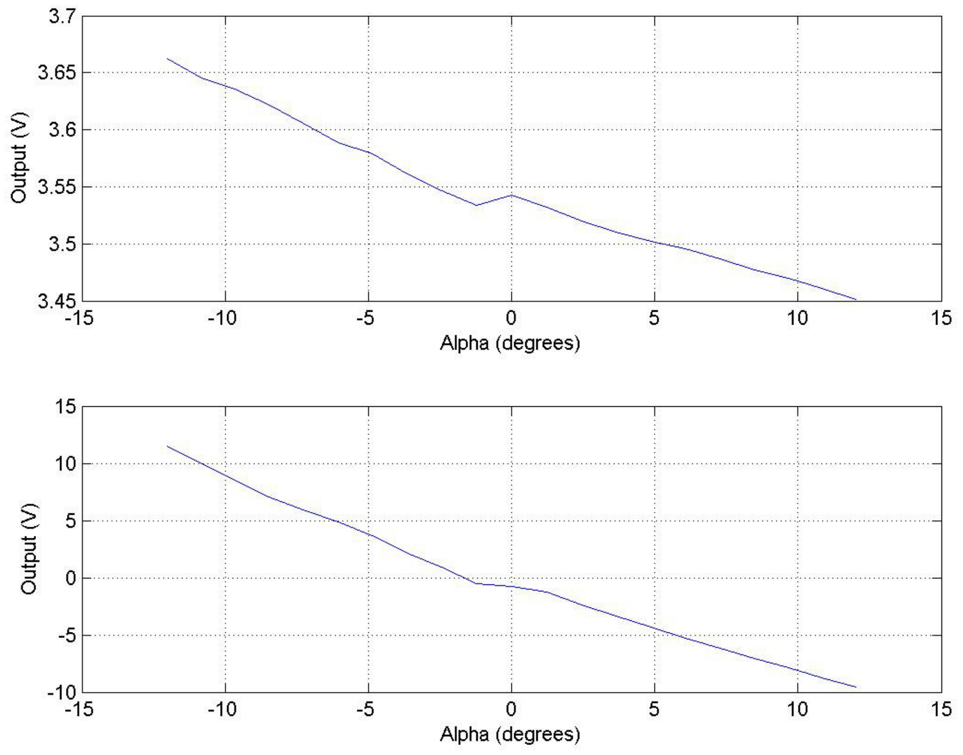


Figure 11.1 Result of Hall-sensor operating without amplifier (upper graph) and with amplifier (lower graph).

Results were not like expected due to a non-linearity, which appears around the up-position of the pendulum. This non-linearity did not appear when I tested the sensor with other plate and other pendulum, that is, not with the final plate and pendulum.

So, I think this non-linearity can be caused by a non-symmetry of the pendulum structure and for a hard system to get the measurements since it was done completely at hand.

12 Conclusion and future work

This master thesis allows me using and extending my knowledge in three big areas of engineering like are Automatic Control, Robotics and Electronic.

Dymola is very powerful software, which is able to realize modeling and simulation of a system.

Modeling is made very simple. It is indispensable to know no Newton's laws or solving long equations. Simply it is necessary to drag and drop the different available blocks.

Simulations are very convincing. They are not limited at a representation over coordinates axes, but you can see a 3D representation of the system dynamics. This is a good way to get an idea about motions that are produced by the pendulum.

Dymola's interface is easy and very intuitive. Besides, it allows to obtain a mathematical model of the system by a simple click of the mouse. This mathematical model is prepared to be manipulated by MATLAB, which is the software most used in engineering.

Dymola uses a high-level language called Modelica. I had never worked with this language, but it was not difficult because I knew C and MATLAB languages, which both are very similar.

I chose a linear-quadratic regulator (LQR) for controlling the system. It gave excellent results at the different simulations that I realized. Its design was very simple since Dymola provided me a mathematical model of the system (at state-space representation) and the calculation of feedback matrix *K* with MATLAB was immediate.

Robot used in this master thesis was a parallel kinematic manipulator. So, I have learnt more things about this kind of robot whose use is increasing on the industry.

The sensing problem of measuring the inclination angle of the pendulum was solved by using a contactless measuring setup based on Hall-effect sensors and a permanent magnet inside the pendulum. Sensors were stuck below the plate where the pendulum is supported.

Signal amplification was solved using two operational amplifiers in cascade for every sensor. Regulating gain and offset, I could adapt the signal to a suitable magnitude in order to the data acquisition card of the robot reads correctly that signal.

With respect to the future work, the most important thing would be checking if the control system designed in this master thesis works correctly in the actual robot and the pendulum is stabilized around the up position. It could be tested first in the table robot and after in the "big robot".

On the other hand, the set-up (plate + pendulum) can also be used in the other serial robots of the Robotics Laboratory, that is IRB 2400 and IRB 140. In this case, I would recommend doing before some simulations using Dymola software.

A State space

In control engineering, a state space representation is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations. To abstract from the number of inputs, outputs and states, the variables are expressed as vectors and the differential and algebraic equations are written in matrix form.

The state space representation (also known as the "time-domain approach") provides a convenient and compact way to model and analyse systems with multiple inputs and outputs. With *p* inputs and *q* outputs, we would otherwise have to write down *qxp* Laplace transforms to encode all the information about a system.

Unlike the frequency domain approach, the use of the state space representation is not limited to systems with linear components and zero initial conditions. "State space" refers to the space whose axes are the state variables. The state of the system can be represented as a vector within that space.

A.1 State variables

The internal state variables are the smallest possible subset of system variables that can represent the entire state of the system at any given time.

State variables must be linearly independent; a state variable cannot be a linear combination of other state variables. The minimum number of state variables required to represent a given system, *n*, is usually equal to the order of the system's defining differential equation.

If the system is represented in transfer function form, the minimum number of state variables is equal to the transfer function's denominator after it has been reduced to a proper fraction. In electronic systems, the number of state variables is the same as the number of energy storage elements in the circuit (capacitors and inductors).

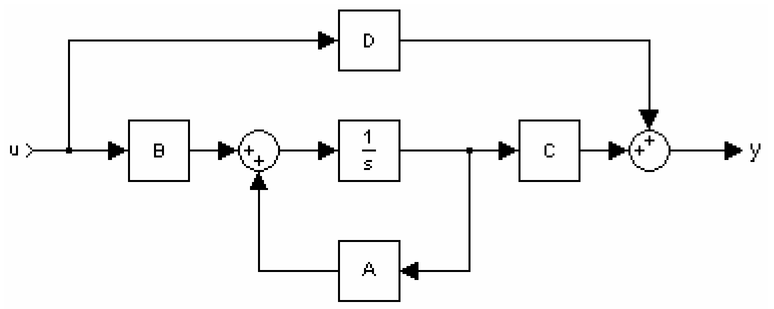


Figure A.1 *Typical state space model.*

A.2 Linear systems

The most general state space representation of a linear system with *p* inputs, *q* outputs and *n* state variables is written in the following form:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)$$

$$y(t) = C(t)x(t) + D(t)u(t)$$

where:

$$x(t) \in \Re^n; \quad y(t) \in \Re^q; \quad u(t) \in \Re^p$$

$$\dim[A(\cdot)] = nxn; \quad \dim[B(\cdot)] = nxp; \quad \dim[C(\cdot)] = qxn; \quad \dim[D(\cdot)] = qxp; \quad \dot{x}(t) = \frac{dx(t)}{dt}$$

x(·) is called the "state vector", *y(·)* is called the "output vector", *u(·)* is called the "input (or control) vector", *A(·)* is the "state matrix", *B(·)* is the "input matrix", *C(·)* is the "output matrix", and *D(·)* is the "feedthrough (or feedforward) matrix".

For simplicity, *D(·)* is often chosen to be the zero matrix, i.e. the system is chosen not to have direct feedthrough.

Notice that in this general formulation all matrixes are supposed time-variant, i.e. some or all their elements can depend from time. The time variable *t* can be a "continuous" one (i.e. $t \in \Re$) or a discrete one (i.e. $t \in \mathrm{Z}$): in the latter case the time variable is usually indicated as *k*.

Depending from the assumptions taken, the state-space model representation can assume the following forms:

System type	**State-space model**
Continuous time-invariant	$\dot{x}(t) = Ax(t) + Bu(t)$ $y(t) = Cx(t) + Du(t)$
Continuous time-variant	$\dot{x}(t) = A(t)x(t) + B(t)u(t)$ $y(t) = C(t)x(t) + D(t)u(t)$
Discrete time-invariant	$x(k+1) = Ax(k) + Bu(k)$ $y(t) = Cx(k) + Du(k)$
Discrete time-variant	$x(k+1) = A(k)x(k) + B(k)u(k)$ $y(k) = C(k)x(k) + D(k)u(k)$
Laplace domain of continuous time-invariant	$sX(s) = AX(s) + BU(s)$ $Y(s) = CX(s) + DU(s)$
Z-domain of discrete time-invariant	$zX(z) = AX(z) + BU(z)$ $Y(z) = CX(z) + DU(z)$

The stability of a time-invariant state-space model can easiest be determined by looking at the system's transfer function in factored form. It will then look something like this:

$$G(s) = k\frac{(s-z_1)(s-z_2)(s-z_3)}{(s-p_1)(s-p_2)(s-p_3)(s-p_4)}$$

The denominator of the transfer function is equal to the characteristic polynomial found by taking the determinant of:

$$\lambda(s) = | sI - A |$$

The roots of this polynomial (the eigenvalues) give in the poles in the system's transfer function. These poles can be used to analyse whether the system is asymptotically stable or marginally stable.

An alternative approach to determining stability, which does not involve calculating eigenvalues, is to analyse the system's Lyapunov stability. The zeros found in the numerator of *G(s)* can similarly be used to determine whether the system is minimum phase.

The system may still be input-output stable even though it is not internally stable. This may be the case if unstable poles are canceled out by zeros.

A.2.1 Controllability and observability

A continuous time-invariant state-space model is controllable if and only if:

$$rank(B \quad AB \quad ... \quad A^{n-1}B) = n$$

A continuous time-invariant state-space model is observable if and only if:

$$rank\begin{bmatrix} C \\ CA \\ ... \\ CA^{n-1} \end{bmatrix} = n$$

A.2.2 Transfer function

The "transfer function" of a continuous time-invariant state-space model can be derived in the following way

$$\dot{x}(t) = Ax(t) + Bu(t)$$

which after the Laplace transform give in:

$$sX(s) = AX(s) + BU(s)$$

$$(sI - A)X(s) = BU(s)$$

$$X(s) = (sI - A)^{-1}BU(s)$$

this is substituted for *X(s)*in the output equation:

$$Y(s) = CX(s) + DU(s)$$

$$Y(s) = C((sI - A)^{-1}BU(s)) + DU(s)$$

which results in the final transfer function:

$$Y(s) = G(s)U(s)$$

$$G(s) = C(sI - A)^{-1}B + D$$

Clearly *G(s)* must have *q* by *p* dimensionality, and thus has a total of *qp* elements. So for every input there are *q* transfer functions with one for each output. This is why the state-space representation can easily be the preferred choice for multiple-input, multiple-output (MIMO) systems.

A.2.3 Canonical realizations

Any given transfer function which is strictly proper can easily be transferred into state-space by the following approach:

Given a transfer function, expand it to reveal all coefficients in both the numerator and denominator. This should result in the following form:

$$G(s) = \frac{n_1 s^3 + n_2 s^2 + n_3 s + n_4}{s^4 + d_1 s^3 + d_2 s^2 + d_3 s + d_4}$$

The coefficients can now be inserted directly into the state-space model by the following approach:

$$\dot{x}(t) = \begin{bmatrix} -d_1 & -d_2 & -d_3 & -d_4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} n_1 & n_2 & n_3 & n_4 \end{bmatrix} x(t)$$

This state-space realization is called *controllable canonical form* because the resulting model is guaranteed to be controllable.

