UNIVERSITETET I AGDER

Development of a Model-Based Control System for the Gantry-Tau Parallel Kinematic Machine

by

Victor F. Hernández R.

Supervisors:
Geir Hovland (UiA)
Torgny Brogårdh (ABB Corporate Research, Sweden)

University of Agder, 2011
Faculty of Engineering and Science
Department of Engineering

Abstract

The purpose of this thesis was to develop a control system for the Gantry-Tau Parallel Kinematic Machine, based in the motion behaviour of the robot, verification stages of the model and testing with an actual prototype.

The Gantry-Tau is a parallel robot where its structure offers additional characteristics not found in other models like high dexterity, large workspace and high reconfiguration. Still, the robot is undergoing through design at different levels.

The model-based control system was made in Matlab/Simulink and includes the kinematics of the Gantry-Tau. Also, the system was verified using SimulationX for allowing a quick progress through the project. The tests were made using an extended version of the mentioned Matlab/Simulink program loaded into the Gantry-Tau prototype. This extension includes standard programming levels in robotics like path planning and a controller for the system.

The results in this thesis permit the development of other characteristics of the Gantry-Tau, and to be used as an example for the modelling and study of other parallel kinematic machines.

Acknowledgement

I would like to express my sincere gratitude to Prof. Geir Hovland, for giving me the opportunity of a Master study and for being my supervisor in this thesis. I could have not imagined a better mentor for my study and life experience in Norway.

I thank my supervisor Mr. Torgny Brogårdh for his guidance and kind support. All his suggestions helped me to overcome many difficulties in this thesis.

I also thank my fellow classmates for helping me to adapt to this study environment, and for all the good experiences and fun we had in the last two years.

Lastly, and most importantly, I wish to thank all my family in Venezuela for their warm support: my parents Ana and Godulfo for all their love and caring, my brother Alejandro and sister María del Carmen for letting me know I'm their biggest example in life, and to my beautiful princess Gianna for all her constant cheering and affection. To all of them I dedicate this thesis.

Contents

List of Figures

Introduction

High performance robots represent a great challenge in engineering, especially when the design must meet specific work parameters. The field of mechatronics is evolving to find better robotic solutions and overcome the difficulties the industry poses every day. Examples are the development of different architectures for robots, in order to search for appropriate solutions depending on the application.

Serial robots is the preferred architecture to be used in the industry because of their large workspace and easy programming, but once the application characteristics demand bigger sizes of these machines, then the drawbacks start to appear and the development costs increases considerably. For serial robots, the main drawbacks are elasticity/flexibility issues, low speed, big actuators and motors, and small payload/robot weight ratio. At this stage the parallel robots take advantage of the above mentioned, explaining why these days the industry is very interested on new developments of parallel machines.

One of the machines that apply into the area of parallel robots is the Gantry-Tau Parallel Kinematic Machine. Developed by ABB, aims for a low cost robot able to develop high speed, very high stiffness and high payload/robot weight ratio. Though, the main drawback is related to the complexity of its structure, the task results of this project represent how the development of a new Model-Based control increases the performance when the physical system has been considered.

Using nowadays procedures for robots kinematics modelling, the set-up of the system equations will contribute to obtain the system behaviour. Then, using simulation programs like SimulationX and Matlab/Simulink, we apply advanced control techniques at an early stage. This allows a better comprehension of how the system reacts for different conditions, without the necessity of testing the preliminary structures on the real system. As the University of Agder (UiA) has its own prototype of the Gantry-Tau machine, the objectives of this project are expanded up to the implementation stage.

However, the main tasks are focused for a specific robot, the UiA Gantry-Tau, meaning that the work presented here also contributes to ease the understanding and design of advanced controller structures when applied in robotics, especially with complex parallel robots.

In this thesis, it is presented the basics for the modelling of the Gantry-Tau, mainly revisiting different parts already developed in past works, with the difference of the implementation of verification and optimization by means of software.

Chapter 1 gives a brief literature review of parallel robots and the Gantry-Tau.

Chapter 2 introduces to the kinematics of the Gantry-Tau, explaining in a simple way the motion behaviour of such complex system. Additionally, some simulations are demonstrated along the sections of this chapter, with the objective of the verification of the model.

Chapter 3 explains the modelling of a flexible path planner for the robot, which is important to make control investigations and verify the accuracy of the control model.

The model-based control system is explained in Chapter 4, where is demonstrated by means of several simulations and the outputs obtained from the tests with Gantry-Tau prototype.

The project is summarized in Chapter 5, offering a conclusion of the scope of the project, suggesting improvements and the recommended future directions of the Gantry-Tau.

Chapter 1

The Gantry-Tau

Robots are mechanical machines with elements constituted mainly by mechanisms, actuators, links connected by kinematic joints, etc., that feature a movable construction under some structure of control. Their appearance varies from model to model and many definitions try to describe precisely if certain machine could be considered as a robot.

According to a scientific interpretation, the robot is seen as a machine that, independently of its exterior, is able to modify the environment in which it operates. This is accomplished by carrying out actions that are conditioned by certain rules of behaviour in the machine as well as by the information the robot acquires regarding its environment and operation. In fact, robotics is commonly defined as the science studying the intelligent connection between perception and action.

A robotic system consists of manipulator, a wrist, an end-effector, actuators, sensors, controllers, processors, and software. One of the main components of the robots architectures described previously is the link, which is any rigid body that make up a robot. Another main component is the joint, being the connection between two links that offers relative motion.

Robots are commonly divided by architecture and the application they develop. From movies, we know robots as humanoids with high autonomy but yet limited mobility. Also, we have seen robots in the industry as a big arm-like shape with a hand conditioned to the application. The last mentioned it is known as serial robot, defined as a system of rigid bodies in which each member is connected to two others, except for the first and last members that are connected to only one other member [8].

There is also another architecture known as a parallel robots, in which a member called a mobile platform is connected to a reference element by at least two links that can be elementary or complex [2].

This thesis centres its work in the Gantry-Tau, which is a parallel robot with a 3-2-1 link configuration. This robot was developed by ABB to overcome the problems that serial robots have like flexibility issues, low speed and low payload/robot weight ratio. The Gantry-Tau takes advantage of it parallel structure to increase the stiffness and reduce its mass. Also, this allows for a better distribution of loads through the links.

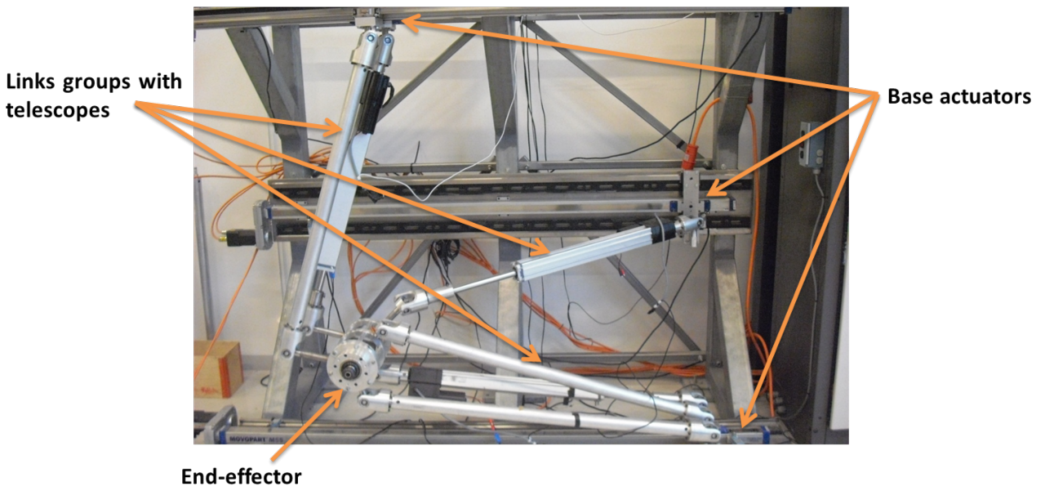


Figure 1.1: *The UiA Gantry-Tau and its main parts.*

The Gantry-Tau PKM shown in the last figure is available at the workshop of the University of Agder (UiA). This model is utilized as a prototype and it's the only model with a vertical distribution. It is composed of 6 links that meet at the manipulated platform (end-effector) of the robot. These links are distributed along three Base actuators.

The UiA Gantry-Tau has a mobility of 5 Degrees of Freedom (DoF) when the telescopic links have variable length. If these telescopic links use a fixed length, then the robots limits to 3 DoF. The robot also has an additional redundant telescopic link that adds additional workspace for the robot.

The telescopic links and the base actuators are Prismatic joints, and by definition these are referred as sliding joints that allow on one degree of freedom.

The remaining joints located at both ends of each link are of the type Universal, which can be referred as a Cardan or Hooke joint with two degrees of freedom.

Chapter 2

Kinematics of the Gantry-Tau

The development of a model-Based control structure for the Gantry-Tau parallel kinematic machine (PKM) requires following the basic stages for modelling a robot. These stages are the robot's kinematics, dynamics and control. According to [3], kinematics in robotics analyses the geometric motion of manipulators, where the kinematic descriptions are utilized to set-up the equations for dynamics and control. Therefore, a good kinematic model is important to have an early understanding of the Gantry-Tau's motion behaviour due to the complex links/joints configuration.

This chapter deals with a comprehensive modelling for the Gantry-Tau, similar to those used in previous investigations of the same model (which will be mentioned gradually). Though, the objective is to extend this formulation to verification phases by means of simulation software. The main program of the robot will be created in Matlab/Simulink and later verified using SimulationX. The latter uses a multi-body modelling system, capable of providing reliable kinematic and dynamic results.

2.1 Calculation of the Gantry-Tau's Degrees of Freedom

Since the Gantry-Tau has a complex structure, it is essential to clarify the mobility of the machine before starting the kinematic formulation, as it might not be obvious at a first look.

The Gantry-Tau is a parallel robot (also PKM) with a spatial-base structure, therefore, the use of classical formulas for the calculation of the Degrees of Freedom (DoF) or mobility may give wrong results for the actual case [4].

We introduce the Chebychev-Grübler-Kutzbach formula, for the evaluation of mobility on a spatial machine with parallel kinematic chains [11],

$$l = d\,(n-g-1) + \sum_{i=1}^{g} f_i \tag{2.1}$$

where:
l is the DoF of the kinematic chain,
d is the DoF of each unconstrained individual body (6 for the spatial case, 3 for the planar case),
n is the number of rigid bodies or links in the chain,
g is the number of joints, and
f_i is the number of DoF allowed by the ith joint.

The last equation can be further simplified by assuming initially two known bodies on the machine (base and platform), L as the number of parallel legs, and the mobility of the ith leg f_{l_i}. Now, Equation 2.1 can be rewritten as follows:

$$\begin{aligned} l &= 6\left[2+\sum_{i=1}^{L}(f_{l_i}-1)-\sum_{i=1}^{L}f_{l_i}-1\right]+\sum_{i=1}^{g}f_i \\ &= 6-6L+\sum_{i=1}^{g}f_i \end{aligned} \tag{2.2}$$

Equation 2.2 can be used to calculate the mobility of the Gantry-Tau. For this, we make use of the links and joints representation in Figure 2.1, where the number of parallel legs L is equal to 6, and the total number of DoF for the 17 joints (variable g) is equal to 29; each one of the 5 prismatic joints (3 base actuators and 2 telescopic links) offers 1 DoF, each one of the 6 base joints offers 2 DoF, and each one of the 6 platform joints offers 3 DoF.

Thus, the mobility of the Gantry-Tau will be $l = 6 - 6(6) + [(5\times 1) + (6\times 2) + (6\times 3] = 5$ DoF.

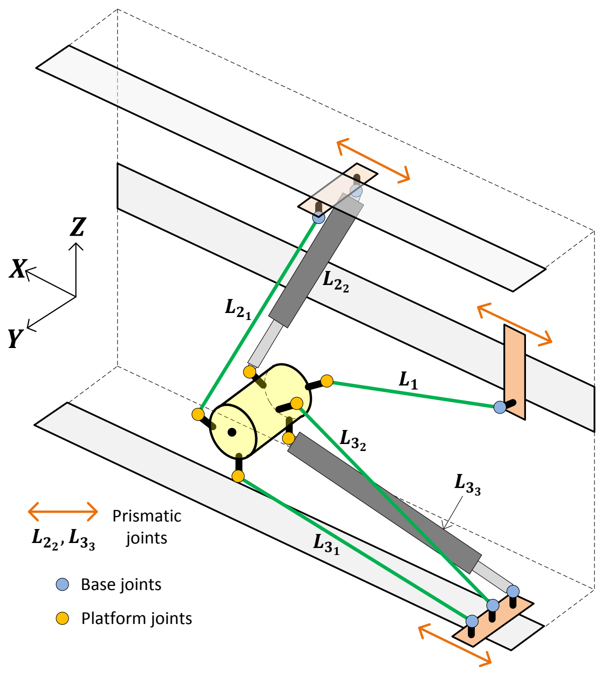


Figure 2.1: *The links and joints of the Gantry-Tau PKM.*

An important comment to say about the mobility of the Gantry-Tau, is when the telescopic links

L_{2_2} and L_{3_3} use a fixed length, then by means of Equation 2.2 we easily find 3 DoF for this configuration.

2.2 Inverse Kinematics

Defining the Gantry-Tau motion behaviour can be accomplished by using two approaches; the Forward Kinematics (FK) and Inverse Kinematics (IK). According to [7], the forward kinematics address the problem of determining the position of the end-effector based on its actuated joint coordinates, while the inverse kinematics establishes the values for the joints coordinates based on the end-effector configuration.

The IK is the best approach to use for several reasons. One of them is an easier and straightforward computation for the case of parallel robots [4]. Likewise, another reason is seen at practice, where the robot's operator or programmer inputs working coordinates (path and trajectories) for the end-effector's tool. Theoretically speaking, parallel robots with many DoF have a complex structure; the direct approach (FK) comprises many unknown variables to find [5].

The procedure to establish the IK for the Gantry-Tau PKM is firstly based on a *simplified model*, which aims to quickly understand the kinematic behaviour of the robot. Later, when a minimum number of kinematic parameters are found, the formulation is reformed to the *general case*, meaning that, the consideration of all significant machine offsets and geometric profiles must be taken into account.

The Gantry-Tau PKM as shown in Figures 1.1 and 2.1 has two possible configurations; 3 DoF when the telescopic links of arms 2 and 3 (L_{2_2} and L_{3_3} respectively) use fixed lengths, and 5 DoF when the mentioned telescopic links use variable lengths. Also, the robot has an additional redundant telescopic link as arm 1 (L_1); allowing a larger dexterity and workspace.

2.2.1 Gantry-Tau with 3 DoF (simplified case)

To get a first understanding of the IK of the Gantry-Tau, the use of a simplified model reduces the geometric complexity of the structure. For this case, simple expressions can be obtained with less dependency of unknown variables.

Figure 2.3 shows the simplified model, with the use of 3 links (L_1, L_2 and L_3) sharing the same joint with position (X_p, Y_p, Z_p) on the manipulated platform.

The platform joint where the links meet together is not realistic, but is mainly used as an auxiliary joint to avoid the offsets at this stage and understand the motion situation of the robot. The global coordinate system (X_G, Y_G, Z_G) is placed at the right-end of the base actuator 3 with positive directions as shown by its coordinate arrows. Variables q_1, q_2 and q_3 represent the position of the base actuators 1, 2 and 3, respectively in the X direction. The Tool Centre Point (TCP) is displaced an y_{offset} distance from the platform joint.

Before proceeding, an important remark to say is about the machine's 8 possible solutions for a same end-effector configuration (see Figure 2.2). The multiple solutions is due to the geometric description of the structure and the wide track for displacement of the base actuators.

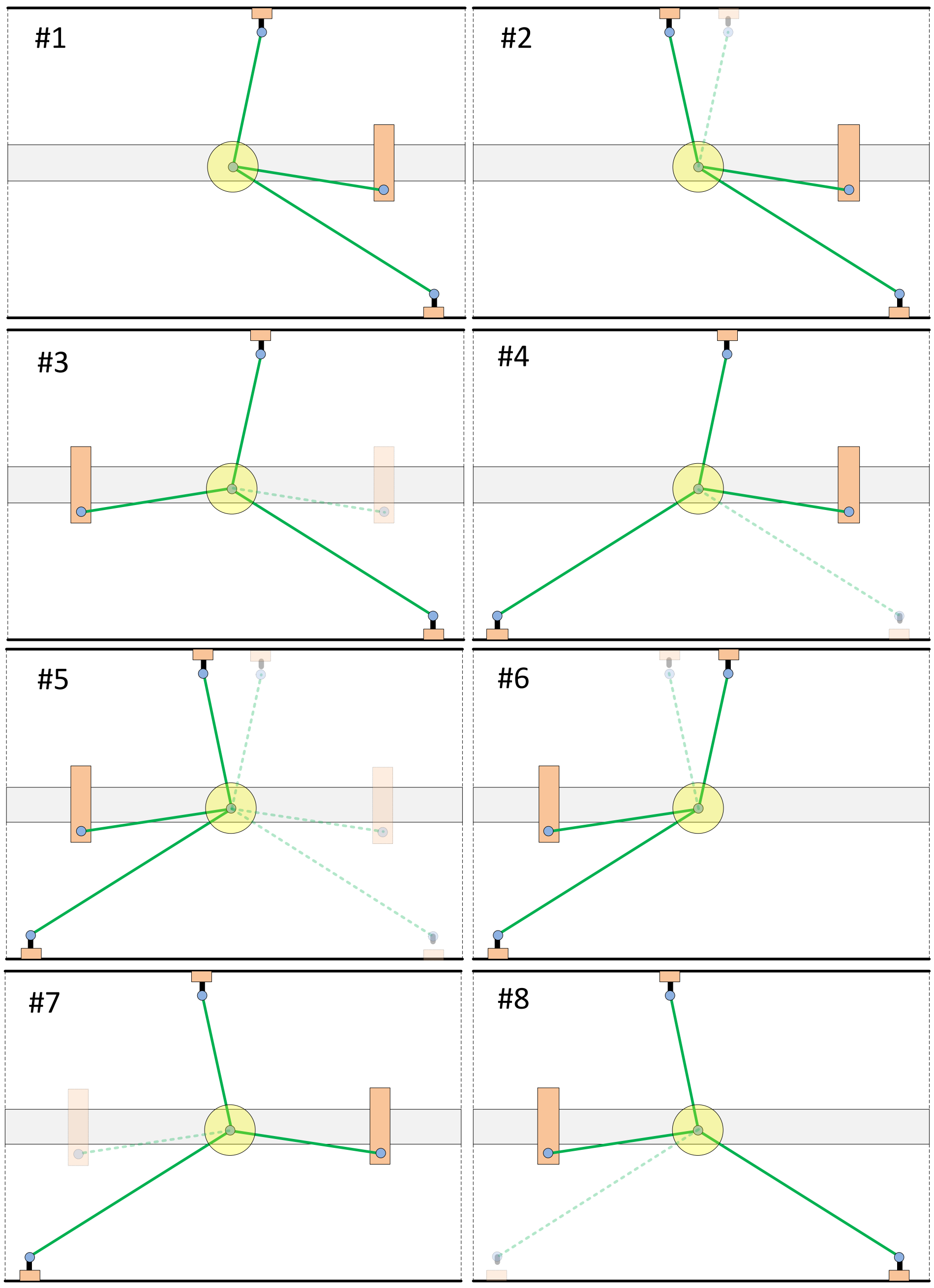


Figure 2.2: *Possible solutions for a same end-effector configuration (the ghost lines represent where the link position was before).*

To start solving the IK and obtaining analytical relations adapted to the possible solutions of each base actuator, the length equations for links L_1, L_2 and L_3 can be obtained by inspection of Figure 2.3, thus:

$$(L_1)^2 = (X_p - X_1)^2 + (Y_p - Y_1)^2 + (Z_p - Z_1)^2 \tag{2.3}$$
$$(L_2)^2 = (X_p - X_2)^2 + (Y_p - Y_2)^2 + (Z_p - Z_2)^2 \tag{2.4}$$
$$(L_3)^2 = (X_p - X_3)^2 + (Y_p - Y_3)^2 + (Z_p - Z_3)^2 \tag{2.5}$$

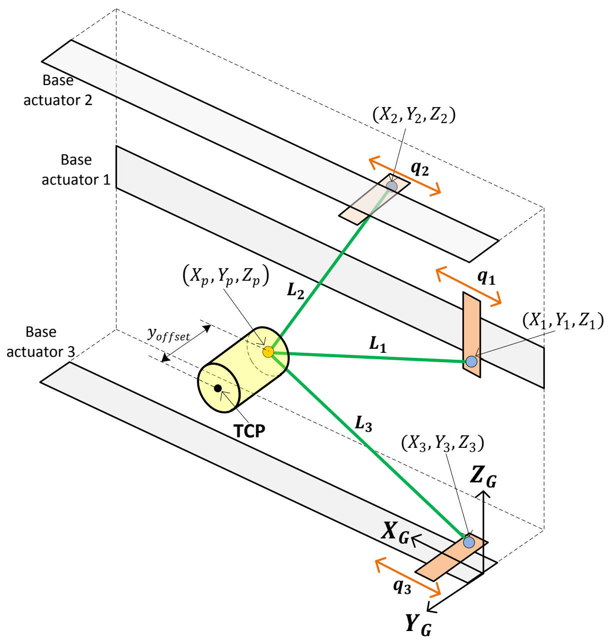


Figure 2.3: *Simplified model of the Gantry-Tau.*

Since $X_1 = q_1$, $X_2 = q_2$ and $X_3 = q_3$, we find expressions of each variable assigned to the base actuators and their possible solutions (denoted by the $\pm$ notation),

$$q_1 = X_p \pm \sqrt{(L_1)^2 - (Y_p - Y_1)^2 - (Z_p - Z_1)^2} \tag{2.6}$$
$$q_2 = X_p \pm \sqrt{(L_2)^2 - (Y_p - Y_2)^2 - (Z_p - Z_2)^2} \tag{2.7}$$
$$q_3 = X_p \pm \sqrt{(L_3)^2 - (Y_p - Y_3)^2 - (Z_p - Z_3)^2} \tag{2.8}$$

where a $+$ sign is used for a left-hand solution of the actuator, and the $-$ sign for a right-hand solution. Also, the lengths L_1, L_2 and L_3 have fixed values, the same occurs with the joints' coordinate values (Y_1, Z_1), (Y_2, Z_2) and (Y_3, Z_3); since the base actuators only vary position in the X direction. The Equations 2.6, 2.7 and 2.8 will give the displacement for each actuator based on the auxiliary joint position (X_p, Y_p, Z_p), and later on the TCP assigned coordinate.

The last equations do not provide the rotation of the manipulated platform around the Y axis for this simplified case. By looking at the XZ plane of the Gantry-Tau (see Figure 2.4) we introduce β as the angle formed between L_{3XZ} (vertical projection of link L_3) and the Z axis of base joint 3.

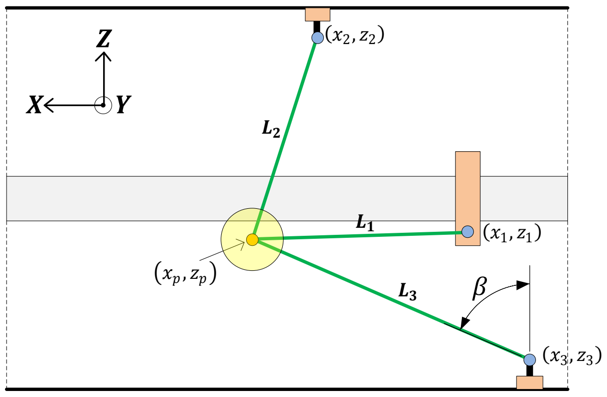


Figure 2.4: *The simplified Gantry-Tau as seen in the XZ plane.*

From the last figure we determine the geometric expression of β as

$$\cos(\beta) = \frac{Z_p - Z_3}{L_{3XZ}} \tag{2.9}$$

where L_{3XZ} is the projection of link L_3 on the XZ plane (see also Figure 2.5),

$$L_{3XZ} = \sqrt{(L_3)^2 - (Y_p - Y_3)^2} \tag{2.10}$$

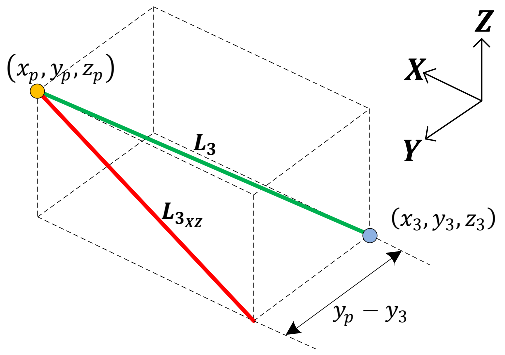


Figure 2.5: *Projection of link L_3 on the XZ plane.*

Then, substituting the Equation 2.10 in 2.9, we obtain the β angle,

$$\beta = \arccos\left(\frac{Z_p - Z_3}{\sqrt{(L_3)^2 - (Y_p - Y_3)^2}}\right) \tag{2.11}$$

where all parameters in the expression are known.

Now, we need to make use of the β angle to compute the orientation of the manipulated platform; providing the redundant rotation around the Y axis. Then, is necessary the consideration of a new point R_p given by the radius R of the manipulated platform, as shown in Figure 2.6. Using transformation matrices or *Homogeneous Transformations* as described in [9], we can easily find the new location of $R_p = [0, 0, R, 1]^T$ in global coordinates from the local frame of the platform, also providing the orientation of the manipulated platform.

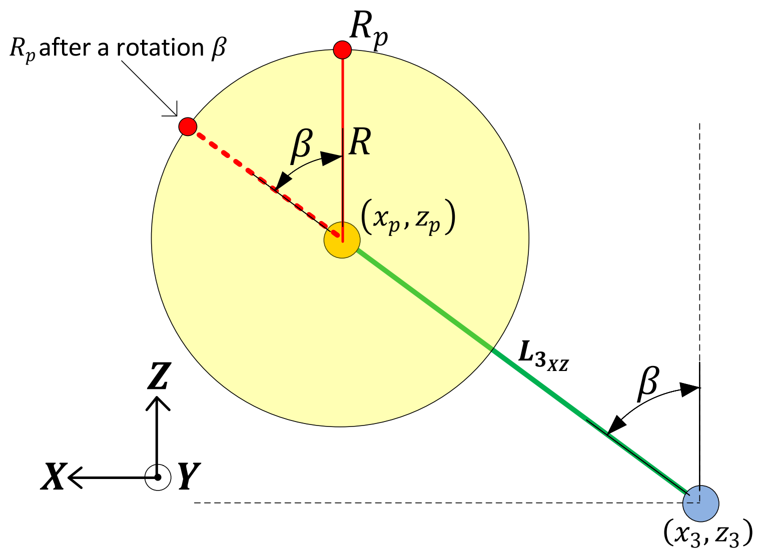


Figure 2.6: *Relation of the β angle to find the orientation of the platform given by the point R_p.*

Initially, and following the definition used in the same reference, we define the *rotation matrix* in the Y plane given by

$$Rot_{Y,\beta} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.12)$$

Let T be the *general transformation matrix* of the platform joint (X_p, Y_p, Z_p)

$$T = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & X_p \\ 0 & 1 & 0 & Y_p \\ -\sin(\beta) & 0 & \cos(\beta) & Z_p \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.13)$$

Then, we find the new position of R_p after a rotation β. As a result we have

$$\begin{aligned} R'_p &= TR_p \\ &= \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & X_p \\ 0 & 1 & 0 & Y_p \\ -\sin(\beta) & 0 & \cos(\beta) & Z_p \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ R \\ 1 \end{bmatrix} \end{aligned} \quad (2.14)$$

For example, if $\beta = 90°$, we obtain the following position for R'_p

$$R'_p = \begin{bmatrix} R + X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \tag{2.15}$$

Now it's time to consider the TCP location in space. By looking again at Figure 2.6, a distance y_{offset} exists between the platform joint and the TCP. It is important to remember that, the TCP is an user-specific coordinate for the IK case.

The TCP coordinate with the inclusion of y_{offset} is as follows:

$$\begin{bmatrix} X_{TCP} \\ Y_{TCP} \\ Z_{TCP} \end{bmatrix} = \begin{bmatrix} X_p \\ Y_p + y_{offset} \\ Z_p \end{bmatrix} \tag{2.16}$$

2.2.2 Gantry-Tau with 3 DoF (general case)

Once understood the basic kinematic relations in the previous section, the general case for the 3 DoF configuration is very similar to solve. The addition of more links connected to the manipulated platform and existing offsets require more attention to find the relations; though, some additional considerations ease the approach. Also, in this case it is essential to use the *Homogeneous Transformations* to easily find the position in space of each joint on the platform.