The transfer function coefficients can also be used to construct another type of canonical form:

$$\dot{x}(t) = \begin{bmatrix} -d_1 & 1 & 0 & 0 \\ -d_2 & 0 & 1 & 0 \\ -d_3 & 0 & 0 & 1 \\ -d_4 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} x(t)$$

This state-space realization is called observable canonical form because the resulting model is guaranteed to be observable.

A.2.4 Proper transfer functions

Transfer functions, which are only proper (and not strictly proper), can also be realised quite easily. The trick here is to separate the transfer function into two parts: a strictly proper part and a constant.

$$G(s) = G_{SP}(s) + G(\infty)$$

The strictly proper transfer function can then be transformed into a canonical state space realization using techniques shown above. The state space realization of the constant is trivially $y(t) = G(\infty)u(t)$. Together we then get a state space realization with matrices A, B and C determined by the strictly proper part, and matrix D determined by the constant.

Here is an example to clear things up a bit:

$$G(s) = \frac{s^2 + 3s + 3}{s^2 + 3s + 3} = \frac{s+2}{s^2 + 2s + 1} + 1$$

which gives in the following controllable realization:

$$\dot{x}(t) = \begin{bmatrix} -2 & -1 \\ 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 2 \end{bmatrix} x(t) + \begin{bmatrix} 1 \end{bmatrix} u(t)$$

Notice how the output also depends directly on the input. This is due to the $G(\infty)$ constant in the transfer function.

A.2.5 Feedback

A common method for feedback is to multiply the output by a matrix *K* and setting this as the input to the system: $u(t) = Ky(t)$. Since the values of *K* are unrestricted the values can easily be negated for negative feedback. The presence of a negative sign (the common notation) is merely a notational one and its absence has no impact on the end results.

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

becomes:

$$\dot{x}(t) = Ax(t) + BKy(t)$$

$$y(t) = Cx(t) + DKy(t)$$

solving the output equation for y(t) and substituting in the state equation results in:

$$\dot{x}(t) = (A + BK(I - DK)^{-1}C)x(t)$$

$$y(t) = (I - DK)^{-1}Cx(t)$$

The advantage of this is that the eigenvalues of *A* can be controlled by setting *K* appropriately through eigendecomposition of $(A + BK(I - DK)^{-1}C)$. This assumes that the open-loop system is controllable or that the unstable eigenvalues of *A* can be made stable through appropriate choice of *K*.

One fairly common simplification to this system is removing *D* and setting *C* to identity, which reduces the equations to:

$$\dot{x}(t) = (A + BK)x(t)$$

$$y(t) = x(t)$$

This reduces the necessary eigendecomposition to just *A + BK.*

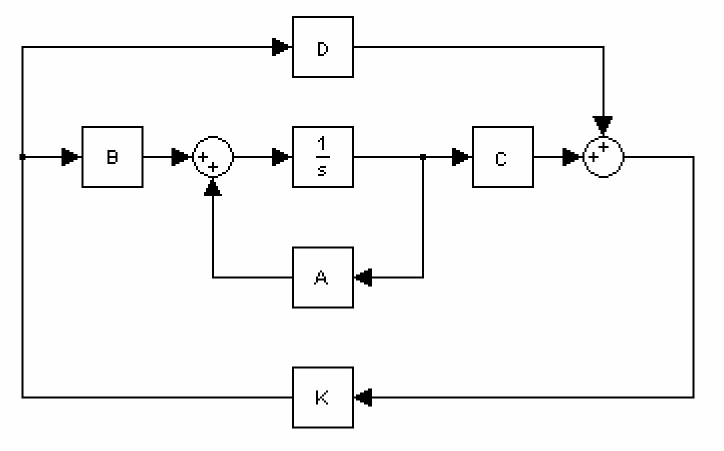


Figure A.2 Typical state space model with feedback.

A.2.6 Feedback with set point (reference) input

In addition to feedback, an input, *r(t)*, can be added such that:

$$u(t) = -Ky(t) + r(t)$$

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

becomes:

$$\dot{x}(t) = Ax(t) - BKu(t) + Br(t)$$

$$y(t) = Cx(t) - DKy(t) + Dr(t)$$

solving the output equation for *y(t)* and substituting in the state equation results in:

$$\dot{x}(t) = (A - BK(I + DK)^{-1}C)x(t) + B(I - K(I + DK)^{-1}D)r(t)$$

$$y(t) = (I + DK)^{-1}Cx(t) + (I + DK)^{-1}Dr(t)$$

One fairly common simplification to this system is removing *D*, which reduces the equations to:

$$\dot{x}(t) = (A - BKC)x(t) + Br(t)$$

$$y(t) = Cx(t)$$

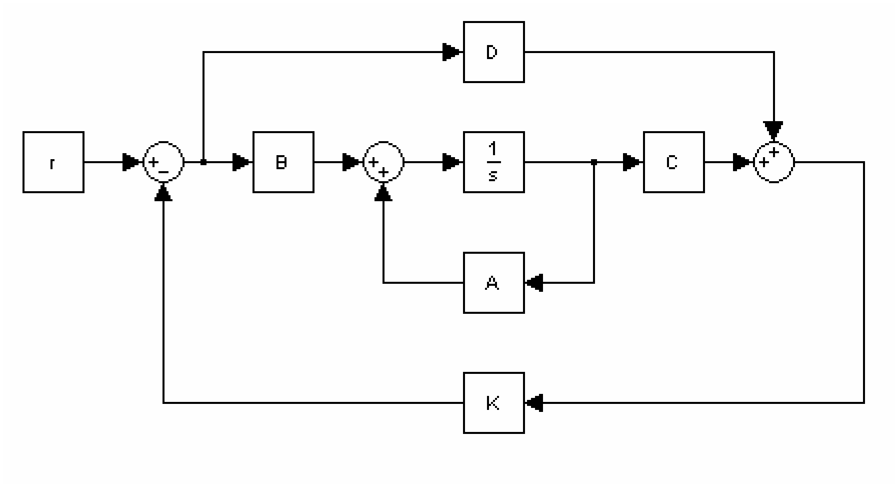


Figure A.3 *State feedback with set point.*

A.3 Non-linear systems

The more general form of a state space model can be written as two functions:

$$\dot{x}(t) = f(t, x(t), u(t))$$

$$y(t) = h(t, x(t), u(t))$$

The first is the state equation and the latter is the output equation.

If the function *f(·,·,·)* is a linear combination of states and inputs then the equations can be written in matrix notation like above.

The *u(t)* argument to the functions can be dropped if the system is unforced (i.e., it has no inputs).

B *A 3515* Hall-sensor

The *A3515* (and *A3516*) are sensitive, temperature-stable linear Hall-effect sensors with greatly improved offset characteristics. Ratiometric, linear Hall-effect sensors provide a voltage output that is proportional to the applied magnetic field and have a quiescent output voltage that is approximately 50% of the supply voltage. These magnetic sensors are ideal for use in linear and rotary position sensing systems in the harsh environments of automotive and industrial applications over extended temperatures to -40° C and +150° C. The *A3515* features an output sensitivity of 5 mV/G while the *A3516* has an output sensitivity of 2.5 mV/G.

Each BiCMOS monolithic circuit integrates a Hall element, improved temperature-compensating circuitry to reduce the intrinsic sensitivity drift of the Hall element, a small-signal high-gain amplifier, and a rail-to-rail low-impedance output stage.

A proprietary dynamic offset cancellation technique, with an internal high-frequency clock, reduces the residual offset voltage, which is normally caused by device overmolding, temperature dependencies, and thermal stress. This technique produces devices that have an extremely stable quiescent output voltage, are immune to mechanical stress, and have precise recoverability after temperature cycling. Many problems normally associated with low-level analog signals are minimized by having the Hall element and amplifier in a single chip. Output precision is obtained by internal gain and offset trim adjustments during the manufacturing process.

These devices are supplied in a 3-pin mini-SIP “U” package or a 3-pin ultra-mini-SIP “UA” package.

B.1 Features and absolute maximum ratings

These are the main features:

- Temperature-stable quiescent output voltage.
- Precise recoverability after temperature cycling.
- Output voltage proportional to applied magnetic field.
- Ratiometric rail-to-rail output.
- Improved sensitivity.
- 4.5 V to 5.5 V operation.
- Immune to mechanical stress.
- Small package size.
- Solid-state reliability.

And some of its absolute maximum ratings are:

- Supply voltage, V_{CC}: 8.0 V.
- Output voltage, V_O: 8.0 V.
- Output sink current, I_O: 10 mA.

- Magnetic flux density, B: unlimited.
- Operating temperature range (depend type), T_A:
  - Suffix E–: -40° C to +85° C.
  - Suffix L–: -40° C to +150° C.
- Storage temperature range, T_S: -65° C to +170° C.

B.2 Characteristics definitions

I am going to explain the most important characteristics of these Hall-sensors:

- **Quiescent voltage output:** in the quiescent state (no magnetic field), the output is ideally equal to one-half of the supply voltage over the operating voltage and temperature range ($V_{OQ} \approx V_{CC}/2$). Due to internal component tolerances and thermal considerations, there is a tolerance on the quiescent voltage output and on the quiescent voltage output as a function of supply voltage and ambient temperature. For purposes of specification, the quiescent voltage output as a function of temperature is defined as:

  $$\Delta V_{OQ(\Delta T)} = \frac{V_{OQ(TA)} - V_{OQ(25°C)}}{Sens_{(25°C)}}$$

  This calculation gives in the device's equivalent accuracy, over the operating temperature range, in gauss.

- **Sensitivity:** the presence of a south-pole magnetic field perpendicular to the package face (the branded surface) will increase the output voltage from its quiescent value toward the supply voltage rail by an amount proportional to the magnetic field applied. Conversely, the application of a north pole will decrease the output voltage from its quiescent value. This proportionality is specified as the sensitivity of the device and is defined as:

  $$Sens = \frac{V_{O(500G)} - V_{O(-500G)}}{1000G}$$

  The stability of sensitivity as a function of temperature is defined as:

  $$\Delta Sens_{(\Delta T)} = \frac{Sens_{(TA)} - Sens_{(25°C)}}{Sens_{(25°C)}} \times 100\%$$

- **Ratiometry:** the *A3515xU*, *A3515xUA*, *A3516xU*, and *A3516xUA* feature a ratiometric output. The quiescent voltage output and sensitivity are proportional to the supply voltage (ratiometric).

  The per cent ratiometric change in the quiescent voltage output is defined as:

  $$\Delta V_{OQ(\Delta V)} = \frac{V_{OQ(VCC)} / V_{OQ(5V)}}{V_{CC} / 5V} \times 100\%$$

  and the per cent ratiometric change in sensitivity is defined as:

$$\Delta Sens_{(\Delta V)} = \frac{Sens_{(VCC)} / Sens_{(5V)}}{V_{CC} / 5V} \times 100\%$$

- **Linearity and symmetry:** the on-chip output stage is designed to provide a linear output to within 500 mV of either rail with a supply voltage of 5 V. This is equivalent to approximately ±800 gauss of ambient field. Although application of stronger magnetic fields will not damage these devices, it will force the output into a non-linear region. Linearity in per cent is measured and defined as:

$$Lin+ = \frac{V_{O(500G)} - V_{OQ}}{2(V_{O(250G)} - V_{OQ})} \times 100\%$$

$$Lin- = \frac{V_{O(-500G)} - V_{OQ}}{2(V_{O(-250G)} - V_{OQ})} \times 100\%$$

  and output symmetry as:

$$Sym = \frac{V_{O(500G)} - V_{OQ}}{V_{OQ} - V_{O(-500G)}} \times 100\%$$

B.3 Applications information

Calibrated linear Hall devices, which can be used to determine the actual flux density presented to the sensor in a particular application, are available.

For safe, reliable operation, the output should not be pulled above the supply voltage or pulled below the device ground.