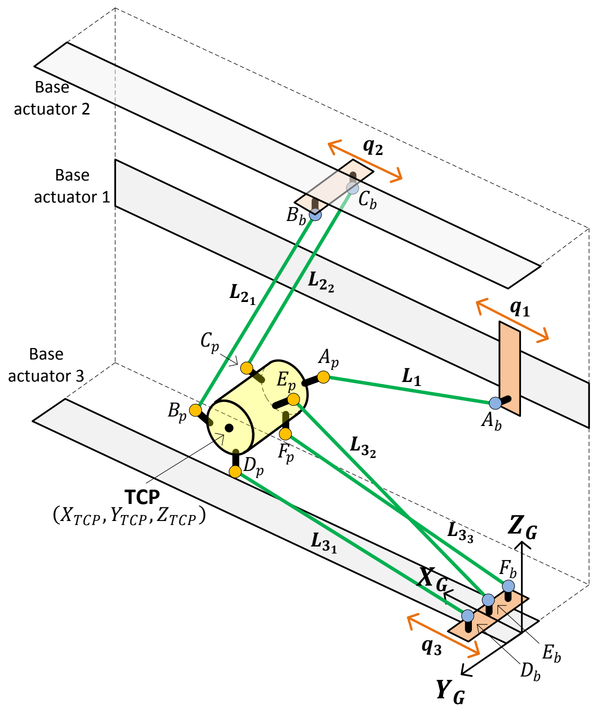


Figure 2.7: *The Gantry-Tau for the 3 DoF general case with notations.*

Before starting the kinematic formulation, it is necessary to add notation to the different parts of the robot. The links, passive joints, active joints (actuators), Tool Centre Point (TCP), the global coordinate system, etc., are named as shown in Figure 2.7.

The links of arm 2 (L_{2_1} and L_{2_2}) are parallel at all times and have the same length, thus it's enough by defining the length equation for one of them. Something similar occurs with the links of arm 3, sharing only the same length and forming a triangular linkage; that later on, defines the rotation angle β of the platform. Then, we have the following length equations for L_1, $L_{2_1}$and L_{3_1},

$$
\begin{aligned}
(L_1)^2 &= (X_{A_p} - X_{A_b})^2 + (Y_{A_p} - Y_{A_b})^2 + (Z_{A_p} - Z_{A_b})^2 && (2.17)\\
(L_{2_1})^2 &= (X_{B_p} - X_{B_b})^2 + (Y_{B_p} - Y_{B_b})^2 + (Z_{B_p} - Z_{B_b})^2 && (2.18)\\
(L_{3_1})^2 &= (X_{D_p} - X_{D_b})^2 + (Y_{D_p} - Y_{D_b})^2 + (Z_{D_p} - Z_{D_b})^2 && (2.19)
\end{aligned}
$$

Since $X_{A_b} = q_1$, $X_{B_b} = q_2$ and $X_{D_b} = q_3$, we find expressions of each variable assigned to the base actuators,

$$
\begin{aligned}
q_1 &= X_{A_p} \pm \sqrt{(L_1)^2 - (Y_{A_p} - Y_{A_b})^2 - (Z_{A_p} - Z_{A_b})^2} && (2.20)\\
q_2 &= X_{B_p} \pm \sqrt{(L_{2_1})^2 - (Y_{B_p} - Y_{B_b})^2 - (Z_{B_p} - Z_{B_b})^2} && (2.21)\\
q_3 &= X_{D_p} \pm \sqrt{(L_{3_1})^2 - (Y_{D_p} - Y_{D_b})^2 - (Z_{D_p} - Z_{D_b})^2} && (2.22)
\end{aligned}
$$

where the $+$ sign is used for a left-hand solution of the actuator, and the $-$ sign for a right-hand solution (refer to previous section and Figure 2.2). Also, the lengths L_1, L_{2_1} and L_{3_1} have fixed values, the same occurs with the joints' Y and Z coordinate values of the base actuators; since they only vary position along the X direction. Though, it is not yet possible to find their position values because the platform joints X_{A_p}, X_{B_p} and X_{D_p} have offsets with respect to the TCP coordinates. For this, we must find the β angle obtained from an auxiliary link L_{aux}, constructed from the triangular linkage formed between the platform and links of arm 3; more specifically constructed from base joint D_b intersecting in a vertical projection of platform joints D_p and E_p. This line can be seen in Figure 2.8.

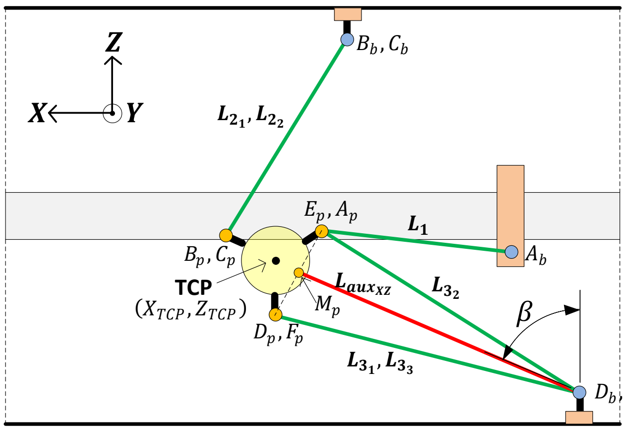


Figure 2.8: *Auxiliary line L_{aux}, in this case seen at the XZ plane.*

Now, we can make further geometrical relations with the auxiliary line L_{aux}, the triangular linkage of arm 3 and the manipulated platform. As shown in Figure 2.9, an intersection point M_p is formed by L_{aux} and a projection line between the platform joints D_p and E_p. Also, we have additional lengths L_4 and L_5 that will serve good in finding a relation for β.

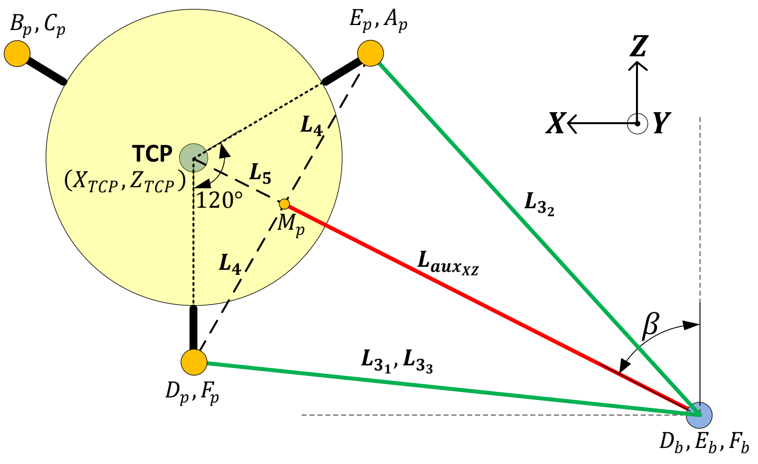


Figure 2.9: *Further geometrical relations between platform and linkage of arm 3.*

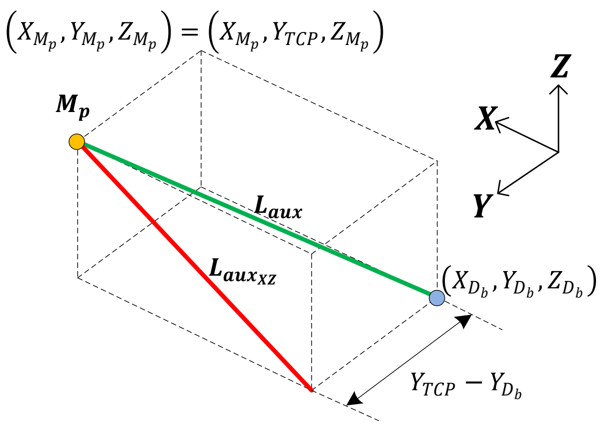
The last extension allows us to make use of simple geometrical relations (right angle triangles, Pythagoras and cosine theorems) and therefore, ease the approach in finding the β angle.

By looking again at Figure 2.9, we have

$$\cos(\beta) = \frac{Z_{TCP} - Z_{D_b}}{L_{auxXZ} + L_5} \tag{2.23}$$

We find L_{auxXZ} as the projection on the XZ plane of L_{aux} (see Figure 2.10),

$$L_{auxXZ} = \sqrt{(L_{aux})^2 - (Y_{TCP} - Y_{D_b})^2} \tag{2.24}$$



$(X_{M_p}, Y_{M_p}, Z_{M_p}) = (X_{M_p}, Y_{TCP}, Z_{M_p})$
M_p
L_{aux}
$L_{aux_{XZ}}$
$(X_{D_b}, Y_{D_b}, Z_{D_b})$
$Y_{TCP} - Y_{D_b}$
Z
X
Y

Figure 2.10: *Projection of the auxiliary link L_{aux} on the XZ plane.*

and we have $L_{aux} = \sqrt{(L_{3_1})^2 - (L_4)^2}$, then

$$L_{auxXZ} = \sqrt{(L_{3_1})^2 - (L_4)^2 - (Y_{TCP} - Y_{D_b})^2} \tag{2.25}$$

The length L_4 can be easily found from the cosine theorem and by inspection of Figure 2.9,

$$\begin{aligned} 2L_4 &= \sqrt{(Z_{D_p})^2 + (Z_{E_p})^2 - 2Z_{D_p}Z_{E_p}\cos(120°)} \\ L_4 &= \frac{1}{2}\sqrt{(Z_{D_p})^2 + (Z_{E_p})^2 - 2Z_{D_p}Z_{E_p}\cos(120°)} \end{aligned} \tag{2.26}$$

And finally we find L_5 as

$$L_5 = \sqrt{(Z_{D_p})^2 - (L_4)^2} \tag{2.27}$$

where at this point we have found all the variables to calculate the appropriate β angle from Equation 2.23:

$$\beta = \pm\arccos\left(\frac{Z_{TCP} - Z_{D_b}}{L_{auxXZ} + L_5}\right) \tag{2.28}$$

where a $+$ sign is used for a right-hand solution of base actuator 3, and the $-$ sign for a left-hand solution.

Now, we need to make use of the β angle to find the orientation of the manipulated platform and its joints coordinates in the global frame. The latter will allow the calculation of displacements for the base actuators.

Before using the homogeneous transformations, it's necessary to show how the offsets of platform joints are distributed. In Figure 2.11, we can see how this distribution is made in the platform's local frame, and later expressed as column vectors.

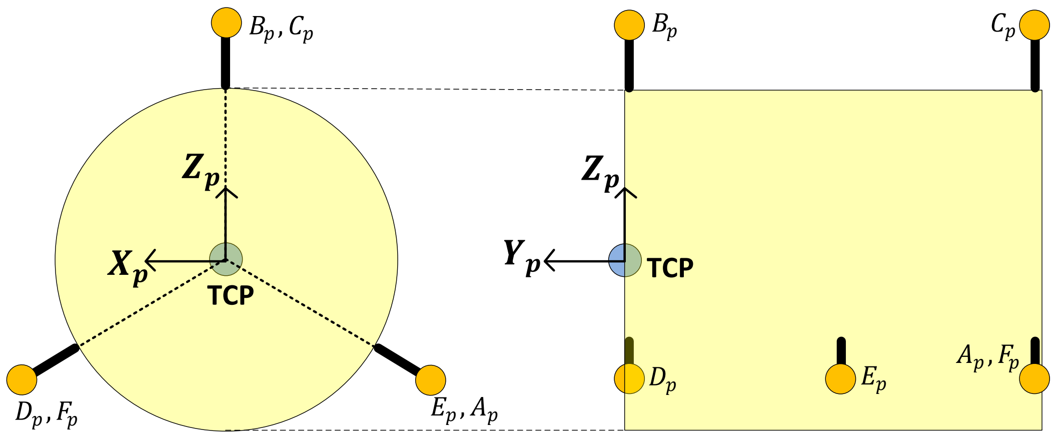


Figure 2.11: *Joints distribution on the platform.*

$$A_{p_{offset}} = \left[X_{A_{p_{offset}}}, Y_{A_{p_{offset}}}, Z_{A_{p_{offset}}}, 1\right]^T \tag{2.29}$$

$$B_{p_{offset}} = \left[X_{B_{p_{offset}}}, Y_{B_{p_{offset}}}, Z_{B_{p_{offset}}}, 1\right]^T \tag{2.30}$$

$$C_{p_{offset}} = \left[X_{C_{p_{offset}}}, Y_{C_{p_{offset}}}, Z_{C_{p_{offset}}}, 1\right]^T \tag{2.31}$$

$$D_{p_{offset}} = \left[X_{D_{p_{offset}}}, Y_{D_{p_{offset}}}, Z_{D_{p_{offset}}}, 1\right]^T \tag{2.32}$$

$$E_{p_{offset}} = \left[X_{E_{p_{offset}}}, Y_{E_{p_{offset}}}, Z_{E_{p_{offset}}}, 1\right]^T \tag{2.33}$$

$$F_{p_{offset}} = \left[X_{F_{p_{offset}}}, Y_{F_{p_{offset}}}, Z_{F_{p_{offset}}}, 1\right]^T \tag{2.34}$$

Initially, we define the rotation matrix around Y given by Equation 2.12. Then we find the position of the platform joints after a rotation of β; as an example, for platform joint A_p the position will be:

$$\begin{aligned} A'_p &= Rot_{Y,\beta} A_{p_{offset}} \\ &= \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{A_{p_{offset}}} \\ Y_{A_{p_{offset}}} \\ Z_{A_{p_{offset}}} \\ 1 \end{bmatrix} \end{aligned} \tag{2.35}$$

and the same procedure is made for the other platform joints.

Once obtained the global coordinates for each platform joint, we can calculate displacements for the base actuators using Equations 2.6, 2.7 and 2.8.

2.2.3 Gantry-Tau with 5 DoF

The Gantry-Tau is an industrial robot thought for many applications, including machining and milling applications, where the tool must have a mobility of at least 5 DoF. The actual prototype has the latter mobility and from now on, the modelling of all remaining characteristics of the Gantry-Tau will be regarding this configuration.

The difference between the 5 DoF configuration with respect to the 3 DoF configuration is two telescopic links (active prismatic joints) that working together with the base actuators offer a change of orientation for the tool point around the X and Z axes; α and γ angles respectively. Still, for every change in the trajectory and orientation of the tool point, a rotation around the Y axis (β angle) will always exist. This is the key for solving the IK of the Gantry-Tau with 5 DoF.

Beforehand, one might think the 5 DoF configuration as more complex to work out. This is true depending on the approach to take, but since this configuration has been solved in [5] and [6] in different ways, this section will revisit only one of them. The author of this thesis considers the general analytical approach as the most appropriate and optimal for obtaining accurate results.

Also, the advantage of nowadays software for analytical and symbolic calculation offers an easier way of finding the needed expressions, using at the same time optimisation techniques for the sake of reducing computation effort.

Initially, the redundant rotation β must be found. As in the IK for the 3 DoF configuration, this rotation is related to the platform, the triangular linkage of arm 3, and the displacement of base actuator 3.

In Figure 2.12 is shown the notation for each part of the Gantry-Tau with 5 DoF. For the 5 DoF

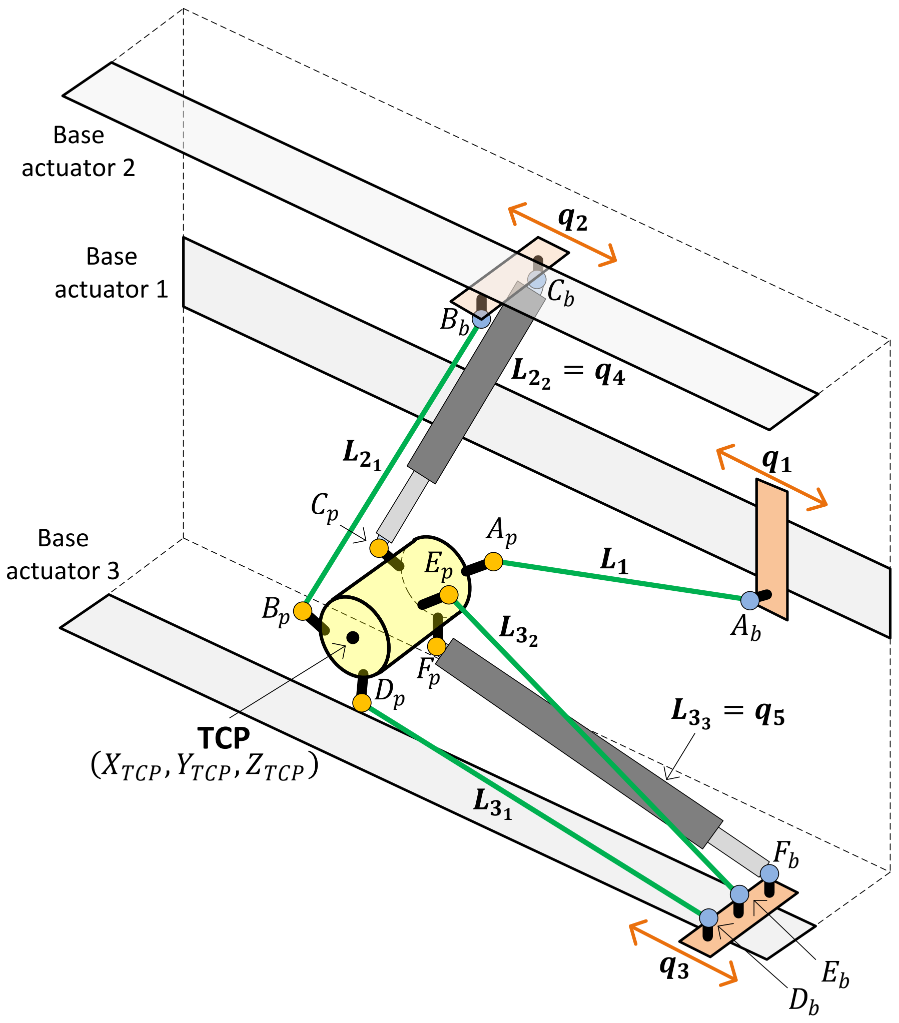


Figure 2.12: *The Gantry-Tau for the 5 DoF configuration with notations.*

configuration, the user specifies (X_{TCP},Y_{TCP},Z_{TCP},α,γ) as coordinates/orientation for the tool point, and therefore, a first step is to generate general transformation expressions for the platform joints.

According to [6], the sequence of angles $r_x - r_z - r_y$ is recommended for the calculation of orientation of the platform joints, which also must be related to the TCP in the global frame. Then, the coordinates of the platform joints after the mentioned sequence are calculated as in the following example for A_p:

$$\begin{bmatrix} X_{A_p} \\ Y_{A_p} \\ Z_{A_p} \end{bmatrix} = R \begin{bmatrix} X_{A_{p offset}} \\ Y_{A_{p offset}} \\ Z_{A_{p offset}} \end{bmatrix} + \begin{bmatrix} X_{TCP} \\ Y_{TCP} \\ Z_{TCP} \end{bmatrix} \tag{2.36}$$

where R is the rotation matrix calculated as

$$\begin{aligned} R &= R_{X,\alpha} R_{Z,\gamma} R_{Y,\beta} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \end{aligned} \tag{2.37}$$

Then, we set-up the length equations for links L_{3_1} and L_{3_2}:

$$L_{3_1}^2 = (X_{D_p} - X_{D_b})^2 + (Y_{D_p} - Y_{D_b})^2 + (Z_{D_p} - Z_{D_b})^2 \tag{2.38}$$

$$L_{3_2}^2 = (X_{E_p} - X_{E_b})^2 + (Y_{E_p} - Y_{E_b})^2 + (Z_{E_p} - Z_{E_b})^2 \tag{2.39}$$

where $q_3 = X_{D_b} = X_{E_b}$. When substituting the platform joints coordinates of D_p and E_p (calculated as in Equation 2.36) into Equations 2.38 and 2.39 respectively, we find two unknown variables; the displacement of q_3 and the rotation around the Y axis (β). Since these equations should give the same length result, we find the variable q_3 from their difference, thus

$$L_{3_1}^2 - L_{3_2}^2 = X_{D_p}^2 - X_{E_p}^2 + 2q_3(X_{E_p} - X_{D_p}) + U - V \tag{2.40}$$

then

$$q_3 = \frac{L_{3_1}^2 - L_{3_2}^2 - X_{D_p}^2 + X_{E_p}^2 - U + V}{2(X_{E_p} - X_{D_p})} \tag{2.41}$$

having $U = (Y_{D_p} - Y_{D_b})^2 + (Z_{D_p} - Z_{D_b})^2$ and $V = (Y_{E_p} - Y_{E_b})^2 + (Z_{E_p} - Z_{E_b})^2$. Later, Equation 2.41 can be substituted in Equation 2.38 resulting in

$$\left(X_{D_p} - \frac{L_{3_1}^2 - L_{3_2}^2 - X_{D_p}^2 + X_{E_p}^2 - U + V}{2(X_{E_p} - X_{D_p})} \right)^2 + U - L_{3_1}^2 = 0 \tag{2.42}$$

Equation 2.42 is a polynomial of eight order where the only unknown is the redundant rotation β. Though, β is found in sine and cosine terms, leading to a difficult to solve polynomial. As [6] suggests, it's necessary to use the Weierstrass substitution as:

$$W = \tan\left(\frac{\beta}{2}\right) \tag{2.43}$$

and its trigonometric equivalences

$$\sin(\beta) = \frac{2W}{1+W^2} \tag{2.44}$$
$$\cos(\beta) = \frac{1-W^2}{1+W^2} \tag{2.45}$$

In Appendix A is shown an analytical way (using Maple) of finding the coefficients of the polynomial 2.42, once the relations 2.44 and 2.45 are substituted. Later, the possible solutions are best found in a numerical way. Since the polynomial is of eight order, this results in eight solutions. Now is needed to replace W by the β angle, this is possible by rearranging Equation 2.43 as

$$\beta = 2\tan^{-1}(W) \tag{2.46}$$

Still, after the replacement of W, the polynomial results in 8 solutions, but this time for β. One way of finding the appropriate β angle is to keep track of the lengths of links L_{3_1} and L_{3_2} after using β as a possible solution.

Once β is found, the calculation of the displacements of the base actuators is made from the IK for q_1, q_2 and q_3. Then we have

$$q_1 = X_{A_p} \pm \sqrt{(L_1)^2 - (Y_{A_p} - Y_{A_b})^2 - (Z_{A_p} - Z_{A_b})^2} \tag{2.47}$$
$$q_2 = X_{B_p} \pm \sqrt{(L_{2_1})^2 - (Y_{B_p} - Y_{B_b})^2 - (Z_{B_p} - Z_{B_b})^2} \tag{2.48}$$
$$q_3 = X_{D_p} \pm \sqrt{(L_{3_1})^2 - (Y_{D_p} - Y_{D_b})^2 - (Z_{D_p} - Z_{D_b})^2} \tag{2.49}$$

where a $+$ sign is used for a left-hand solution of the actuator, and the $-$ sign for a right-hand solution.

Finally, the displacement of variables q_4 and q_5 are found using the length equations for their respective telescopic links:

$$q_4 = \sqrt{(X_{C_p} - X_{C_b})^2 + (Y_{C_p} - Y_{C_b})^2 + (Z_{C_p} - Z_{C_b})^2} \tag{2.50}$$
$$q_5 = \sqrt{(X_{F_p} - X_{F_b})^2 + (Y_{F_p} - Y_{F_b})^2 + (Z_{F_p} - Z_{F_b})^2} \tag{2.51}$$

2.2.4 Simulations

Part of this work is to include tests of the model-based system with the actual prototype. Thus, it is worthy to do stepwise the creation of the robot's program and verify it by means of simulations.

The main program is initially modelled in Matlab, and later extended to Simulink to be used in the prototype. Also, a simulation model of the robot is made in SimulationX using its multi-body system, enabling reliable results and the possibility of easily changing the kinematics approach. In this case the model in SimulationX uses the IK approach and can be seen in Appendix D.

The simulations demonstrated in this section are regarding the 5 DoF configuration of the Gantry-Tau.

Simulation 1:
A Matlab function for the Gantry-Tau was made, having the following input form:

```
IK_5DOF(X_TCP,Y_TCP,Z_TCP,alpha,gamma,ba1,ba2,ba3)
```

where `X_TCP`, `Y_TCP` and `Z_TCP` are given in meters, `alpha` and `gamma` in radians, and `ba1`, `ba2` and `ba3` represent the solution for the base actuators (0 for a right-hand solution and 1 for a left-hand solution).

For this test we want the tool point to have the coordinate $(1.2, 0.8, 1.0)$ without any change of orientation, and all base actuators with a right-hand solution. Then, the input function is:

```
IK_5DOF(1.2,0.8,1.0,0,0,0,0,0)
```

generating the plot as shown in Figure 2.13.

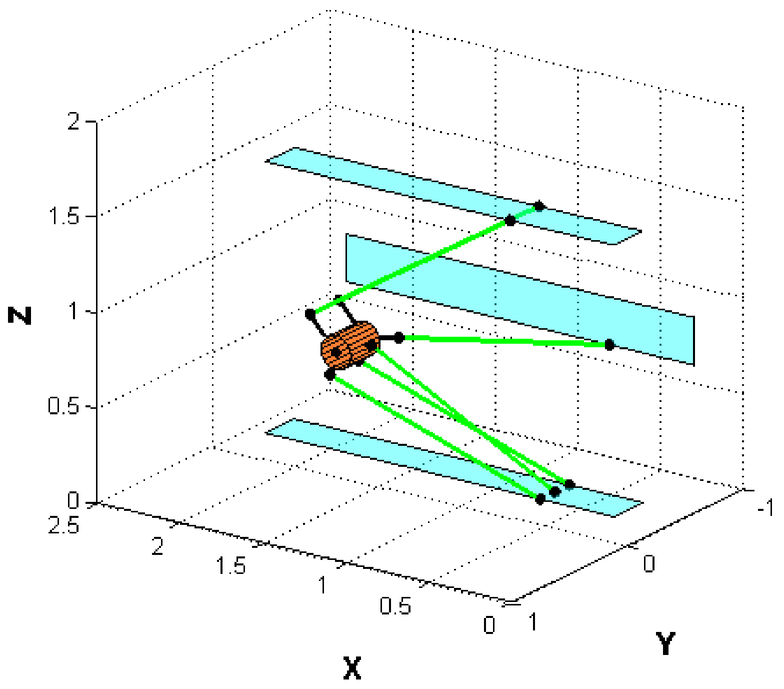
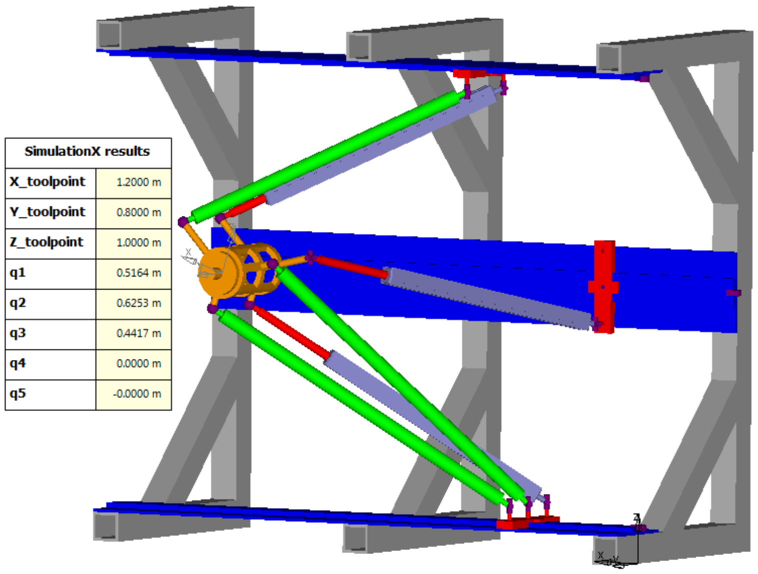


Figure 2.13: *3D plot result of the Matlab function (Test 1).*

The results of displacements obtained for the five actuators (q_1 to q_5) and position of the tool point are the same as in SimulationX, shown in Figure 2.14.



SimulationX results	
X_toolpoint	1.2000 m
Y_toolpoint	0.8000 m
Z_toolpoint	1.0000 m
q1	0.5164 m
q2	0.6253 m
q3	0.4417 m
q4	0.0000 m
q5	-0.0000 m

Figure 2.14: *3D view and results in SimulationX (Test 1).*

Simulation 2:
Now, an extension of Simulation 1 is made, this time adding only a change of orientation for the tool point, where `gamma=0.2` radians and `gamma=0.6` radians. Then, the input function will be:

```
IK_5DOF(1.2,0.8,1.0,0.2,0.6,0,0,0)
```

generating the Matlab plot as seen in Figure 2.16.