For optimum performance, a 0.1 mF capacitor between the supply and ground, and a 100 pF capacitor between the output and ground, should be added.

The ratiometric feature is especially valuable when these devices are used with an analog-to-digital converter. A/D converters typically derive their LSB step size by ratioing off a reference voltage line. If the reference voltage varies, the LSB will vary proportionally. This is a major error source in many sensing systems. The *A3515xU*, *A3515xUA*, *A3516xU*, and *A3516xUA* can eliminate this source of error by their ratiometric operation. Because their gain and offsets are proportional to the supply voltage, if they are powered from the A/D reference voltage, the sensor output voltage will track changes in the LSB value.

These devices can withstand infrequent temperature excursions, beyond the Absolute Maximum Ratings, to T_A = 170°C provided the junction temperature, T_J, does not exceed 200° C.

C MATLAB code for calculating LQR controller and simulating

```
% Clear variables and command window and close windows
clear all;
clc;
close all;

% Load file with mathematical model got with Dymola
load dslin_4outputs_ob.mat % 8 states, 4 outputs (Xs, Zs, Aphi and
Bphi)

% Show 8 states, 2 inputs and 4 outputs
xuyName

% Initial condition
Aphi=10; % [degrees]
Bphi=4; % [degrees]
Aphi_rad=Aphi*(pi/180); % [rad]
Bphi_rad=Bphi*(pi/180); % [rad]
x0=[0 0 0 0 Aphi_rad 0 Bphi_rad 0];

% Size of (ABCD): 12x10
A=ABCD(1:8,1:8);   % 8x8
B=ABCD(1:8,9:10);  % 8x2
C=ABCD(9:12,1:8);  % 4x8
D=ABCD(9:12,9:10); % 4x2

% Check if system is controllable
CON=[B A*B (A^2)*B (A^3)*B (A^4)*B (A^5)*B (A^6)*B (A^7)*B];
if rank(CON)==length(A)
    disp('System is controllable')
else
    disp('System is not controllable')
end

% LQR CONTROL DESIGN ************************************************
R=eye(2); % Identity matrix 2x2 (R matrix must be square with as many
columns as B)
Q=eye(8); % Identity matrix 8x8 (A and Q matrixes must be the same
size)
K = lqr(A,B,Q,R) % Calculate and show optimal gain matrix

% Closed loop matrix
Acl=A-B*K;
Bcl=B;
Ccl=C;
Dcl=D;

% Simulation closed loop
sysCL=ss(Acl,Bcl,Ccl,Dcl);
T=0:0.01:10; % Simulation time vector

[Y,T,X]=initial(sysCL,x0,T); % Response of state-space models with
initial condition
Y(:,1:2)=Y(:,1:2)*180/pi;    % Outputs (angles) in degrees

U=-K*X'; % Control law
```

```
% Graphs
figure
subplot(211);
plot(T,U(1,:));
hold on;
plot(T,U(2,:),'r');
title('Pendulum + feedback matrix K');
legend('AccelerateX','AceclerateZ');
%legend('AccelerateX','AceclerateY');
xlabel('Time (s)');
ylabel('Inputs (m/s^2)');
grid on;

subplot(212);
plot(T,Y(:,1));
hold on;
plot(T,Y(:,2),'r');
legend('Aphi','Bphi');
%legend('Alpha1','Alpha2');
xlabel('Time (s)');
%ylabel('Outputs (rad)');
ylabel('Outputs (degrees)');
grid on;

% Check if system is observable
OB=[C; C*A; C*(A^2); C*(A^3); C*(A^4); C*(A^5); C*(A^6); C*(A^7)];
if rank(OB)==length(A)
    disp('System is observable')
else
    disp('System is not observable')
end

% DESIGN CONTROL WITH STATE OBSERVER ********************************
G=eye(8); % Identity matrix 8x8 (A and G matrixes must have the same
number of rows)
Q=eye(8); % Identity matrix 8x8 (Q must be square with as many
columns as G)
R=eye(4); % Identity matrix 4x4 (R matrix must be square with as many
rows as C)
L=lqe(A,G,C,Q,R) % Calculate and show observer gain matrix

% Closed loop matrix
Aob=A-L*C-B*K;
Bob=B;
Cob=C;
Dob=D;

% Simulation closed loop with observer
sysOB=ss(Aob,Bob,Cob,Dob);

[Y,T,X]=initial(sysOB,x0,T); % Response of state-space models with
initial condition
Y(:,1:2)=Y(:,1:2)*180/pi;     % Outputs (angles) in degrees

% Graphs
figure
subplot(211);
plot(T,U(1,:));
hold on;
plot(T,U(2,:),'r');
title('Pendulum + feedback matrix K + observer matrix L');
```

```
%legend('AccelerateX','AceclerateZ');
legend('AccelerateX','AccelerateY');
xlabel('Time (s)');
ylabel('Inputs (m/s^2)');
grid on;

subplot(212);
plot(T,Y(:,1));
hold on;
plot(T,Y(:,2),'r');
%legend('Aphi','Bphi');
legend('Alpha1','Alpha2');
xlabel('Time (s)');
%ylabel('Outputs (rad)');
ylabel('Outputs (degrees)');
grid on;
```

D MATLAB code for calculating inverse kinematics for PKM

```
% INVERSE KINEMATICS

% Reference system: WORLD

% INPUT ARGUMENTS
% TCP -> 3x1 -> TCP = [TCPx; TCPy; TCPz] -> Tool Centre Point
% P1 -> 3x1 -> P1 = [P1x; P1y; P1z] -> Position of cart 1
% P2 -> 3x1 -> P2 = [P2x; P2y; P2z] -> Position of cart 2
% P3 -> 3x1 -> P3 = [P3x; P3y; P3z] -> Position of cart 3
% L -> 1x3 -> L = [L1 L2 L3] -> Lengths of the links 1,2 and 3
% mp_D -> 3x3 -> % Offset on the plate

% OUTPUT ARGUMENTS
% X1 -> 1x1 -> Position of cart 1 in X-direction
% X2 -> 1x1 -> Position of cart 2 in Y-direction
% X3 -> 1x1 -> Position of cart 3 in Z-direction

function [X1, X2, X3] = robot2(TCP,P1,P2,P3,L,mp_D)

% Length every link
L1 = L(1); % 1x1
L2 = L(2);
L3 = L(3);

% Offset on the plate
mp_d1 = mp_D(:,1); % 3x1
mp_d2 = mp_D(:,2);
mp_d3 = mp_D(:,3);

% Coordinates of the carts
Y1 = P1(2); % 1x1
Z1 = P1(3);
Y2 = P2(2);
Z2 = P2(3);
Y3 = P3(2);
Z3 = P3(3);

% Coordinates of the TCP
X=TCP(1); % 1x1
Y=TCP(2);
Z=TCP(3);

r1 = L1^2 - (Y1-Y-mp_d1(2))^2 - (Z1-Z-mp_d1(3))^2;
r2 = L2^2 - (Y2-Y-mp_d2(2))^2 - (Z2-Z-mp_d2(3))^2;
r3 = L3^2 - (Y3-Y-mp_d3(2))^2 - (Z3-Z-mp_d3(3))^2;

% Test if there is solution and calculate it
if (r1<0 | r2<0 | r3<0)
    error('No solution')
    return;
else % Two possible solutions
    X11 = X + mp_d1(1) + sqrt(r1);
    X21 = X + mp_d2(1) + sqrt(r2);
    X31 = X + mp_d3(1) + sqrt(r3);
    X1 = X + mp_d1(1) - sqrt(r1);
    X2 = X + mp_d2(1) - sqrt(r2);
    X3 = X + mp_d3(1) - sqrt(r3);
%{
    % Draw solutions
    figure()
    A = [Y2;Z2];B=[Y3;Z3];C=[Y1;Z1];

    % A 'blue star' show every cart position;
    plot3(X2,A(1),A(2),'b*', X3,B(1),B(2),'b*', X1,C(1),C(2),'b*')
    hold on
    plot3(X21,A(1),A(2),'b*', X31,B(1),B(2),'b*', X11,C(1),C(2),'b*')
    hold on

    % A 'blue square' show TCP position
    plot3(TCP(1),TCP(2),TCP(3),'b square')

    % One possibility
    plot3([X21 TCP(1)],[A(1) TCP(2)],[A(2) TCP(3)],'g','Linewidth',2)
    plot3([X31 TCP(1)],[B(1) TCP(2)],[B(2) TCP(3)],'g','Linewidth',2)
    plot3([X11 TCP(1)],[C(1) TCP(2)],[C(2) TCP(3)],'g','Linewidth',2)

    % Other possibility
    plot3([X2 TCP(1)],[A(1) TCP(2)],[A(2) TCP(3)],'r','Linewidth',2)
    plot3([X3 TCP(1)],[B(1) TCP(2)],[B(2) TCP(3)],'r','Linewidth',2)
    plot3([X1 TCP(1)],[C(1) TCP(2)],[C(2) TCP(3)],'r','Linewidth',2)

    % Show origin
    plot3([0 4],[0 0],[0 0],'k','Linewidth',3)
    plot3([0 4],[2 2],[0 0],'k','Linewidth',3)
    plot3([0 4],[1 1],[2 2],'k','Linewidth',3)

    grid on
    xlabel('X Axis')
    ylabel('Y Axis')
    zlabel('Z Axis')
%}
end
```