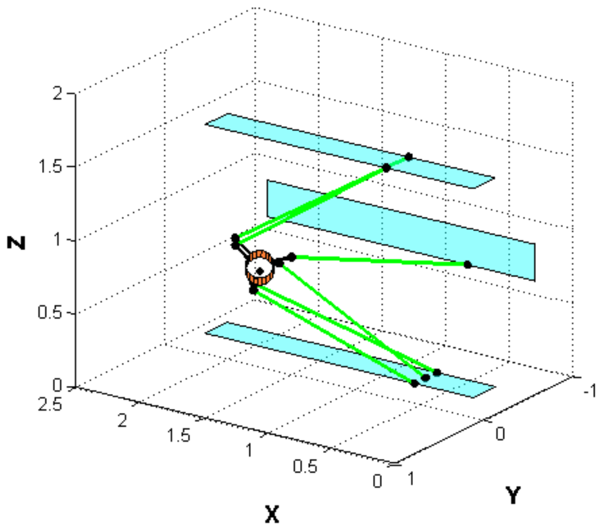
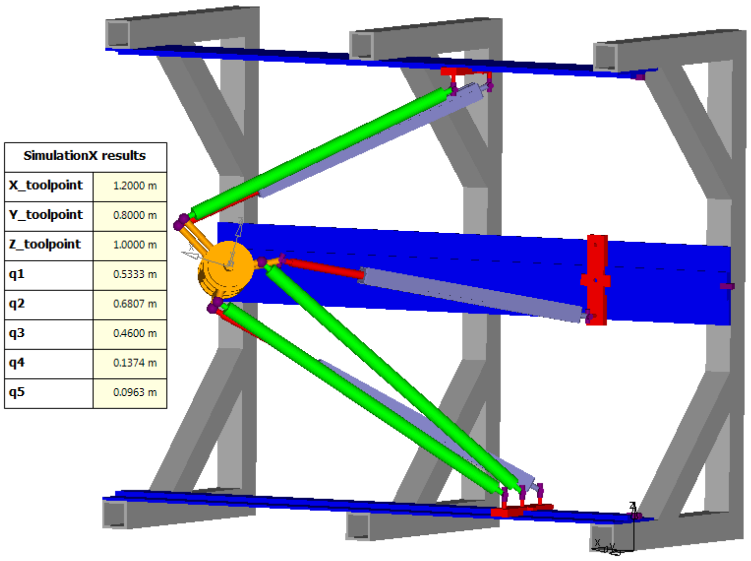


Figure 2.15: *3D plot result of the Matlab function (Test 2).*

Also in this simulation, the Matlab results for the actuators and the tool point are the same as in

SimulationX, shown in Figure 2.16.



SimulationX results	
X_toolpoint	1.2000 m
Y_toolpoint	0.8000 m
Z_toolpoint	1.0000 m
q1	0.5333 m
q2	0.6807 m
q3	0.4600 m
q4	0.1374 m
q5	0.0963 m

Figure 2.16: *3D view and results in SimulationX (Test 2).*

2.3 Inverse Velocity Kinematics

In order to control the tool point at a desired velocity, it is important to establish the relationship between the velocities of the actuators (active joints) with respect to the velocity of the manipulator. This is called the inverse velocity kinematic (IVK) problem.

As explained in [1] and [10], the inverse velocity kinematics is a linear representation between the joint velocity space and the operation joint space. The actuators velocities can be obtained as:

$$\dot{Q} = J^{-1}\dot{X} \tag{2.52}$$

where $\dot{Q}$ is a vector of the actuators velocities, J^{-1} is an inverse Jacobian matrix, and $\dot{X}$ is a vector of the manipulator's velocity.

In the case of the Gantry-Tau with 5 DoF, the actuators velocity vector is

$$\dot{Q} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \end{bmatrix} \tag{2.53}$$

the manipulator velocities is

$$\dot{X} = \begin{bmatrix} \dot{X_{TCP}} \\ \dot{Y_{TCP}} \\ \dot{Z_{TCP}} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} \tag{2.54}$$

and J^{-1} is a (5x6) inverse Jacobian matrix, thus

$$\begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \\ \dot{q_4} \\ \dot{q_5} \end{bmatrix} = \begin{bmatrix} \frac{\partial q_1}{\partial X_{TCP}} & \frac{\partial q_1}{\partial Y_{TCP}} & \frac{\partial q_1}{\partial Z_{TCP}} & \frac{\partial q_1}{\partial \alpha} & \frac{\partial q_1}{\partial \beta} & \frac{\partial q_1}{\partial \gamma} \\ \frac{\partial q_2}{\partial X_{TCP}} & \frac{\partial q_2}{\partial Y_{TCP}} & \frac{\partial q_2}{\partial Z_{TCP}} & \frac{\partial q_2}{\partial \alpha} & \frac{\partial q_2}{\partial \beta} & \frac{\partial q_2}{\partial \gamma} \\ \frac{\partial q_3}{\partial X_{TCP}} & \frac{\partial q_3}{\partial Y_{TCP}} & \frac{\partial q_3}{\partial Z_{TCP}} & \frac{\partial q_3}{\partial \alpha} & \frac{\partial q_3}{\partial \beta} & \frac{\partial q_3}{\partial \gamma} \\ \frac{\partial q_4}{\partial X_{TCP}} & \frac{\partial q_4}{\partial Y_{TCP}} & \frac{\partial q_4}{\partial Z_{TCP}} & \frac{\partial q_4}{\partial \alpha} & \frac{\partial q_4}{\partial \beta} & \frac{\partial q_4}{\partial \gamma} \\ \frac{\partial q_5}{\partial X_{TCP}} & \frac{\partial q_5}{\partial Y_{TCP}} & \frac{\partial q_5}{\partial Z_{TCP}} & \frac{\partial q_5}{\partial \alpha} & \frac{\partial q_5}{\partial \beta} & \frac{\partial q_5}{\partial \gamma} \end{bmatrix} \begin{bmatrix} \dot{X_{TCP}} \\ \dot{Y_{TCP}} \\ \dot{Z_{TCP}} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} \tag{2.55}$$

The reason for the calculation of the IVK, is to be able to define linear and angular velocities for the tool point, when for example a path planner is applied and more importantly to compute the dynamics of the robot.

The IVK of the Gantry-Tau is obtained by differentiating the Equations 2.47, 2.48, 2.49, 2.50 and 2.51. These differentiations produce too lengthy equations to show here, but instead, a stepwise program using Maple (Appendix B) was made to obtain analytically the elements of the inverse Jacobian matrix J^{-1}. This analytical result is later used in the robot's main program.

Chapter 3

Path Planning

In applications where is needed to have a predefined trajectory for the tool of the robot, it is necessary to include a program able to define motion, velocity and acceleration with respect of time. This is called path planning.

Several industrial robots are embedded with programs where an operator can input in several ways the trajectory for the end effector, and later the robot has to make use of its predefined IK formulation to find the required actuators position. Also, when developing control systems, it is worthy to implement a flexible path planner that allows the robot to be tested in several ways and therefore ease the study of the accuracy of the control system.

This chapter explains the theory of the path planner to use in the model-based simulations and tests with the prototype.

The path planner used in this work is called Cartesian Path Planning. This one, uses two basic equations for the interpolation of the path by means of a start and stop point for the end effector's tool.

By looking at Figure 3.1, we can see a path that starts from r_0 at time t_0 with a linear trajectory, later goes nearby r_1 doing a transition curve between the switching times $(t_1 - t')$ and $(t_1 + t')$, and finally arrives on r_2 at time t_2 following a linear path. The advantage of this curve representation, is that allows a smooth transition through the initial and final points.

The theory of the cartesian path planning is demonstrated in [3], but the motion equations are represented here as follows:

For the first transition $(t_0 \leq t \leq t_1 - t')$

$$r(t) = r_1 - \frac{t_1 - t}{t_1 - t_0}(r_1 - r_0) \tag{3.1}$$

For the second transition $(t_1 - t' \leq t \leq t_1 + t')$

$$r(t) \quad = \quad r_1 - \frac{t - t' - t_1}{4t'(t_1 - t_0}(r_1 - r_0) + \frac{t + t' - t_1}{4t'(t_2 - t_1}(r_2 - r_1) \tag{3.2}$$

and for the final transition $(t_1 + t' \leq t \leq t_2)$

$$r(t) \quad = \quad r_1 - \frac{t_1 - t}{t_2 - t_1}(r_2 - r_1) \tag{3.3}$$

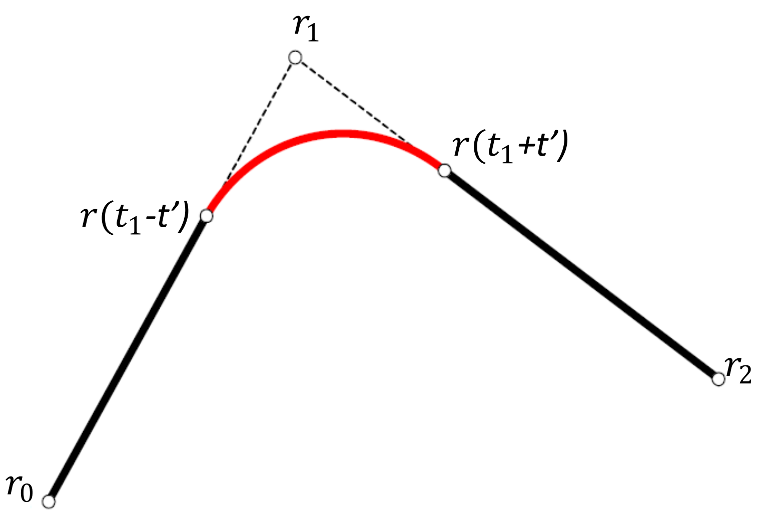


Figure 3.1: *Cartesian path example.*

Since Equations 3.1, 3.2 and 3.3 are used for interpolating the trajectory of the tool point, these do not work out a change of orientation. Though, Equation 3.1 can be slightly altered to interpolate a change of orientation.

3.1 Simulation

A cartesian path planning program was made in Matlab following the theory presented in this chapter. This program is a function of time and the points of the path are established in the following way:

```
moveL(1,:) = [X_TCP Y_TCP Z_TCP alpha gamma vel zone]
```

where `X_TCP`, `Y_TCP` and `Z_TCP` are given in meters, `alpha` and `gamma` are given in radians, `vel` is the velocity of the toolpoint in m/s, and `zone` is a fillet radius in millimetres for smoothing corners. The Matlab code of the path planning program is shown in more detail in Appendix C Now, we want to simulate the following path:

```
moveL(1,:) = [0.6542 0.7964 1.0886 0 0 0 0];
moveL(2,:) = [1.6542 0.7964 1.0886 0 0 50 60];
moveL(3,:) = [1.6542 0.7964 0.7386 0.2 0 50 60];
moveL(4,:) = [1.2542 0.7964 0.7386 0.2 0.2 50 60];
moveL(5,:) = [1.2542 0.7964 1.0886 0 0.2 50 60];
moveL(6,:) = [1.6542 0.7964 1.0886 0 0 50 60];
moveL(7,:) = [1.6542 0.7964 0.7386 0.2 0 50 60];
moveL(8,:) = [1.2542 0.7964 0.7386 0.2 0.2 50 60];
moveL(9,:) = [1.2542 0.7964 1.0886 0 0.2 50 60];
moveL(10,:) = [0.6042 0.7964 1.0886 0 0 50 0];
```

that generates the following results:

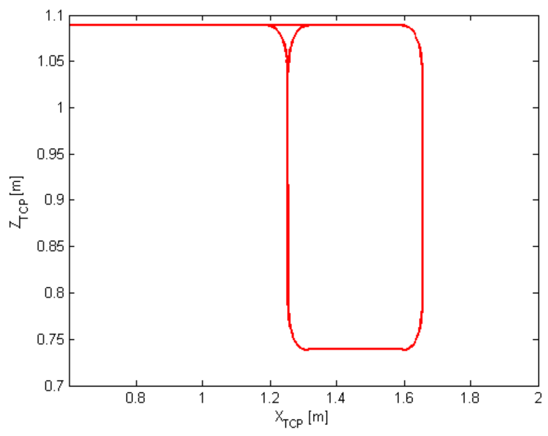


Figure 3.2: *Tool point trajectory on the XZ plane.*

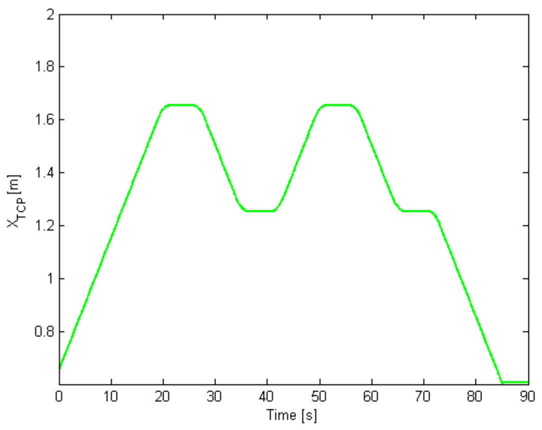


Figure 3.3: *X position of the tool point with respect of time.*

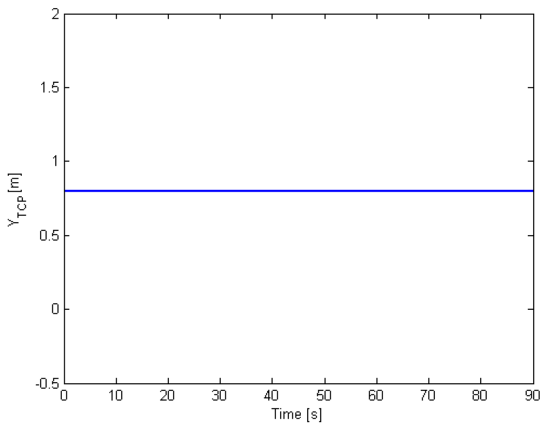


Figure 3.4: *Y position of the tool point with respect of time.*

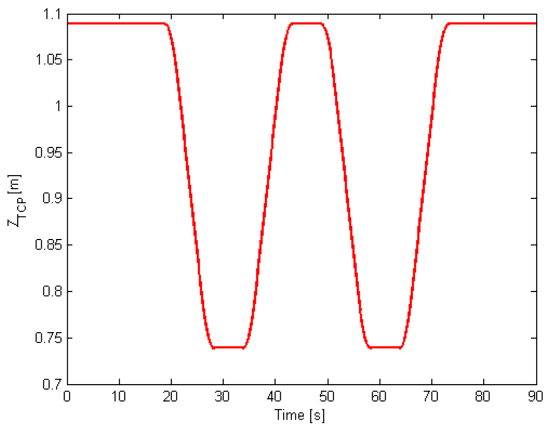


Figure 3.5: *Z position of the tool point with respect of time.*

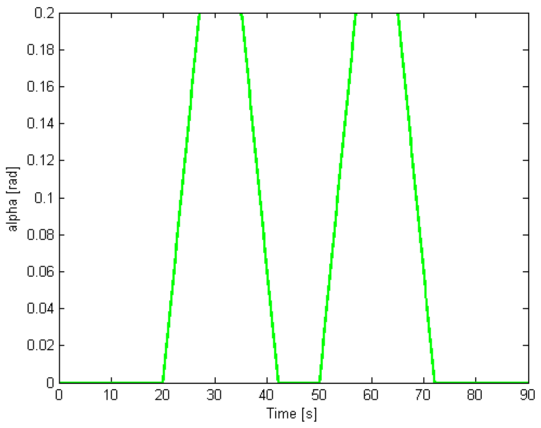


Figure 3.6: *Alpha angle with respect of time.*

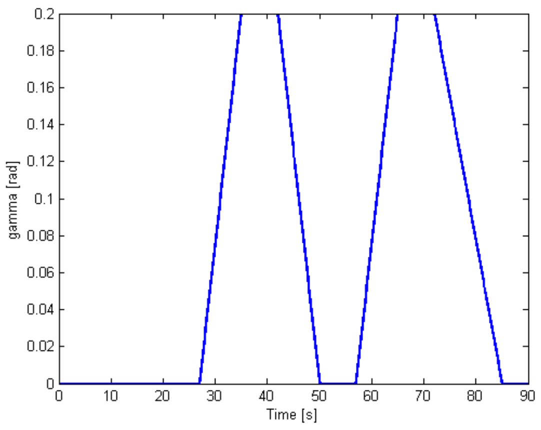


Figure 3.7: *Gamma angle with respect of time.*

Chapter 4

Control of the Gantry-Tau

As explained in chapter 2, the inverse kinematics allows to calculate the actuators position for a desired path of the tool point, but since the system in not ideal, perturbations affect the response of the robot leading to errors. Then, we need to apply control techniques to minimize the error as much as possible.

In this chapter we will develop a simulation that gradually describe the controller used for the Gantry-Tau. For this, we build up an extension to Simulink of the Matlab program, including the functions obtained for the IK, the IVK, the path planning, and now the controller for all the actuators of the robot. Also, an important remark to say, is that the simulation is cooperative with a feature call Co-Simulation from SimulationX, that allows to communicate with other simulations programs like Simulink, and in this way, simulate the system in SimulationX which will be controlled from Simulink. This feature greatly expands the possibilities for an improved control study since Simulink has specialized tools for this.

All the actuators of the Gantry-Tau are driven by servomotors with an internal velocity control. Meaning that, we can design a simple position-velocity controller for each motor, taking set-points from the outputs of the robot's main program. Then, the controller structure will be as shown in Figure 4.1

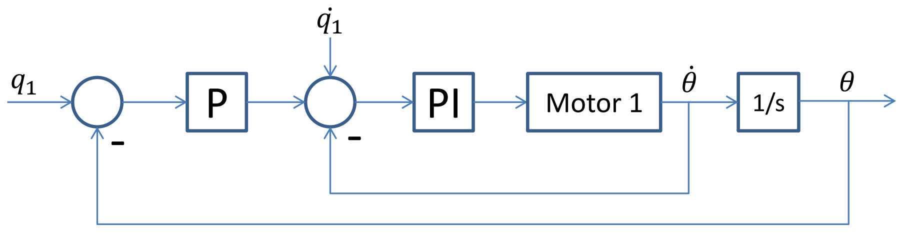


Figure 4.1: *Controller of the Gantry-Tau for the servomotor of actuator q_1.*

4.1 Co-Simulation

Now, we introduce to the Co-Simulation between Simulink and SimulationX, where their models are shown in Appendices E and F respectively. Since SimulationX does not have the possibility of creating a program code like in Matlab/Simulink, we make use of its principal feature as the modelling of multi-body systems, and leave Simulink all the control part.

The actuators of the Gantry-Tau are driven by servomotors and a gearing relation in between. This gearing relation is given by a belt drive and a ball screw drive. These two are shown in Figure 4.2. However, these elements had to be modelled for the system of the Gantry-Tau using their main characteristics as data to input the program. Then SimulationX is capable of representing these elements as the real ones. Of course, one might extend or refine the model by adding friction for example. An advantage of the Co-Simulation is a safe way of testing the robot's program without taking the risk on the real prototype, and allows more access to other results, providing a better interpretation of the program and system behaviour.

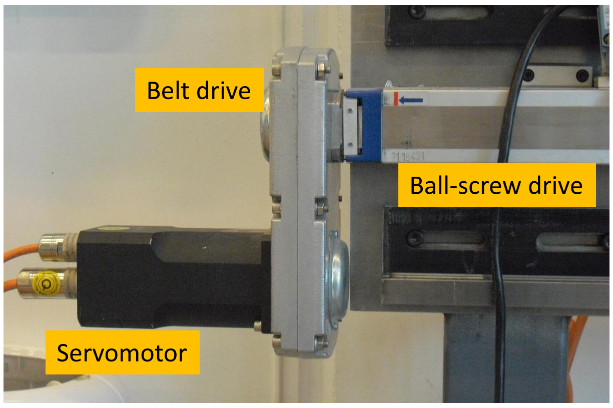


Figure 4.2: *Gearing found between the base actuator and servomotor.*

A long path for the tool point of the robot was created, very similar to the one shown in the last chapter. This path starts the tool point from the home position of the robot and afterwards it draws several square with round corners. Also it is present changes of speed and zone for the path trajectory. The computation part of the Simulink program is to calculate the appropriate displacements, velocity and accelerations for the robot, and the the Co-Simulation feature loads this data into the SimulationX environment, updating the kinematic and kinetic results. Also, a feedback of position of each actuator occurs, this feedback is needed to compensate for the error the tool point has with respect to the actual path.

The Co-Simulation result is shown in Figure 4.3, showing the path of tool point on the XZ plane while it was drawing a trajectory of several squares.

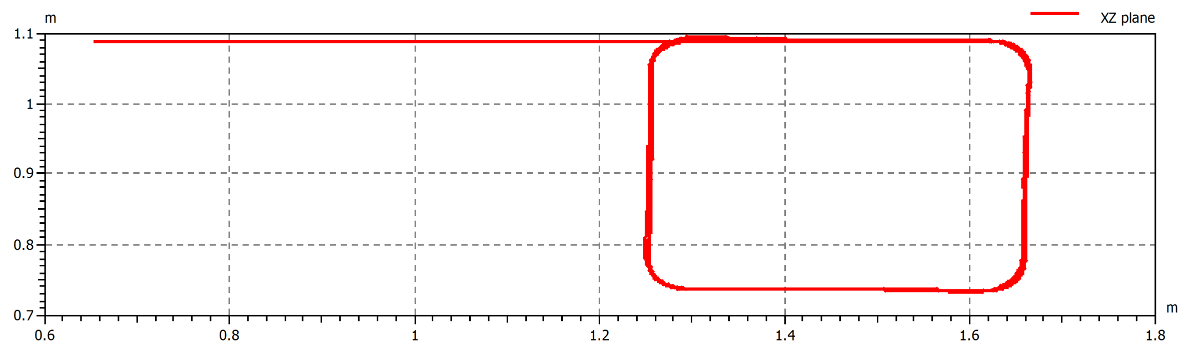


Figure 4.3: *Co-simulation result of the tool point position.*

4.2 Experiment with the prototype

The prototype of the Gantry-Tau uses a xPC target as the control system, where we upload the program to test it. The xPC target is a feature of Simulink that allows rapid prototyping, using a lightweight operative system in a target computer capable of running different types of simulation models in real-time. Figure 4.4, shows the actual configuration of the Gantry-Tau xPC target and control system.

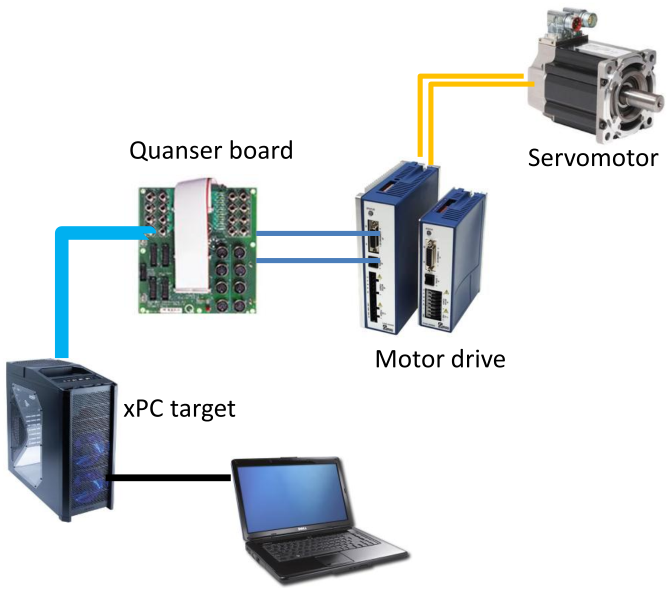


Figure 4.4: *Configuration of the xPC target with the actuators drives.*

By looking again at Figure 4.4, we see that the xPC target and the motor drives communicate with a Quanser board. This board is a Hardware-in-the-Loop control board for interfacing with different types of devices like sensors and encoders, and also is capable of writing digital and analog outputs, as well reading from inputs. This card interacts via a data bus with the xPC target and must be configured in the Simulink environment.

Appendix G shows the robot's main program in Simulink, including the path planning, the controllers for each actuator, and the mentioned configuration for interaction with the Quanser board.

Since, the actuators of the robot have encoders, we are able to know their position while the prototype is being tested. But knowing their position won't allow us to compute the position of the tool point since no formulation for the FK is made. For this, the university (UiA) holding the prototype of the Gantry-Tau offered the use of their high precision calibration machine also known as the FARO Laser Tracker, as shown in Figure 4.5.

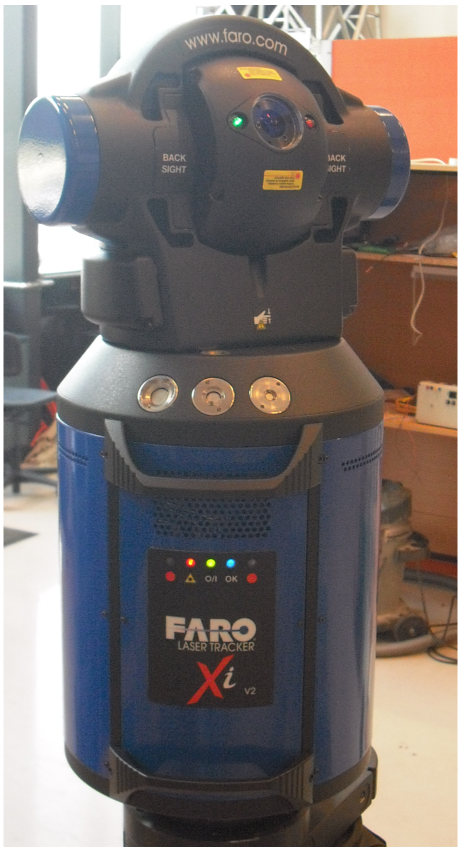


Figure 4.5: *Configuration of the xPC target with the actuators drives.*

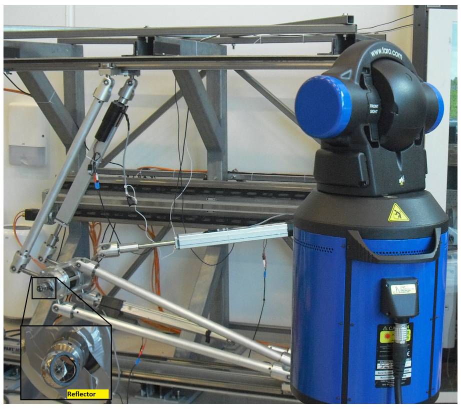


Figure 4.6: *Configuration of the xPC target with the actuators drives.*

The Laser Tracker is able to measure the absolute distance with the use of a reflector (Figure 4.6).

This reflector is to be mounted on the tool point, and in that way we can measure its position while the prototype is working.

Figures 4.7 and 4.8 show the position of the tool point in the planes XZ and YZ respectively while the prototype was doing the path used in the previous section. It is important to say that, the measurement is not similar and not calibrated due to the presence of many offsets. Also, the tracks of the prototype among many other parts are not well aligned, leading to many measurement errors.

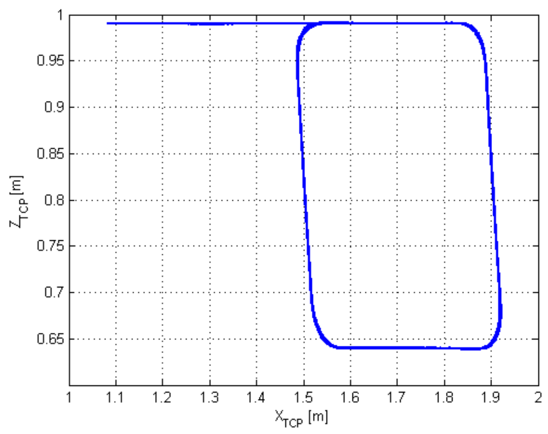


Figure 4.7: *Measurement of the prototype's tool point with the laser tracker (XZ plane).*

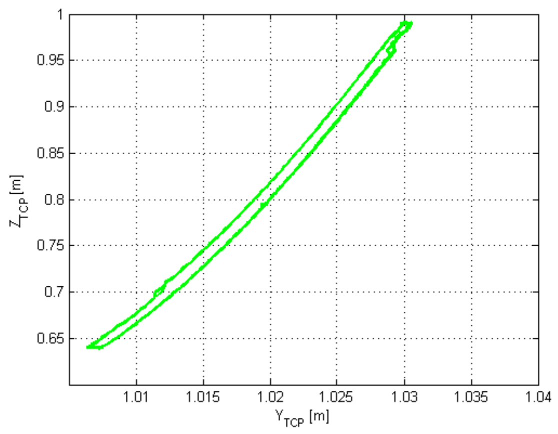


Figure 4.8: *Measurement of the prototype's tool point with the laser tracker (YZ plane).*

Chapter 5

Conclusion

The Gantry-Tau is a system that comprises many difficulties along the way. However, it offers a great deal of knowledge for industrial robotics for its complex and yet singular nature as a parallel robot. The work in this thesis result in revisiting many modelling parts of the Gantry-Tau, having at the same time a slightly different approach aimed to verification methods, and the introduction of alternative ways for getting a general perception of this amazing system.