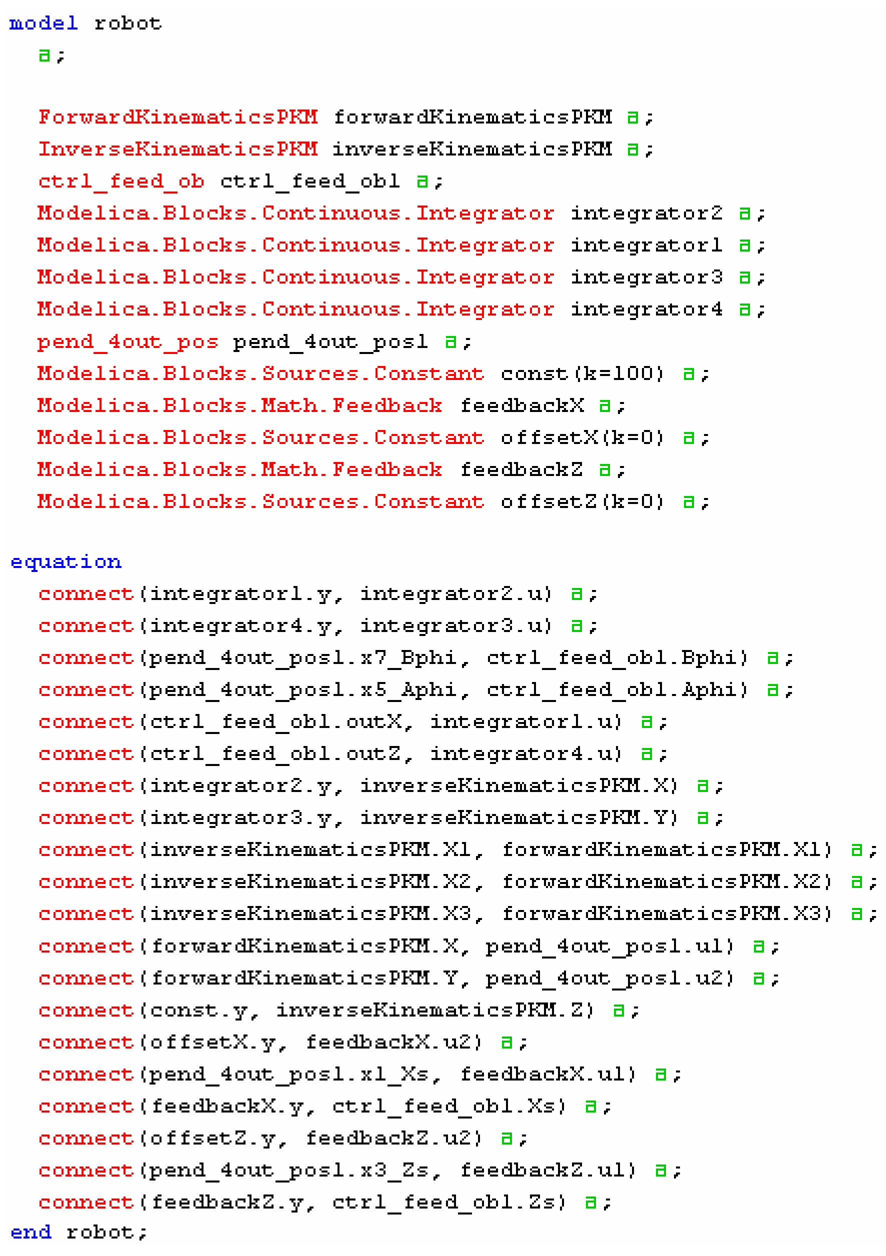
E MATLAB code for calculating forward kinematics for PKM

```
% DIRECT KINEMATICS

% Reference system: WORLD

% INPUT ARGUMENTS
% P1 -> 3x1 -> P1 = [P1x; P1y; P1z] -> Position of cart 1
% P2 -> 3x1 -> P2 = [P2x; P2y; P2z] -> Position of cart 2
% P3 -> 3x1 -> P3 = [P3x; P3y; P3z] -> Position of cart 3
% L -> 1x3 -> L = [L1 L2 L3] -> Lengths of the links 1,2 and 3
% mp_D -> 3x3 -> % Offset on the plate

% OUTPUT ARGUMENTS
% TCP1 -> 3x1 -> TCP1 = [TCP1x; TCP1y; TPC1z] -> Tool centre point 1
% TCP2 -> 3x1 -> TCP2 = [TCP2x; TCP2y; TPC2z] -> Tool centre point 2

function [TCP1,TCP2] = robot(P1, P2, P3, L, mp_D)

% Length every link
L1 = L(1); % 1x1
L2 = L(2);
L3 = L(3);

% Offset on the plate
mp_d1 = mp_D(:,1); % 3x1
mp_d2 = mp_D(:,2);
mp_d3 = mp_D(:,3);

% Centre of spheres
P1c = P1 - mp_d1; % 3x1
P2c = P2 - mp_d2;
P3c = P3 - mp_d3;

% Distance between spheres
P12 = P1c - P2c; % 3x1
P13 = P1c - P3c;

% Intersection of two spheres, assume solution is a circle (not a
point)
s1 = (L2^2+norm(P12)^2-L1^2)/(2*norm(P12)); % 1x1
s2 = (L3^2+norm(P13)^2-L1^2)/(2*norm(P13));

% Radio of the intersection circle
r = sqrt(L2^2-s1^2); % 1x1

% Center of the circle
D = P2c + s1*P12/norm(P12); % 3x1

% A point on the plane
E = P3c + s2*P13/norm(P13); % 3x1

% Normal vector of the plane
N = P13; % 3x1

% Rotation matrixes
theta = atan2((P2c(1)-P1c(1)),(P1c(2)-P2c(2))); % 1x1
beta = acos((P1c(3)-P2c(3))/(norm(P12)));
```

```
Rotz = [cos(theta) sin(theta) 0; -sin(theta) cos(theta) 0; 0 0 1]; %
3x3
Rotx = [1 0 0; 0 cos(beta) -sin(beta); 0 sin(beta) cos(beta)];

Rotxz = Rotx*Rotz; % 3x3

% The normal vector for the plane and points D and E are transformed
into a
% coordinate system with the z-axis pointing from P2 to P1.
N1 = Rotxz*N; % 3x1
Nx = N1(1);
Ny = N1(2);
Nz = N1(3);

D1 = Rotxz*D; % 3x1
xd = D1(1);
yd = D1(2);
zd = D1(3);

E1 = Rotxz*E; % 3x1
xe = E1(1);
ye = E1(2);
ze = E1(3);

% Intermediate parameters to calculate TCP
T1 = xd - xe;
T2 = yd - ye;
T3 = zd - ze;
Nxy = Nx*Ny;
Nyz = Ny*Nz;
Nxz = Nx*Nz;
Nq = Nx^2 + Ny^2;
Q = Ny*r;
R = Ny*T2 + Nz*T3;

if Nx^2*(r+T1)*(r-T1) - 2*Nx*T1*R + Q^2 -R^2 < 0
    error('No solution');
    return;
else
    S =  sqrt(Nx^2*(r+T1)*(r-T1) - 2*Nx*T1*R + Q^2 -R^2);
end

% One possibility
xr1 = (Ny^2*xd+Nx^2*xe-Nxy*T2-Nxz*T3-Ny*S)/Nq;
yr1 = (Ny^2*ye+Nx^2*yd-Nxy*T1-Nyz*T3+Nx*S)/Nq;

% Other possibility
xr2 = (Ny^2*xd+Nx^2*xe-Nxy*T2-Nxz*T3+Ny*S)/Nq;
yr2 = (Ny^2*ye+Nx^2*yd-Nxy*T1-Nyz*T3-Nx*S)/Nq;

zr = zd;

% Solutions
TCP1 = inv(Rotxz)*[xr1 ; yr1 ; zr]; % 3x1
TCP2 = inv(Rotxz)*[xr2 ; yr2 ; zr];
```

F Modelica code for the final simulation

The program below belongs to final simulation:



```
model robot
  a;

  ForwardKinematicsPKM forwardKinematicsPKM a;
  InverseKinematicsPKM inverseKinematicsPKM a;
  ctrl_feed_ob ctrl_feed_ob1 a;
  Modelica.Blocks.Continuous.Integrator integrator2 a;
  Modelica.Blocks.Continuous.Integrator integrator1 a;
  Modelica.Blocks.Continuous.Integrator integrator3 a;
  Modelica.Blocks.Continuous.Integrator integrator4 a;
  pend_4out_pos pend_4out_pos1 a;
  Modelica.Blocks.Sources.Constant const(k=100) a;
  Modelica.Blocks.Math.Feedback feedbackX a;
  Modelica.Blocks.Sources.Constant offsetX(k=0) a;
  Modelica.Blocks.Math.Feedback feedbackZ a;
  Modelica.Blocks.Sources.Constant offsetZ(k=0) a;

equation
  connect(integrator1.y, integrator2.u) a;
  connect(integrator4.y, integrator3.u) a;
  connect(pend_4out_pos1.x7_Bphi, ctrl_feed_ob1.Bphi) a;
  connect(pend_4out_pos1.x5_Aphi, ctrl_feed_ob1.Aphi) a;
  connect(ctrl_feed_ob1.outX, integrator1.u) a;
  connect(ctrl_feed_ob1.outZ, integrator4.u) a;
  connect(integrator2.y, inverseKinematicsPKM.X) a;
  connect(integrator3.y, inverseKinematicsPKM.Y) a;
  connect(inverseKinematicsPKM.X1, forwardKinematicsPKM.X1) a;
  connect(inverseKinematicsPKM.X2, forwardKinematicsPKM.X2) a;
  connect(inverseKinematicsPKM.X3, forwardKinematicsPKM.X3) a;
  connect(forwardKinematicsPKM.X, pend_4out_pos1.u1) a;
  connect(forwardKinematicsPKM.Y, pend_4out_pos1.u2) a;
  connect(const.y, inverseKinematicsPKM.Z) a;
  connect(offsetX.y, feedbackX.u2) a;
  connect(pend_4out_pos1.x1_Xs, feedbackX.u1) a;
  connect(feedbackX.y, ctrl_feed_ob1.Xs) a;
  connect(offsetZ.y, feedbackZ.u2) a;
  connect(pend_4out_pos1.x3_Zs, feedbackZ.u1) a;
  connect(feedbackZ.y, ctrl_feed_ob1.Zs) a;
end robot;
```

Now, I am going to show the code of the four bocks (*forwardKinematicsPKM*, *inverseKinematicsPKM*, *ctrl_feed_ob1* and *pend_4out_pos1*), which are contained inside this program.

F.1 ForwardKinematicsPKM

```
block ForwardKinematicsPKM
  "Join-coordinates from TCP-coordinates of PKM (step1)"
  a;

  Modelica.Blocks.Interfaces.RealInput X1 "Join 1" a;
  Modelica.Blocks.Interfaces.RealInput X2 "Join 2" a;
  Modelica.Blocks.Interfaces.RealInput X3 "Join 3" a;

  Modelica.Blocks.Interfaces.RealOutput P1x "Cart1, X-coordinate";
    //annotation (extent=[-140,90; -120,110]);
  Modelica.Blocks.Interfaces.RealOutput P1y "Cart1, Y-coordinate";
    //annotation (extent=[-140,70; -120,90]);
  Modelica.Blocks.Interfaces.RealOutput P1z "Cart1, Z-coordinate";
    //annotation (extent=[-140,50; -120,70]);
  Modelica.Blocks.Interfaces.RealOutput P2x "Cart2, X-coordinate";
    //annotation (extent=[-140,10; -120,30]);
  Modelica.Blocks.Interfaces.RealOutput P2y "Cart2, Y-coordinate";
    //annotation (extent=[-140,-10; -120,10]);
  Modelica.Blocks.Interfaces.RealOutput P2z "Cart2, Z-coordinate";
    //annotation (extent=[-140,-30; -120,-10]);
  Modelica.Blocks.Interfaces.RealOutput P3x "Cart3, X-coordinate";
    //annotation (extent=[-140,-70; -120,-50]);
  Modelica.Blocks.Interfaces.RealOutput P3y "Cart3, Y-coordinate";
    //annotation (extent=[-140,-90; -120,-70]);
  Modelica.Blocks.Interfaces.RealOutput P3z "Cart3, Z-coordinate";
    //annotation (extent=[-140,-110; -120,-90]);

  Modelica.Blocks.Interfaces.RealOutput P1cx "Centre of sphere1, X-coordinate";
    //annotation (extent=[-120,90; -100,110]);
  Modelica.Blocks.Interfaces.RealOutput P1cy "Centre of sphere1, Y-coordinate";
    //annotation (extent=[-120,70; -100,90]);
  Modelica.Blocks.Interfaces.RealOutput P1cz "Centre of sphere1, Z-coordinate";
    //annotation (extent=[-120,50; -100,70]);
  Modelica.Blocks.Interfaces.RealOutput P2cx "Centre of sphere2, X-coordinate";
    //annotation (extent=[-120,10; -100,30]);
  Modelica.Blocks.Interfaces.RealOutput P2cy "Centre of sphere2, Y-coordinate";
    //annotation (extent=[-120,-10; -100,10]);
  Modelica.Blocks.Interfaces.RealOutput P2cz "Centre of sphere2, Z-coordinate";
    //annotation (extent=[-120,-30; -100,-10]);
  Modelica.Blocks.Interfaces.RealOutput P3cx "Centre of sphere3, X-coordinate";
    //annotation (extent=[-120,-70; -100,-50]);
  Modelica.Blocks.Interfaces.RealOutput P3cy "Centre of sphere3, Y-coordinate";
    //annotation (extent=[-120,-90; -100,-70]);
  Modelica.Blocks.Interfaces.RealOutput P3cz "Centre of sphere3, Z-coordinate";
    //annotation (extent=[-120,-110; -100,-90]);

  Modelica.Blocks.Interfaces.RealOutput P12x
    "Distance between spheres 1 and 2, X-coordinate";
    //annotation (extent=[-100,50; -80,70]);
  Modelica.Blocks.Interfaces.RealOutput P12y
    "Distance between spheres 1 and 2, Y-coordinate";
    //annotation (extent=[-100,30; -80,50]);
  Modelica.Blocks.Interfaces.RealOutput P12z
    "Distance between spheres 1 and 2, Z-coordinate";
    //annotation (extent=[-100,10; -80,30]);
```

```
Modelica.Blocks.Interfaces.RealOutput P13x
  "Distance between spheres 1 and 3, X-coordinate";
  //annotation (extent=[-100,-30; -80,-10]);
Modelica.Blocks.Interfaces.RealOutput P13y
  "Distance between spheres 1 and 3, Y-coordinate";
  //annotation (extent=[-100,-50; -80,-30]);
Modelica.Blocks.Interfaces.RealOutput P13z
  "Distance between spheres 1 and 3, Z-coordinate";
  //annotation (extent=[-100,-70; -80,-50]);

Modelica.Blocks.Interfaces.RealOutput s1
  "Intersection between spheres 1 and 2";
                                        //annotation (extent=[-80,30; -60,50]);
Modelica.Blocks.Interfaces.RealOutput s2
  "Intersection between spheres 1 and 3";
                                        //annotation (extent=[-80,-50; -60,-30]);

Modelica.Blocks.Interfaces.RealOutput r "Radio of the intersection circle";
                                       //annotation (extent=[-80,-90; -60,-70]);

Modelica.Blocks.Interfaces.RealOutput Dx "Centre of the circle, X-coordinate";
  //annotation (extent=[-60,90; -40,110]);
Modelica.Blocks.Interfaces.RealOutput Dy "Centre of the circle, Y-coordinate";
  //annotation (extent=[-60,70; -40,90]);
Modelica.Blocks.Interfaces.RealOutput Dz "Centre of the circle, Z-coordinate";
  //annotation (extent=[-60,50; -40,70]);

Modelica.Blocks.Interfaces.RealOutput Ex "A point on the plane, X-coordinate";
  //annotation (extent=[-60,10; -40,30]);
Modelica.Blocks.Interfaces.RealOutput Ey "A point on the plane, Y-coordinate";
  //annotation (extent=[-60,-10; -40,10]);
Modelica.Blocks.Interfaces.RealOutput Ez "A point on the plane, Z-coordinate";
  //annotation (extent=[-60,-30; -40,-10]);

Modelica.Blocks.Interfaces.RealOutput Nx
  "Normal vector of the plane, X-coordinate";
  //annotation (extent=[-60,-70; -40,-50]);
Modelica.Blocks.Interfaces.RealOutput Ny
  "Normal vector of the plane, Y-coordinate";
  //annotation (extent=[-60,-90; -40,-70]);
Modelica.Blocks.Interfaces.RealOutput Nz
  "Normal vector of the plane, Z-coordinate";
  //annotation (extent=[-60,-110; -40,-90]);

Modelica.Blocks.Interfaces.RealOutput theta
  "Parameter used for rotation matrixes"; //annotation (extent=[-40,-50; -20,-30]);
Modelica.Blocks.Interfaces.RealOutput beta(redeclare type SignalType = Real)
  "Parameter used for rotation matrixes"; //annotation (extent=[-40,30; -20,50]);
```