From all the content developed in this work, a more detail explanation of the kinematics was made for the model-based system of the robot, still leaving place to adventure in other methods and modelling techniques. The idea of elevating more the prototyping and testing of the Gantry-Tau was made by the inclusion of a flexible path planner, justifying that, a deeper study must be made in the control study of this machine.

The verifications made using SimulationX demonstrate the elevated simulation capacity many software solutions offer without inquiring into a slow pace learning. This not only eases the investigation and work, but also encourages other disciplines to give contributions in pro of the science.

The experiments made with the prototype and the obtained results explain the accuracy of the modelling and control development, even when the real system is attached to many irregularities in its structure.

5.1 Suggested Improvements and Future Directions

The work and results obtained in this thesis represent only a small part of all the topics developed for the Gantry-Tau. Since the contents here are based on basic stages of the systems, a number of suggestions is offered for improving the scope of this work:

- Generalize the inverse kinematics modelling.
- Work out the inverse velocity kinematics for all the links.

- Develop the inverse dynamics and verify it with the SimulationX model.
- From the inverse dynamics develop a model-based dynamic controller.
- Extend the path planning program to have other types of trajectories and elevate the control testing.

Also, many potential opportunities for progress were discovered along the development of this thesis wich could be considered as future directions for the Gantry-Tau and therefore the general modelling and identification of parallel robots:

- SimulationX uses the Newton-Euler approach to model the dynamics of any multi-body system. It's convenient to model through this technique since the Lagrange method is too systematic and generalized for obtaining accurate results.
- SimulationX is capable of generating numerical and analytical Jacobian with respect to almost any part of the system. This could be an advantage when verifying the elements of the obtained Jacobians.
- SimulationX is capable of offering an inverse kinematic approach to the system, but also a forward kinematic approach without conditioning or condtraining any parts of the model. This could be studied for generating the forward kinematic equations and ease the way of obtaining Jacobians for velocities and accelerations of actuators, pasive joints, etc.

Bibliography

[1] Luigi Villani Bruno Siciliano, Lorenzo Sciavicco and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control.* Springer, 2009.

[2] G. Gogu. *Structural Synthesis of Parallel Robots.* Springer, 2008.

[3] R. N. Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control.* Springer, 2nd edition, 2010.

[4] W. Khalil and E. Dombre. *Introduction to geometric and kinematic modeling of parallel robots.* Kogan Page Science, 2004.

[5] C. Lyzell. Modeling and identification of the gantry-tau parallel kinematic machine. Master's thesis, Linköping University.

[6] Geir Hovland Matthew Murray and Torgny Brogårdh. Collision-free workspace design of the 5-axis gantry-tau parallel kinematic machine. *International Conference on Intelligent Robots and Systems.*

[7] J.-P. Merlet. *Parallel Robots.* Springer, 2nd edition, 2006.

[8] B. Siciliano and O. Khatib. *Springer Handbook of Robotics.* Springer, 2008.

[9] Hutchinson S. Spong, M. W. and M. Vidyasagar. *Robot Modeling and Control*, pages 54–54. John Wiley and Sons., 2005.

[10] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley and Sons., 2004.

[11] D. Zhang. *Parallel Robotic Machine tools*, pages 38–39. Springer, 2010.

Appendix A

Coefficients of W

> # Analytical Inverse Kinematics (IK) for the Gantry-Tau 5 DOF

> # This program computes analytical coefficients of each root of W

>

> *restart* :

> *with*(*LinearAlgebra*) :

> *with*(*CodeGeneration*) :

> # Define rotation matrices with respect of t:

> $Rx := \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) \\ 0 & \sin(a) & \cos(a) \end{bmatrix}$:

> $Ry := \begin{bmatrix} \cos(b) & 0 & \sin(b) \\ 0 & 1 & 0 \\ -\sin(b) & 0 & \cos(b) \end{bmatrix}$:

> $Rz := \begin{bmatrix} \cos(g) & -\sin(g) & 0 \\ \sin(g) & \cos(g) & 0 \\ 0 & 0 & 1 \end{bmatrix}$:

> # Find the R matrix with the following rotation order:

> $R := Rx.Rz.Ry$:

>

> # *We give values to the offsets and links lengths, later this will greatly reduce the size of analytical equations:*

> $Dpoffset := \begin{bmatrix} 0.1126 \\ 0 \\ -0.065 \end{bmatrix}$:

> $Epoffset := \begin{bmatrix} -0.1126 \\ -0.125 \\ -0.065 \end{bmatrix}$:

> $Dboffset := \begin{bmatrix} 0 \\ 0.125 \\ 0.188 \end{bmatrix}$:

> $Eboffset := \begin{bmatrix} 0 \\ 0 \\ 0.188 \end{bmatrix}$:

> $L31 := 1.25$:

> $L32 := 1.25$:

>

> # *Define platform joints Dp and Ep, and later assign them separately:*

> $Dp := R.\begin{bmatrix} Dpoffset(1) \\ Dpoffset(2) \\ Dpoffset(3) \end{bmatrix} + \begin{bmatrix} XTCP \\ YTCP \\ ZTCP \end{bmatrix}:$

> $Ep := R.\begin{bmatrix} Epoffset(1) \\ Epoffset(2) \\ Epoffset(3) \end{bmatrix} + \begin{bmatrix} XTCP \\ YTCP \\ ZTCP \end{bmatrix}:$

> $X_{Dp} := Dp(1):$

> $Y_{Dp} := Dp(2):$

> $Z_{Dp} := Dp(3):$

> $X_{Ep} := Ep(1):$

> $Y_{Ep} := Ep(2):$

> $Z_{Ep} := Ep(3):$

> $L_{31} := \mathrm{L31}:$

> $L_{32} := L32:$

>

> # *Assign offsets of base joints Db and Eb:*

> $Y_{Db} := Dboffset(2):$

> $Z_{Db} := Dboffset(3):$

> $Y_{Eb} := Eboffset(2):$

> $Z_{Eb} := Eboffset(3):$

>

> # *Create equations from the IK modelling for 5DOF:*

> $U := (Y_{Dp} - Y_{Db})^2 + (Z_{Dp} - Z_{Db})^2:$

> $V := (Y_{Ep} - Y_{Eb})^2 + (Z_{Ep} - Z_{Eb})^2:$

> $eq1 := \left(X_{Dp} - \frac{L_{31}^2 - L_{32}^2 - X_{Dp}^2 + X_{Ep}^2 - U + V}{2\,(X_{Ep} - X_{Dp})}\right)^2 + U - L_{31}^2 = 0:$

>

> # *Relations of sin(b) and cos(b) with the Weierstrass substitution:*

> $eq2 := \sin(b) = \frac{2\,W}{1 + W^2}:$

> $eq3 := \cos(b) = \frac{1 - W^2}{1 + W^2}:$

>

> # *Substitute the past relations in the main equation:*

> $eq4 := simplify(subs(eq2, eq3, eq1)):$

>

> # *Since the equation equals 0, then we take only the numerator:*

```
> eq5 := numer(lhs(eq4)) :
>
> # Sort W in descending order:
> eq6 := collect(eq5, W) :
> eq7 := sort(eq6, W, descending) :
>
> # Take each coefficient of the 9 roots of W:
> Root8 := simplify(coeff(eq7, W, 8)) :
> Root7 := simplify(coeff(eq7, W, 7)) :
> Root6 := simplify(coeff(eq7, W, 6)) :
> Root5 := simplify(coeff(eq7, W, 5)) :
> Root4 := simplify(coeff(eq7, W, 4)) :
> Root3 := simplify(coeff(eq7, W, 3)) :
> Root2 := simplify(coeff(eq7, W, 2)) :
> Root1 := simplify(coeff(eq7, W, 1)) :
> Root0 := simplify(coeff(eq7, W, 0)) :
>
> ROOTS := Vector(9, [Root8, Root7, Root6, Root5, Root4, Root3, Root2, Root1, Root0]) :
>
> # Generate the Matlab code:
> Matlab(ROOTS, resultname = "Roots", output = "Roots_W_5DOF.m", optimize) :
>
```

Appendix B

Inverse Jacobian Matrix

```
> # Analytical Inverse Velocity Kinematics (IVK) for the Gantry-Tau 5DOF
> # This program computes the analytical Inverse Jacobian Matrix (5x6)
>
> restart :
> with( LinearAlgebra ) :
> with( CodeGeneration ) :
>
> # Define rotation matrices with respect of t:
```

$$> Rx := \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(a(t)) & -\sin(a(t)) \\ 0 & \sin(a(t)) & \cos(a(t)) \end{bmatrix} :$$

$$> Ry := \begin{bmatrix} \cos(b(t)) & 0 & \sin(b(t)) \\ 0 & 1 & 0 \\ -\sin(b(t)) & 0 & \cos(b(t)) \end{bmatrix} :$$

$$> Rz := \begin{bmatrix} \cos(g(t)) & -\sin(g(t)) & 0 \\ \sin(g(t)) & \cos(g(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix} :$$

```
>
> # Find the R matrix with the following rotation order:
> R := Rx Rz Ry :
>
> # General platform joints calculation from the Inverse Kinematics, where the TCP coordinates
      are also with respect of t:
```

$$> Kp := R \begin{bmatrix} Kpoffset_1 \\ Kpoffset_2 \\ Kpoffset_3 \end{bmatrix} + \begin{bmatrix} XTCP(t) \\ YTCP(t) \\ ZTCP(t) \end{bmatrix} :$$

> # General equation for q_1, q_2 and q_3, later the letter K will be replaced by A, B and D. Also, sol = −1 for right solution and sol = 1 for left solution :

$$> q123 := Kp(1) + sol\sqrt{L^2 - \left(Kp(2) - Kboffset_2\right)^2 - \left(Kp(3) - Kboffset_3\right)^2} :$$

```
>
> # Declare function for the equation:
> q123f := unapply( q123, t ) :
>
> # Now differentiate:
> q123dot := D( q123f ) :
>
> # Convert back to an equation:
```

```
> q123dot := apply(q123dot, t) :
>
> # Substitute names of the differential variables with the names used in the Matlab program:
> q123dot := subs({D(XTCP)(t) = XTCPdot, D(YTCP)(t) = YTCPdot, D(ZTCP)(t) = ZTCPdot,
      D(a)(t) = adot, D(b)(t) = bdot, D(g)(t) = gdot}, q123dot) :
> # then substitute the variables with respect of t with the ones used in the Matlab program:
> q123dot := subs({XTCP(t) = XTCP, YTCP(t) = YTCP, ZTCP(t) = ZTCP, a(t) = a, b(t) = b, g(t)
      = g}, q123dot) :
>
> # Create q1dot and denq1 by substituting K for A and L for L1:
> q1dot := subs({Kboffset_1 = Aboffset_1, Kboffset_2 = Aboffset_2, Kboffset_3 = Aboffset_3, Kpoffset_1
      = Apoffset_1, Kpoffset_2 = Apoffset_2, Kpoffset_3 = Apoffset_3, L = 1.25, sol = sol1}, q123dot) :
>
> # Create q2dot and denq2 by substituting K for B and L for L21:
> q2dot := subs({Kboffset_1 = Bboffset_1, Kboffset_2 = Bboffset_2, Kboffset_3 = Bboffset_3, Kpoffset_1
      = Bpoffset_1, Kpoffset_2 = Bpoffset_2, Kpoffset_3 = Bpoffset_3, L = 1.09, sol = sol2}, q123dot) :
>
> # Create q3dot and denq3 by substituting K for D and L for L31:
> q3dot := subs({Kboffset_1 = Dboffset_1, Kboffset_2 = Dboffset_2, Kboffset_3 = Dboffset_3, Kpoffset_1
      = Dpoffset_1, Kpoffset_2 = Dpoffset_2, Kpoffset_3 = Dpoffset_3, L = 1.25, sol = sol3}, q123dot) :
>
> # Extract each differentiated coefficient to generate equations of each element of the Inverse
      Jacobian matrix (rows 1 to 3):
> j11 := simplify(coeff(q1dot, XTCPdot), size) :
> j12 := simplify(coeff(q1dot, YTCPdot), size) :
> j13 := simplify(coeff(q1dot, ZTCPdot), size) :
> j14 := simplify(coeff(q1dot, adot), size) :
> j15 := simplify(coeff(q1dot, bdot), size) :
> j16 := simplify(coeff(q1dot, gdot), size) :
>
> j21 := simplify(coeff(q2dot, XTCPdot), size) :
> j22 := simplify(coeff(q2dot, YTCPdot), size) :
> j23 := simplify(coeff(q2dot, ZTCPdot), size) :
> j24 := simplify(coeff(q2dot, adot), size) :
> j25 := simplify(coeff(q2dot, bdot), size) :
> j26 := simplify(coeff(q2dot, gdot), size) :
>
> j31 := simplify(coeff(q3dot, XTCPdot), size) :
> j32 := simplify(coeff(q3dot, YTCPdot), size) :
> j33 := simplify(coeff(q3dot, ZTCPdot), size) :
> j34 := simplify(coeff(q3dot, adot), size) :
> j35 := simplify(coeff(q3dot, bdot), size) :
> j36 := simplify(coeff(q3dot, gdot), size) :
```

```
>
> # Now we work for q_4 and q_5 :
> # General equation for q_4 and q_5, later the letter K will be replaced by C and F :
```