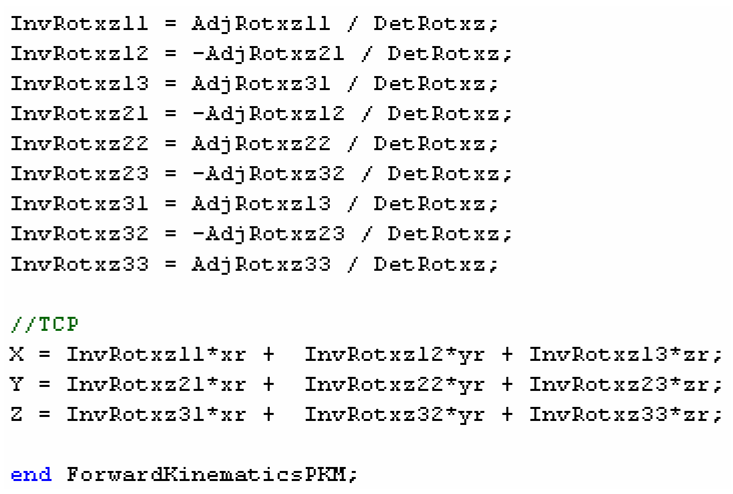
```
Modelica.Blocks.Interfaces.RealOutput Rotz11 "Rotation matrix z";
  //annotation (extent=[-20,90; 0,110]);
Modelica.Blocks.Interfaces.RealOutput Rotz12 "Rotation matrix z";
  //annotation (extent=[-20,70; 0,90]);
Modelica.Blocks.Interfaces.RealOutput Rotz13 "Rotation matrix z";
  //annotation (extent=[-20,50; 0,70]);
Modelica.Blocks.Interfaces.RealOutput Rotz21 "Rotation matrix z";
  //annotation (extent=[-20,10; 0,30]);
Modelica.Blocks.Interfaces.RealOutput Rotz22 "Rotation matrix z";
  //annotation (extent=[-20,-10; 0,10]);
Modelica.Blocks.Interfaces.RealOutput Rotz23 "Rotation matrix z";
  //annotation (extent=[-20,-30; 0,-10]);
Modelica.Blocks.Interfaces.RealOutput Rotz31 "Rotation matrix z";
  //annotation (extent=[-20,-70; 0,-50]);
Modelica.Blocks.Interfaces.RealOutput Rotz32 "Rotation matrix z";
  //annotation (extent=[-20,-90; 0,-70]);
Modelica.Blocks.Interfaces.RealOutput Rotz33 "Rotation matrix z";
  //annotation (extent=[-20,-110; 0,-90]);

Modelica.Blocks.Interfaces.RealOutput Rotx11 "Rotation matrix x";
  //annotation (extent=[0,90; 20,110]);
Modelica.Blocks.Interfaces.RealOutput Rotx12 "Rotation matrix x";
  //annotation (extent=[0,70; 20,90]);
Modelica.Blocks.Interfaces.RealOutput Rotx13 "Rotation matrix x";
  //annotation (extent=[0,50; 20,70]);
Modelica.Blocks.Interfaces.RealOutput Rotx21 "Rotation matrix x";
  //annotation (extent=[0,10; 20,30]);
Modelica.Blocks.Interfaces.RealOutput Rotx22 "Rotation matrix x";
  //annotation (extent=[0,-10; 20,10]);
Modelica.Blocks.Interfaces.RealOutput Rotx23 "Rotation matrix x";
  //annotation (extent=[0,-30; 20,-10]);
Modelica.Blocks.Interfaces.RealOutput Rotx31 "Rotation matrix x";
  //annotation (extent=[0,-70; 20,-50]);
Modelica.Blocks.Interfaces.RealOutput Rotx32 "Rotation matrix x";
  //annotation (extent=[0,-90; 20,-70]);
Modelica.Blocks.Interfaces.RealOutput Rotx33 "Rotation matrix x";
  //annotation (extent=[0,-110; 20,-90]);

Modelica.Blocks.Interfaces.RealOutput Rotxz11 "Rotation matrix z and x";
  //annotation (extent=[20,90; 40,110]);
Modelica.Blocks.Interfaces.RealOutput Rotxz12 "Rotation matrix z and x";
  //annotation (extent=[20,70; 40,90]);
Modelica.Blocks.Interfaces.RealOutput Rotxz13 "Rotation matrix z and x";
  //annotation (extent=[20,50; 40,70]);
Modelica.Blocks.Interfaces.RealOutput Rotxz21 "Rotation matrix z and x";
  //annotation (extent=[20,10; 40,30]);
Modelica.Blocks.Interfaces.RealOutput Rotxz22 "Rotation matrix z and x";
  //annotation (extent=[20,-10; 40,10]);
Modelica.Blocks.Interfaces.RealOutput Rotxz23 "Rotation matrix z and x";
  //annotation (extent=[20,-30; 40,-10]);
Modelica.Blocks.Interfaces.RealOutput Rotxz31 "Rotation matrix z and x";
  //annotation (extent=[20,-70; 40,-50]);
Modelica.Blocks.Interfaces.RealOutput Rotxz32 "Rotation matrix z and x";
  //annotation (extent=[20,-90; 40,-70]);
Modelica.Blocks.Interfaces.RealOutput Rotxz33 "Rotation matrix z and x";
  //annotation (extent=[20,-110; 40,-90]);
```

```
Modelica.Blocks.Interfaces.RealOutput N1x
  "Transformed normal vector, X-coordinate";
  //annotation (extent=[40,90; 60,110]);
Modelica.Blocks.Interfaces.RealOutput N1y
  "Transformed normal vector, Y-coordinate";
  //annotation (extent=[40,70; 60,90]);
Modelica.Blocks.Interfaces.RealOutput N1z
  "Transformed normal vector, Z-coordinate";
  //annotation (extent=[40,50; 60,70]);

Modelica.Blocks.Interfaces.RealOutput D1x "Transformed point D, X-coordinate";
  //annotation (extent=[40,10; 60,30]);
Modelica.Blocks.Interfaces.RealOutput D1y "Transformed point D, Y-coordinate";
  //annotation (extent=[40,-10; 60,10]);
Modelica.Blocks.Interfaces.RealOutput D1z "Transformed point D, Z-coordinate";
  //annotation (extent=[40,-30; 60,-10]);

Modelica.Blocks.Interfaces.RealOutput E1x "Transformed point E, X-coordinate";
  //annotation (extent=[40,-70; 60,-50]);
Modelica.Blocks.Interfaces.RealOutput E1y "Transformed point E, Y-coordinate";
  //annotation (extent=[40,-90; 60,-70]);
Modelica.Blocks.Interfaces.RealOutput E1z "Transformed point E, Z-coordinate";
  //annotation (extent=[40,-110; 60,-90]);

Modelica.Blocks.Interfaces.RealOutput T1 "Intermediate parameter";
  //annotation (extent=[60,80; 80,100]);
Modelica.Blocks.Interfaces.RealOutput T2 "Intermediate parameter";
  //annotation (extent=[60,60; 80,80]);
Modelica.Blocks.Interfaces.RealOutput T3 "Intermediate parameter";
  //annotation (extent=[60,40; 80,60]);
Modelica.Blocks.Interfaces.RealOutput Nxy "Intermediate parameter";
  //annotation (extent=[60,20; 80,40]);
Modelica.Blocks.Interfaces.RealOutput Nyz "Intermediate parameter";
  //annotation (extent=[60,0; 80,20]);
Modelica.Blocks.Interfaces.RealOutput Nxz "Intermediate parameter";
  //annotation (extent=[60,-20; 80,0]);
Modelica.Blocks.Interfaces.RealOutput Nq "Intermediate parameter";
  //annotation (extent=[60,-40; 80,-20]);
Modelica.Blocks.Interfaces.RealOutput Q "Intermediate parameter";
  //annotation (extent=[60,-60; 80,-40]);
Modelica.Blocks.Interfaces.RealOutput R "Intermediate parameter";
  //annotation (extent=[60,-80; 80,-60]);
Modelica.Blocks.Interfaces.RealOutput S "Intermediate parameter";
  //annotation (extent=[60,-100; 80,-80]);

Modelica.Blocks.Interfaces.RealOutput xr "Intermediate parameter";
  //annotation (extent=[80,10; 100,30]);
Modelica.Blocks.Interfaces.RealOutput yr "Intermediate parameter";
  //annotation (extent=[80,-10; 100,10]);
Modelica.Blocks.Interfaces.RealOutput zr "Intermediate parameter";
  //annotation (extent=[80,-30; 100,-10]);

Modelica.Blocks.Interfaces.RealOutput DetRotxz "Necessary to calculate Rotxz";
  //annotation (extent=[100,90; 120,110]);
```

```
Modelica.Blocks.Interfaces.RealOutput AdjRotxz11
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,50; 120,70]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz12
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,30; 120,50]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz13
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,10; 120,30]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz21
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,-10; 120,10]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz22
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,-30; 120,-10]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz23
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,-50; 120,-30]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz31
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,-70; 120,-50]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz32
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,-90; 120,-70]);
Modelica.Blocks.Interfaces.RealOutput AdjRotxz33
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[100,-110; 120,-90]);

Modelica.Blocks.Interfaces.RealOutput InvRotxz11
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,70; 140,90]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz12
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,50; 140,70]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz13
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,30; 140,50]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz21
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,10; 140,30]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz22
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,-10; 140,10]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz23
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,-30; 140,-10]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz31
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,-50; 140,-30]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz32
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,-70; 140,-50]);
Modelica.Blocks.Interfaces.RealOutput InvRotxz33
  "Necessary to calculate Rotxz inverse matrix";
  //annotation (extent=[120,-90; 140,-70]);
  Modelica.Blocks.Interfaces.RealOutput X "X-coordinate TCP" a;
  Modelica.Blocks.Interfaces.RealOutput Y "Y-coordinate TCP" a;
  Modelica.Blocks.Interfaces.RealOutput Z "Z-coordinate TCP" a;

//Length of links
parameter Real L1 = 400;
parameter Real L2 = 300;
parameter Real L3 = 300;

//Offset global origin to q=0 on track - offset TCP frame origin to joint
parameter Real X1offset = 0;
parameter Real Y1offset = 260;
parameter Real Z1offset = 370;

parameter Real X2offset = 0;
parameter Real Y2offset = 0;
parameter Real Z2offset = 370;

parameter Real X3offset = 0;
parameter Real Y3offset = 260;
parameter Real Z3offset = 100;

//Offset on the plate
parameter Real mp_d1[3,1] = [-X1offset;0;0];
parameter Real mp_d2[3,1] = [-X2offset;0;0];
parameter Real mp_d3[3,1] = [-X3offset;0;0];

equation
//Position of the carts
P1x = X1;
P1y = Y1offset;
P1z = Z1offset;

P2x = X2;
P2y = Y2offset;
P2z = Z2offset;

P3x = X3;
P3y = Y3offset;
P3z = Z3offset;

//Centre of spheres
P1cx = P1x - mp_d1[1,1];
P1cy = P1y - mp_d1[2,1];
P1cz = P1z - mp_d1[3,1];

P2cx = P2x - mp_d2[1,1];
P2cy = P2y - mp_d2[2,1];
P2cz = P2z - mp_d2[3,1];

P3cx = P3x - mp_d3[1,1];
P3cy = P3y - mp_d3[2,1];
P3cz = P3z - mp_d3[3,1];

//Distance between spheres
P12x = P1cx - P2cx;
P12y = P1cy - P2cy;
P12z = P1cz - P2cz;
```

```
P13x = P1cx - P3cx;
P13y = P1cy - P3cy;
P13z = P1cz - P3cz;

//Intersection of two spheres, assume solution is a circle (not a point)
s1 = (L2^2 + P12x^2 + P12y^2 + P12z^2 - L1^2) / (2*sqrt(P12x^2 + P12y^2 + P12z^2));
s2 = (L3^2 + P13x^2 + P13y^2 + P13z^2 - L1^2) / (2*sqrt(P13x^2 + P13y^2 + P13z^2));

//Radio of the intersection circle
r = sqrt(L2^2 - s1^2);

//Centre of the circle
Dx = P2cx + s1*P12x / sqrt(P12x^2 + P12y^2 + P12z^2);
Dy = P2cy + s1*P12y / sqrt(P12x^2 + P12y^2 + P12z^2);
Dz = P2cz + s1*P12z / sqrt(P12x^2 + P12y^2 + P12z^2);

//A point on the plane
Ex = P3cx + s2*P13x / sqrt(P13x^2 + P13y^2 + P13z^2);
Ey = P3cy + s2*P13y / sqrt(P13x^2 + P13y^2 + P13z^2);
Ez = P3cz + s2*P13z / sqrt(P13x^2 + P13y^2 + P13z^2);

//Normal vector of the plane
Nx = P13x;
Ny = P13y;
Nz = P13z;

//Rotation matrixes
theta = Modelica.Math.atan2((P2cx - P1cx), (P1cy - P2cy));
beta = Modelica.Math.acos((P1cz - P2cz) / sqrt(P12x^2 + P12y^2 + P12z^2));

Rotz11 = cos(theta);
Rotz12 = sin(theta);
Rotz13 = 0;
Rotz21 = -sin(theta);
Rotz22 = cos(theta);
Rotz23 = 0;
Rotz31 = 0;
Rotz32 = 0;
Rotz33 = 1;

Rotx11 = 1;
Rotx12 = 0;
Rotx13 = 0;
Rotx21 = 0;
Rotx22 = cos(beta);
Rotx23 = -sin(beta);
Rotx31 = 0;
Rotx32 = sin(beta);
Rotx33 = cos(beta);
```

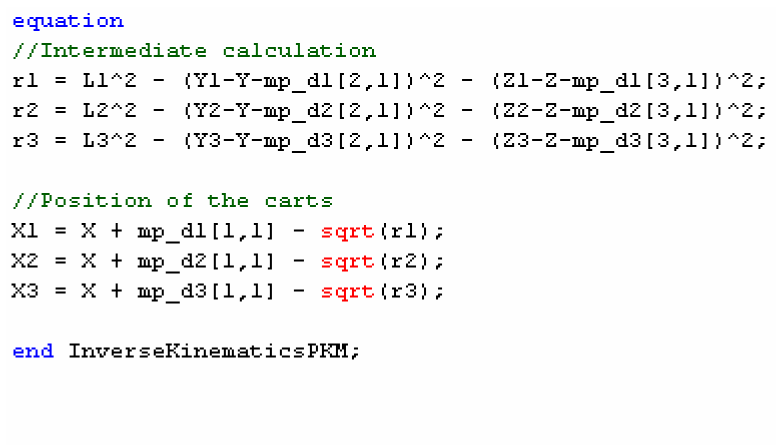
```
//parameter Real Rotxz[3,3] = Rotx*Rotz;
Rotxz11 = Rotx11*Rotz11 + Rotx12*Rotz21 + Rotx13*Rotz31;
Rotxz12 = Rotx11*Rotz12 + Rotx12*Rotz22 + Rotx13*Rotz32;
Rotxz13 = Rotx11*Rotz13 + Rotx12*Rotz23 + Rotx13*Rotz33;
Rotxz21 = Rotx21*Rotz11 + Rotx22*Rotz21 + Rotx23*Rotz31;
Rotxz22 = Rotx21*Rotz12 + Rotx22*Rotz22 + Rotx23*Rotz32;
Rotxz23 = Rotx21*Rotz13 + Rotx22*Rotz23 + Rotx23*Rotz33;
Rotxz31 = Rotx31*Rotz11 + Rotx32*Rotz21 + Rotx33*Rotz31;
Rotxz32 = Rotx31*Rotz12 + Rotx32*Rotz22 + Rotx33*Rotz32;
Rotxz33 = Rotx31*Rotz13 + Rotx32*Rotz23 + Rotx33*Rotz33;

//The normal vector for the plane and points D and E are transformed into a
//coordinate system with the z-axis pointing from P2 to P1.
N1x = Rotxz11*Nx + Rotxz12*Ny + Rotxz13*Nz;
N1y = Rotxz21*Nx + Rotxz22*Ny + Rotxz23*Nz;
N1z = Rotxz31*Nx + Rotxz32*Ny + Rotxz33*Nz;

D1x = Rotxz11*Dx + Rotxz12*Dy + Rotxz13*Dz;
D1y = Rotxz21*Dx + Rotxz22*Dy + Rotxz23*Dz;
D1z = Rotxz31*Dx + Rotxz32*Dy + Rotxz33*Dz;

E1x = Rotxz11*Ex + Rotxz12*Ey + Rotxz13*Ez;
E1y = Rotxz21*Ex + Rotxz22*Ey + Rotxz23*Ez;
E1z = Rotxz31*Ex + Rotxz32*Ey + Rotxz33*Ez;

//Intermediate parameters to calculate TCP
T1 = D1x - E1x;
T2 = D1y - E1y;
T3 = D1z - E1z;
Nxy = N1x*N1y;
Nyz = N1y*N1z;
Nxz = N1x*N1z;
Nq = N1x^2 + N1y^2;
Q = N1y*r;
R = N1y*T2 + N1z*T3;
S = sqrt(N1x^2*(r+T1)*(r-T1) - 2*N1x*T1*R + Q^2 -R^2);

//Intermediate parameters to calculate TCP
xr = (N1y^2*D1x + N1x^2*E1x - Nxy*T2 - Nxz*T3 - N1y*S) / Nq;
yr = (N1y^2*E1y + N1x^2*D1y - Nxy*T1 - Nyz*T3 + N1x*S) / Nq;
zr = D1z;

//Intermediate parameters to calculate Rotxz inverse matrix
DetRotxz = Rotxz11*Rotxz22*Rotxz33 + Rotxz12*Rotxz23*Rotxz31 + Rotxz13*Rotxz21*Rotxz32
 - (Rotxz13*Rotxz22*Rotxz31 + Rotxz12*Rotxz21*Rotxz33 + Rotxz11*Rotxz23*Rotxz32);

AdjRotxz11 = Rotxz22*Rotxz33 - Rotxz23*Rotxz32;
AdjRotxz12 = Rotxz21*Rotxz33 - Rotxz23*Rotxz31;
AdjRotxz13 = Rotxz21*Rotxz32 - Rotxz22*Rotxz31;
AdjRotxz21 = Rotxz12*Rotxz33 - Rotxz13*Rotxz32;
AdjRotxz22 = Rotxz11*Rotxz33 - Rotxz13*Rotxz31;
AdjRotxz23 = Rotxz11*Rotxz32 - Rotxz12*Rotxz31;
AdjRotxz31 = Rotxz12*Rotxz23 - Rotxz13*Rotxz22;
AdjRotxz32 = Rotxz11*Rotxz23 - Rotxz13*Rotxz21;
AdjRotxz33 = Rotxz11*Rotxz22 - Rotxz12*Rotxz21;
```



```
InvRotxz11 = AdjRotxz11 / DetRotxz;
InvRotxz12 = -AdjRotxz21 / DetRotxz;
InvRotxz13 = AdjRotxz31 / DetRotxz;
InvRotxz21 = -AdjRotxz12 / DetRotxz;
InvRotxz22 = AdjRotxz22 / DetRotxz;
InvRotxz23 = -AdjRotxz32 / DetRotxz;
InvRotxz31 = AdjRotxz13 / DetRotxz;
InvRotxz32 = -AdjRotxz23 / DetRotxz;
InvRotxz33 = AdjRotxz33 / DetRotxz;

//TCP
X = InvRotxz11*xr +  InvRotxz12*yr + InvRotxz13*zr;
Y = InvRotxz21*xr +  InvRotxz22*yr + InvRotxz23*zr;
Z = InvRotxz31*xr +  InvRotxz32*yr + InvRotxz33*zr;

end ForwardKinematicsPKM;
```

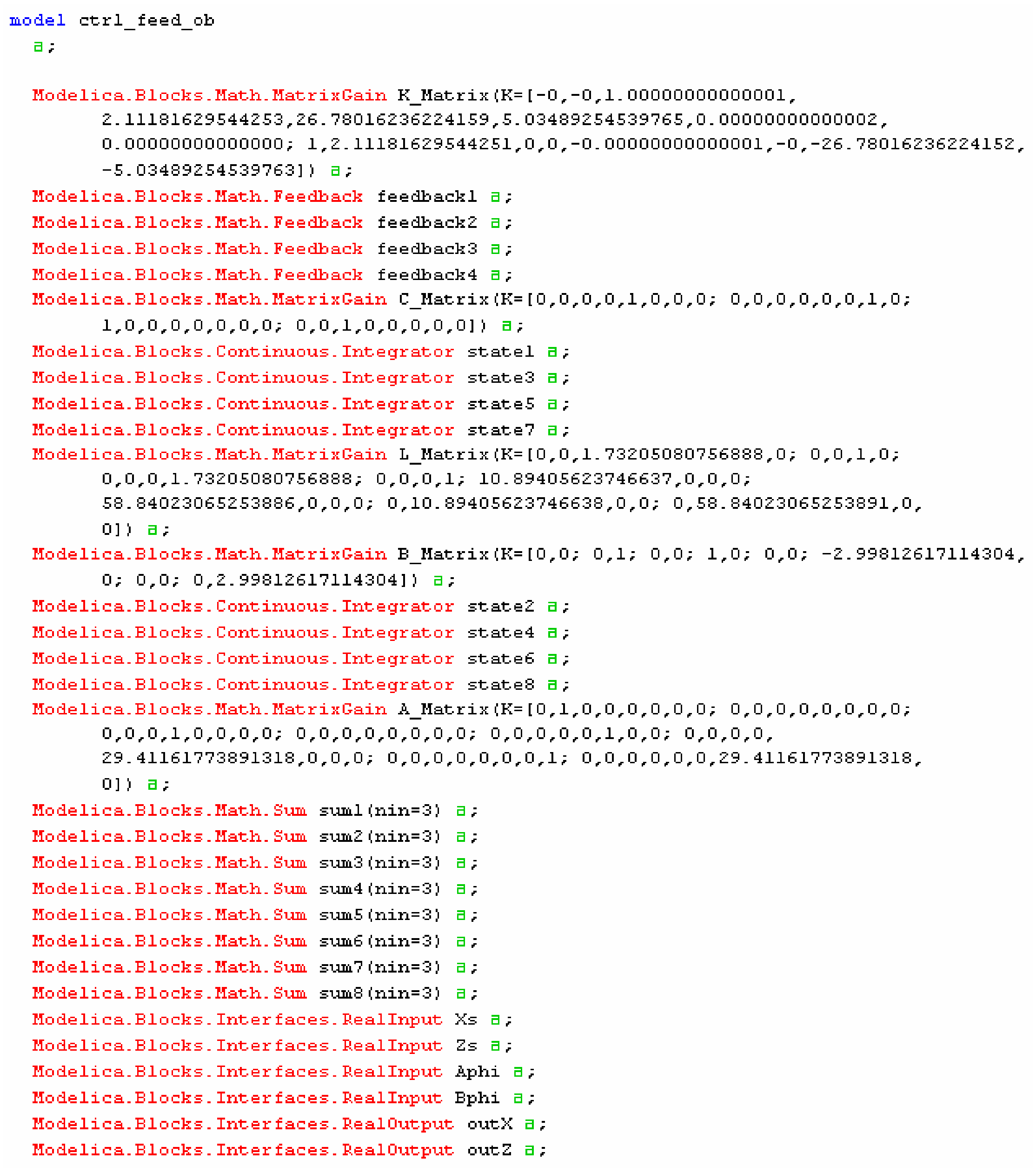
F.2 ForwardKinematicsPKM

```
block InverseKinematicsPKM
  "Join-coordinates from TCP-coordinates of PKM (step1)"
  a;

  Modelica.Blocks.Interfaces.RealInput X "X-coordinate TCP" a;
  Modelica.Blocks.Interfaces.RealInput Y "Y-coordinate TCP" a;
  Modelica.Blocks.Interfaces.RealInput Z "Z-coordinate TCP" a;

  Modelica.Blocks.Interfaces.RealOutput r1 "Cart1";
    //annotation (extent=[0,70; 20,90]);
  Modelica.Blocks.Interfaces.RealOutput r2 "Cart2";
    //annotation (extent=[0,-10; 20,10]);
  Modelica.Blocks.Interfaces.RealOutput r3 "Cart3";
    //annotation (extent=[0,-90; 20,-70]);

  Modelica.Blocks.Interfaces.RealOutput X1 "Join 1" a;
  Modelica.Blocks.Interfaces.RealOutput X2 "Join 2" a;
  Modelica.Blocks.Interfaces.RealOutput X3 "Join 3" a;

//Length of links
parameter Real L1 = 400;
parameter Real L2 = 300;
parameter Real L3 = 300;

//Offset global origin to q=0 on track - offset TCP frame origin to joint
parameter Real X1offset = 0;
parameter Real Y1offset = 260;
parameter Real Z1offset = 370;

parameter Real X2offset = 0;
parameter Real Y2offset = 0;
parameter Real Z2offset = 370;

parameter Real X3offset = 0;
parameter Real Y3offset = 260;
parameter Real Z3offset = 100;

//Offset on the plate
parameter Real mp_d1[3,1] = [-X1offset;0;0];
parameter Real mp_d2[3,1] = [-X2offset;0;0];
parameter Real mp_d3[3,1] = [-X3offset;0;0];

//Position of the carts
parameter Real P1[3,1] = [165; Y1offset; Z1offset];
parameter Real P2[3,1] = [205; Y2offset; Z2offset];
parameter Real P3[3,1] = [290; Y3offset; Z3offset];

//Coordinates of the carts
parameter Real Y1 = P1[2,1];
parameter Real Z1 = P1[3,1];
parameter Real Y2 = P2[2,1];
parameter Real Z2 = P2[3,1];
parameter Real Y3 = P3[2,1];
parameter Real Z3 = P3[3,1];
```

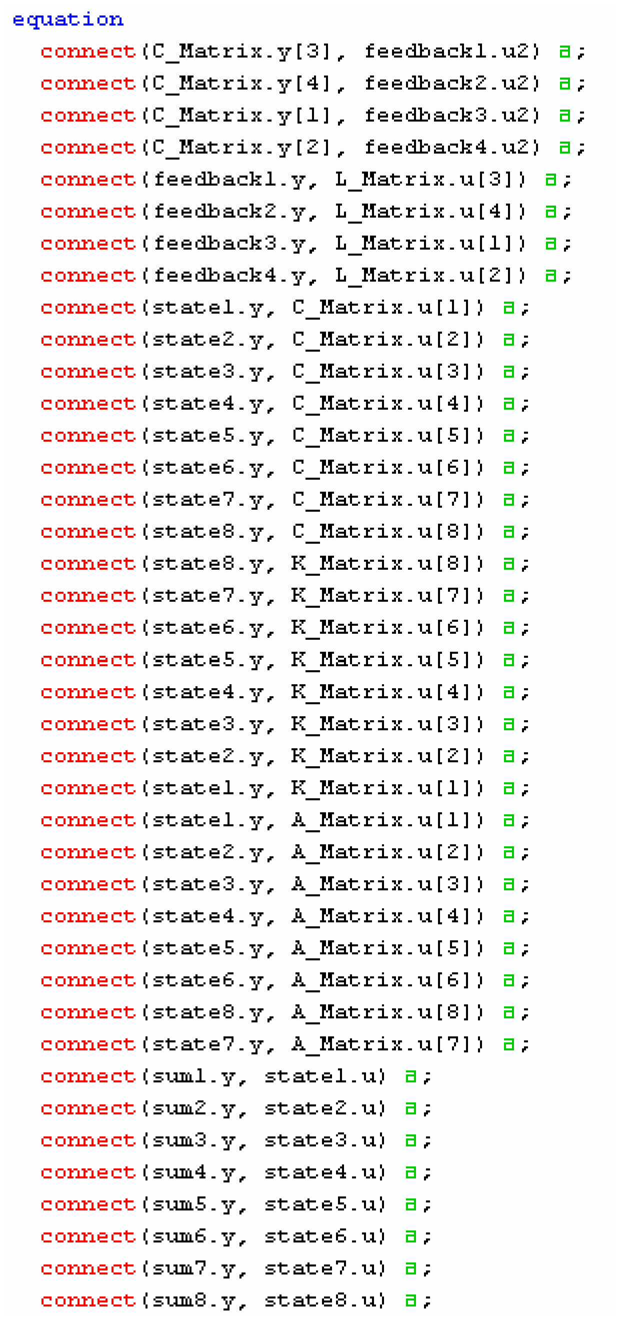


```
equation
//Intermediate calculation
r1 = L1^2 - (Y1-Y-mp_d1[2,1])^2 - (Z1-Z-mp_d1[3,1])^2;
r2 = L2^2 - (Y2-Y-mp_d2[2,1])^2 - (Z2-Z-mp_d2[3,1])^2;
r3 = L3^2 - (Y3-Y-mp_d3[2,1])^2 - (Z3-Z-mp_d3[3,1])^2;

//Position of the carts
X1 = X + mp_d1[1,1] - sqrt(r1);
X2 = X + mp_d2[1,1] - sqrt(r2);
X3 = X + mp_d3[1,1] - sqrt(r3);

end InverseKinematicsPKM;
```

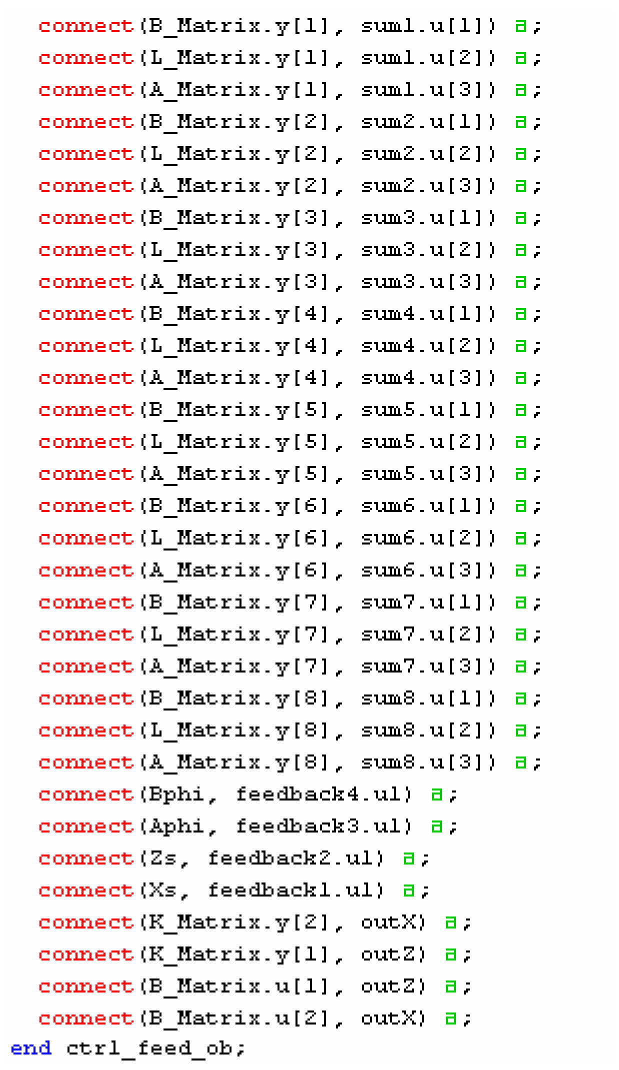
F.3 ctrl_feed_ob



```
model ctrl_feed_ob
  a;

  Modelica.Blocks.Math.MatrixGain K_Matrix(K=[-0,-0,1.00000000000001,
        2.11181629544253,26.78016236224159,5.03489254539765,0.00000000000002,
        0.00000000000000; 1,2.11181629544251,0,0,-0.00000000000001,-0,-26.78016236224152,
        -5.03489254539763]) a;
  Modelica.Blocks.Math.Feedback feedback1 a;
  Modelica.Blocks.Math.Feedback feedback2 a;
  Modelica.Blocks.Math.Feedback feedback3 a;
  Modelica.Blocks.Math.Feedback feedback4 a;
  Modelica.Blocks.Math.MatrixGain C_Matrix(K=[0,0,0,0,1,0,0,0; 0,0,0,0,0,0,1,0;
        1,0,0,0,0,0,0,0; 0,0,1,0,0,0,0,0]) a;
  Modelica.Blocks.Continuous.Integrator state1 a;
  Modelica.Blocks.Continuous.Integrator state3 a;
  Modelica.Blocks.Continuous.Integrator state5 a;
  Modelica.Blocks.Continuous.Integrator state7 a;
  Modelica.Blocks.Math.MatrixGain L_Matrix(K=[0,0,1.73205080756888,0; 0,0,1,0;
        0,0,0,1.73205080756888; 0,0,0,1; 10.89405623746637,0,0,0;
        58.84023065253886,0,0,0; 0,10.89405623746638,0,0; 0,58.84023065253891,0,
        0]) a;
  Modelica.Blocks.Math.MatrixGain B_Matrix(K=[0,0; 0,1; 0,0; 1,0; 0,0; -2.99812617114304,
        0; 0,0; 0,2.99812617114304]) a;
  Modelica.Blocks.Continuous.Integrator state2 a;
  Modelica.Blocks.Continuous.Integrator state4 a;
  Modelica.Blocks.Continuous.Integrator state6 a;
  Modelica.Blocks.Continuous.Integrator state8 a;
  Modelica.Blocks.Math.MatrixGain A_Matrix(K=[0,1,0,0,0,0,0,0; 0,0,0,0,0,0,0,0;
        0,0,0,1,0,0,0,0; 0,0,0,0,0,0,0,0; 0,0,0,0,0,1,0,0; 0,0,0,0,
        29.41161773891318,0,0,0; 0,0,0,0,0,0,0,1; 0,0,0,0,0,0,29.41161773891318,
        0]) a;
  Modelica.Blocks.Math.Sum sum1(nin=3) a;
  Modelica.Blocks.Math.Sum sum2(nin=3) a;
  Modelica.Blocks.Math.Sum sum3(nin=3) a;
  Modelica.Blocks.Math.Sum sum4(nin=3) a;
  Modelica.Blocks.Math.Sum sum5(nin=3) a;
  Modelica.Blocks.Math.Sum sum6(nin=3) a;
  Modelica.Blocks.Math.Sum sum7(nin=3) a;
  Modelica.Blocks.Math.Sum sum8(nin=3) a;
  Modelica.Blocks.Interfaces.RealInput Xs a;
  Modelica.Blocks.Interfaces.RealInput Zs a;
  Modelica.Blocks.Interfaces.RealInput Aphi a;
  Modelica.Blocks.Interfaces.RealInput Bphi a;
  Modelica.Blocks.Interfaces.RealOutput outX a;
  Modelica.Blocks.Interfaces.RealOutput outZ a;
```



```
equation
  connect(C_Matrix.y[3], feedback1.u2) a;
  connect(C_Matrix.y[4], feedback2.u2) a;
  connect(C_Matrix.y[1], feedback3.u2) a;
  connect(C_Matrix.y[2], feedback4.u2) a;
  connect(feedback1.y, L_Matrix.u[3]) a;
  connect(feedback2.y, L_Matrix.u[4]) a;
  connect(feedback3.y, L_Matrix.u[1]) a;
  connect(feedback4.y, L_Matrix.u[2]) a;
  connect(state1.y, C_Matrix.u[1]) a;
  connect(state2.y, C_Matrix.u[2]) a;
  connect(state3.y, C_Matrix.u[3]) a;
  connect(state4.y, C_Matrix.u[4]) a;
  connect(state5.y, C_Matrix.u[5]) a;
  connect(state6.y, C_Matrix.u[6]) a;
  connect(state7.y, C_Matrix.u[7]) a;
  connect(state8.y, C_Matrix.u[8]) a;
  connect(state8.y, K_Matrix.u[8]) a;
  connect(state7.y, K_Matrix.u[7]) a;
  connect(state6.y, K_Matrix.u[6]) a;
  connect(state5.y, K_Matrix.u[5]) a;
  connect(state4.y, K_Matrix.u[4]) a;
  connect(state3.y, K_Matrix.u[3]) a;
  connect(state2.y, K_Matrix.u[2]) a;
  connect(state1.y, K_Matrix.u[1]) a;
  connect(state1.y, A_Matrix.u[1]) a;
  connect(state2.y, A_Matrix.u[2]) a;
  connect(state3.y, A_Matrix.u[3]) a;
  connect(state4.y, A_Matrix.u[4]) a;
  connect(state5.y, A_Matrix.u[5]) a;
  connect(state6.y, A_Matrix.u[6]) a;
  connect(state8.y, A_Matrix.u[8]) a;
  connect(state7.y, A_Matrix.u[7]) a;
  connect(sum1.y, state1.u) a;
  connect(sum2.y, state2.u) a;
  connect(sum3.y, state3.u) a;
  connect(sum4.y, state4.u) a;
  connect(sum5.y, state5.u) a;
  connect(sum6.y, state6.u) a;
  connect(sum7.y, state7.u) a;
  connect(sum8.y, state8.u) a;
```



```
  connect(B_Matrix.y[1], sum1.u[1]) a;
  connect(L_Matrix.y[1], sum1.u[2]) a;
  connect(A_Matrix.y[1], sum1.u[3]) a;
  connect(B_Matrix.y[2], sum2.u[1]) a;
  connect(L_Matrix.y[2], sum2.u[2]) a;
  connect(A_Matrix.y[2], sum2.u[3]) a;
  connect(B_Matrix.y[3], sum3.u[1]) a;
  connect(L_Matrix.y[3], sum3.u[2]) a;
  connect(A_Matrix.y[3], sum3.u[3]) a;
  connect(B_Matrix.y[4], sum4.u[1]) a;
  connect(L_Matrix.y[4], sum4.u[2]) a;
  connect(A_Matrix.y[4], sum4.u[3]) a;
  connect(B_Matrix.y[5], sum5.u[1]) a;
  connect(L_Matrix.y[5], sum5.u[2]) a;
  connect(A_Matrix.y[5], sum5.u[3]) a;
  connect(B_Matrix.y[6], sum6.u[1]) a;
  connect(L_Matrix.y[6], sum6.u[2]) a;
  connect(A_Matrix.y[6], sum6.u[3]) a;
  connect(B_Matrix.y[7], sum7.u[1]) a;
  connect(L_Matrix.y[7], sum7.u[2]) a;
  connect(A_Matrix.y[7], sum7.u[3]) a;
  connect(B_Matrix.y[8], sum8.u[1]) a;
  connect(L_Matrix.y[8], sum8.u[2]) a;
  connect(A_Matrix.y[8], sum8.u[3]) a;
  connect(Bphi, feedback4.u1) a;
  connect(Aphi, feedback3.u1) a;
  connect(Zs, feedback2.u1) a;
  connect(Xs, feedback1.u1) a;
  connect(K_Matrix.y[2], outX) a;
  connect(K_Matrix.y[1], outZ) a;
  connect(B_Matrix.u[1], outZ) a;
  connect(B_Matrix.u[2], outX) a;
end ctrl_feed_ob;
```

F.4 pend_4out_pos

```
model pend_4out_pos
  a;
  inner Modelica.Mechanics.MultiBody.World world a;
  Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic motorX a;
  Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic motorZ(n={0,0,1}) a;
  Modelica.Mechanics.MultiBody.Joints.Universal cone(n_b={0,0,1}) a;
  Modelica.Mechanics.MultiBody.Parts.BodyBox bar(r={0,0.5,0}) a;
  Modelica.Blocks.Interfaces.RealOutput x5_Aphi a;
  Modelica.Mechanics.MultiBody.Sensors.AbsoluteSensor Sensor(
    get_angles=true,
    get_w_abs=true,
    get_r_abs=true,
    get_v_abs=true) a;
  Modelica.Blocks.Interfaces.RealOutput x7_Bphi a;
  Modelica.Blocks.Interfaces.RealInput u2 a;
  Modelica.Blocks.Interfaces.RealInput u1 a;
  Modelica.Blocks.Interfaces.RealOutput x1_Xs a;
  Modelica.Blocks.Interfaces.RealOutput x3_Zs a;
  Modelica.Mechanics.Translational.Position positionX a;
  Modelica.Mechanics.Translational.Position positionZ a;
  Modelica.Mechanics.MultiBody.Joints.ActuatedPrismatic motorY(n={0,1,0}) a;
  Modelica.Mechanics.Translational.Position positionY a;
  Modelica.Blocks.Sources.Sine motion_Z_coordinate(
    amplitude=0,
    freqHz=0,
    startTime=0) a;
equation
  connect(cone.frame_b, bar.frame_a) a;
  connect(x5_Aphi, Sensor.y[7]) a;
  connect(x7_Bphi, Sensor.y[9]) a;
  connect(x1_Xs, Sensor.y[1]) a;
  connect(x3_Zs, Sensor.y[3]) a;
  connect(u1, positionX.s_ref) a;
  connect(Sensor.frame_a, cone.frame_b) a;
  connect(u2, positionZ.s_ref) a;
  connect(world.frame_b, motorY.frame_a) a;
  connect(positionY.flange_b, motorY.axis) a;
  connect(motion_Z_coordinate.y, positionY.s_ref) a;
  connect(motorY.frame_b, motorZ.frame_a) a;
  connect(motorZ.frame_b, motorX.frame_a) a;
  connect(motorX.frame_b, cone.frame_a) a;
  connect(positionZ.flange_b, motorZ.axis) a;
  connect(positionX.flange_b, motorX.axis) a;
end pend_4out_pos;
```

References

[1] Brochier, B., *Control of A Gantry-Tau Structure,* Lund University, 2006.

[2] Brogårdh, T., Williams, I. and Hovland, G., *Kinematic Error Calibration of the Gantry-Tau Parallel Manipulator*, IEEE, 2006.

[3] Data Sheet 3515.

[4] Data Sheet KMZ10A.

[5] Domingo, J., *Robótica. Apuntes para la Asignatura,* Valencia University, 2001.

[6] I. Dressler, A. Robertsson, and R. Johansson, *Accuracy of Kinematic and Dynamic Models of a Gantry-Tau Parallel Kinematic Robot*, in Proc. International Conference on Robotics and Automation, Rome, 2007.

[7] Fritzson, P. *Principles of Object-Oriented Modeling and Simulation with Modelica2.1,* IEEE Press, 2000.

[8] García-Sanz, M. and Motilva Casado, M., *Herramientas para el Estudio de Robots de Cinemática Paralela: Simulador y Prototipo Experimental,* Navarra University (Spain), 2005.

[9] *Getting Started with Dymola,* PDF manual included in Dymola Help.

[10] Gunnar, J., *Dynamical Analysis and System Identification of the Gantry-Tau Parallel Manipulator*, Technical Institute of Linköping, 2005.

[11] Johannesson, L., Berbyuk, V. and Brogårdh, T., *Gantry-Tau – A New Three Degrees of Freedom Parallel Kinematic Robot,* Chalmers University of Technology, Göteborg (Sweden).

[12] Mallo, S. and Mazzone, V., *Construcción y Diseño de Controladores de un Péndulo Invertido Rotante,* National University of Quilmes (Argentina), 2003.

[13] Merlet, J-P., *Parallel Robots,* Kluwer Academic Publishers, 2001.

[14] Ogata, K. *Ingeniería de Control Moderna*, Prentice Hall, 1998.

[15] Romeo Tello, A., *Apuntes de Robótica. Análisis Geométrico y Cinemático,* Zaragoza University, 2003.

[16] Spong, M. W. and Vidyasagar, M., *Robot Dynamics and Control*, John Wiley & Sons, 1989.

[17] Sprenger, B., Kucera, L. and Mourad, S., *Balancing of an Inverted Pendulum with a SCARA Robot*, IEEE, 1998.

[18] http://www.ctr.unican.es/asignaturas/instrumentacion_5_IT/

[19] http://www.dynasim.com

[20] http://www.elfa.se

[21] http://www.electro.patent-invent.com/electricity/inventions/hall_effect.html

[22] http://ib.cnea.gov.ar/~control2/Links/Tutorial_Matlab_esp/

[23] http://www.modelica.org

[24] http://www.prodigyweb.net.mx/saucedo8

[25] http://www.wikipedia.org