$$> q45 := \sqrt{(Kp(1) - qn)^2 + (Kp(2) - Kboffset_2)^2 + (Kp(3) - Kboffset_3)^2} :$$

```
>
> # Create equations q2 and q3 that will be substituted later by qn to create q4 and q5
      respectively:
> q2 := subs( {Kboffset_1 = Bboffset_1, Kboffset_2 = Bboffset_2, Kboffset_3 = Bboffset_3, Kpoffset_1
       = Bpoffset_1, Kpoffset_2 = Bpoffset_2, Kpoffset_3 = Bpoffset_3, L = 1.09, sol = sol2}, q123) :
> q3 := subs( {Kboffset_1 = Dboffset_1, Kboffset_2 = Dboffset_2, Kboffset_3 = Dboffset_3, Kpoffset_1
       = Dpoffset_1, Kpoffset_2 = Dpoffset_2, Kpoffset_3 = Dpoffset_3, L = 1.25, sol = sol3}, q123) :
>
> q4 := subs( {qn = q2, Kpoffset_1 = Cpoffset_1, Kpoffset_2 = Cpoffset_2, Kpoffset_3 = Cpoffset_3,
      Kboffset_2 = Cboffset_2, Kboffset_3 = Cboffset_3}, q45) :
> q5 := subs( {qn = q3, Kpoffset_1 = Fpoffset_1, Kpoffset_2 = Fpoffset_2, Kpoffset_3 = Fpoffset_3,
      Kboffset_2 = Fboffset_2, Kboffset_3 = Fboffset_3}, q45) :
>
> # Declare functions for the equations and differentiate them:
> q4f := unapply(q4, t) :
> q5f := unapply(q5, t) :
> q4dot := D(q4f) :
> q5dot := D(q5f) :
>
> # Convert back to equations and subtitute variables and diff. variables to the ones used in
      Matlab:
> q4dot := apply(q4dot, t) :
> q5dot := apply(q5dot, t) :
>
> q4dot := subs( {D(XTCP)(t) = XTCPdot, D(YTCP)(t) = YTCPdot, D(ZTCP)(t) = ZTCPdot,
      D(a)(t) = adot, D(b)(t) = bdot, D(g)(t) = gdot}, q4dot) :
> q4dot := subs( {XTCP(t) = XTCP, YTCP(t) = YTCP, ZTCP(t) = ZTCP, a(t) = a, b(t) = b, g(t)
       = g}, q4dot) :
>
> q5dot := subs( {D(XTCP)(t) = XTCPdot, D(YTCP)(t) = YTCPdot, D(ZTCP)(t) = ZTCPdot,
      D(a)(t) = adot, D(b)(t) = bdot, D(g)(t) = gdot}, q5dot) :
> q5dot := subs( {XTCP(t) = XTCP, YTCP(t) = YTCP, ZTCP(t) = ZTCP, a(t) = a, b(t) = b, g(t)
       = g}, q5dot) :
>
> # Extract each differentiated coefficient to generate equations of each element of the Inverse
      Jacobian matrix (rows 4 and 5):
> j41 := simplify(coeff(q4dot, XTCPdot), size) :
> j42 := simplify(coeff(q4dot, YTCPdot), size) :
```

```
> j43 := simplify( coeff( q4dot, ZTCPdot), size) :
> j44 := simplify( coeff( q4dot, adot), size) :
> j45 := simplify( coeff( q4dot, bdot), size) :
> j46 := simplify( coeff( q4dot, gdot), size) :
>
> j51 := simplify( coeff( q5dot, XTCPdot), size) :
> j52 := simplify( coeff( q5dot, YTCPdot), size) :
> j53 := simplify( coeff( q5dot, ZTCPdot), size) :
> j54 := simplify( coeff( q5dot, adot), size) :
> j55 := simplify( coeff( q5dot, bdot), size) :
> j56 := simplify( coeff( q5dot, gdot), size) :
>
> # Generate the Inverse Jacobian matrix:
> IJ := Matrix( 5, 6, [ [ j11, j12, j13, j14, j15, j16 ], [ j21, j22, j23, j24, j25, j26 ], [ j31, j32, j33, j34,
      j35, j36 ], [ j41, j42, j43, j44, j45, j46 ], [ j51, j52, j53, j54, j55, j56 ] ]) :
>
> # Generate the Matlab code:
> Matlab( IJ, resultname = "ij", output = "InvJacobianRobot_5DOF.m", optimize) :
>
```

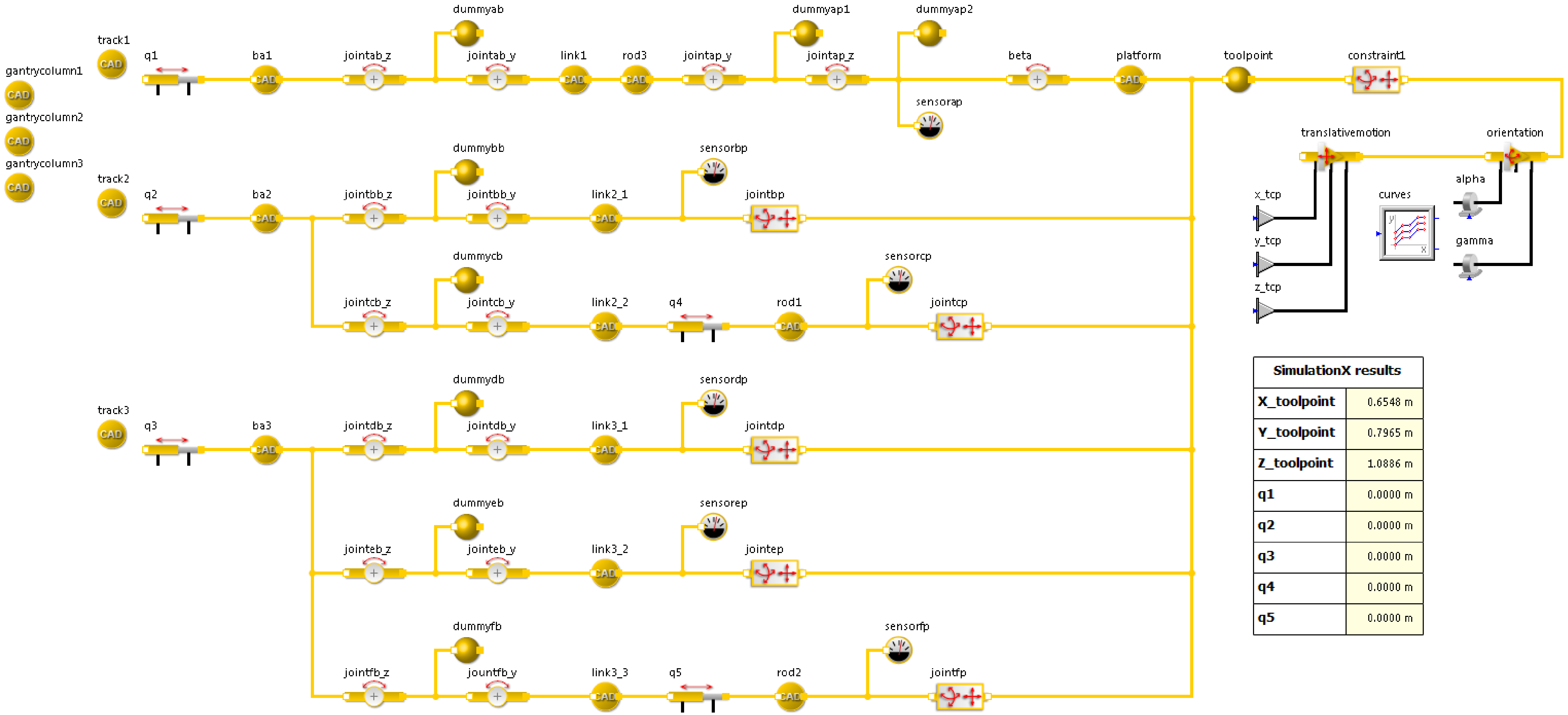
Appendix C

Path Planning program

```matlab
function [pathTrans,pathRot,totalTime] = fcn(time)
%#eml

% This program generates the values for path planning in real time (or
% simulation time). One must input the number of points for the path and
% the number of loops the path in going to make. Later in the area of Path
% Points the user must specify each point in space and its orientation,
% velocity and zone.


Tstep = 0.01;     % IMPORTANT, must be equal to the simulation Time-Step

n = 10;         % Number of points for the path
pathLoops = 1;         % Number of path Path loops (>=1)


% ---------------------------------------------------------------
% Preassign variables
% ---------------------------------------------------------------
% It's good to give an starting value to variables that will be used in
% operations like if-statements and for-loops AND will be used as outputs:
pathTrans = zeros(1,3);     % Output vector for X_TCP, Y_TCP and Z_TCP
pathRot = zeros(1,2);       % Output vector for alpha and gamma
moveL = zeros(n,7);         % Path matrix

if pathLoops >= 2
    t = n*(pathLoops);
    X = zeros(t+1,1);
    Y = zeros(t+1,1);
    Z = zeros(t+1,1);
    alpha = zeros(t+1,1);
    gamma = zeros(t+1,1);
    vel = zeros(t+1,1);
    zone = zeros(t+1,1);
    Tpoint = zeros(t+1,1);
    Tzone = zeros(t+1,1);
else
    X = zeros(n,1);
    Y = zeros(n,1);
    Z = zeros(n,1);
    alpha = zeros(n,1);
    gamma = zeros(n,1);
    vel = zeros(n,1);
    zone = zeros(n,1);
    Tpoint = zeros(n,1);
    Tzone = zeros(n,1);
    t = n-1;
end
% ---------------------------------------------------------------
% ---------------------------------------------------------------



% ---------------------------------------------------------------
% Path points
% ---------------------------------------------------------------
% Example: moveL(1,:) = [X_TCP Y_TCP Z_TCP alpha gamma vel zone]
% X_TCP, Y_TCP and Z_TCP in meters
% alpha and gamma in radians
% vel in mm/s
% zone is fillet radius in mm for smoothing corners
% For one path loop, in the first moveL the vel and zone are 0
% For one path loop, in the last moveL the zone is 0
% For 2 or more path loops the first and last moveL must have values in vel
% and zone


% Beginning of SHORT PATH 5 DoF
% n=10, pathLoops=1, time for simulation=90 seconds
% Make sure the Tool-Point starts in home.
moveL(1,:) = [0.6542 0.7964 1.0886 0 0 0 0];

moveL(2,:) = [1.6542 0.7964 1.0886 0 0 50 60];
moveL(3,:) = [1.6542 0.7964 0.7386 0.2 0 50 60];
moveL(4,:) = [1.2542 0.7964 0.7386 0.2 0.2 50 60];
moveL(5,:) = [1.2542 0.7964 1.0886 0 0.2 50 60];

moveL(6,:) = [1.6542 0.7964 1.0886 0 0 50 60];
moveL(7,:) = [1.6542 0.7964 0.7386 0.2 0 50 60];
moveL(8,:) = [1.2542 0.7964 0.7386 0.2 0.2 50 60];
moveL(9,:) = [1.2542 0.7964 1.0886 0 0.2 50 60];

moveL(10,:) = [0.6042 0.7964 1.0886 0 0 50 0];
% End of SHORT PATH 5 DoF

% ---------------------------------------------------------------
% ---------------------------------------------------------------



% ---------------------------------------------------------------
% Vector assignments for each variable
% ---------------------------------------------------------------
aux = 0;
if pathLoops >= 2
    for rep = 1:pathLoops
        X(rep+aux:rep*n,1) = moveL(:,1);
        Y(rep+aux:rep*n,1) = moveL(:,2);
        Z(rep+aux:rep*n,1) = moveL(:,3);
        alpha(rep+aux:rep*n,1) = moveL(:,4);
        gamma(rep+aux:rep*n,1) = moveL(:,5);
        vel(rep+aux:rep*n,1) = moveL(:,6);
        zone(rep+aux:rep*n,1) = moveL(:,7);
        if rep == pathLoops
            X(t+1,1) = moveL(1,1);
            Y(t+1,1) = moveL(1,2);
            Z(t+1,1) = moveL(1,3);
            alpha(t+1,1) = moveL(1,4);
            gamma(t+1,1) = moveL(1,5);
            vel(t+1,1) = moveL(1,6);
            zone(t+1,1) = moveL(1,7);
        end
        aux = aux+n-1;
    end
else
    X = moveL(:,1);
    Y = moveL(:,2);
    Z = moveL(:,3);
    alpha = moveL(:,4);
    gamma = moveL(:,5);
    vel = moveL(:,6);
    zone = moveL(:,7);
end
% -------------------------------------------------------------
% -------------------------------------------------------------



% -------------------------------------------------------------
% Calculate the times at each point and for each zone
% -------------------------------------------------------------
for m = 1:t
    L = sqrt((X(m+1)-X(m))^2 + (Y(m+1)-Y(m))^2 + (Z(m+1)-Z(m))^2);
    if m == 1
        Tpoint(m)  = 0;
        Tpoint(m+1)  = L/(vel(m+1)*0.001);
        Tzone(m)  = 0;
        Tzone(m+1)  = zone(m+1)/vel(m+1);
    end
    if (m > 1) && (m < t)
        Tpoint(m+1)  = Tpoint(m)  + L/(vel(m+1)*0.001);
        Tzone(m+1)  = zone(m+1)/vel(m+1);
    end
    if m == t
        Tpoint(m+1)  = Tpoint(m)  + L/(vel(m+1)*0.001);
        Tzone(m+1)  = 0;
    end
end
% -------------------------------------------------------------
% -------------------------------------------------------------



% -------------------------------------------------------------
% Output of path
% -------------------------------------------------------------
% First build path matrix:
tcp = [X Y Z alpha gamma];

% Update for pathTrans (X_TCP, Y_TCP and Z_TCP)
for k = 1:t
    if time == 0
        pathTrans = tcp(1,1:3);
    end
    for t = time:(Tpoint(k+1) + Tzone(k+1) + Tstep)
        if (time > Tpoint(k) + Tzone(k)) && ...
                (time <= Tpoint(k+1) - Tzone(k+1) + Tstep)
            pathTrans = tcp(k+1,1:3) - (tcp(k+1,1:3) -
tcp(k,1:3))*(Tpoint(k+1) - time)/(Tpoint(k+1) - Tpoint(k));
        end
        if (time > Tpoint(k+1) - Tzone(k+1) + Tstep) && ...
                (time <= Tpoint(k+1) + Tzone(k+1)) && (k < t)
            pathTrans = tcp(k+1,1:3) - (tcp(k+1,1:3) - tcp(k,1:3))*((time -
Tzone(k+1) - Tpoint(k+1))^2)/(4*Tzone(k+1)*(Tpoint(k+1) - Tpoint(k)))...
                + (tcp(k+2,1:3) - tcp(k+1,1:3))*((time + Tzone(k+1) -
Tpoint(k+1))^2)/(4*Tzone(k+1)*(Tpoint(k+2) - Tpoint(k+1)));
        end
    end
end

% Update for pathRot (alpha and gamma)
for i = 1:t
    if time == 0
        pathRot = tcp(1,4:5);
    end
    for j = time:Tpoint(i+1)
        if (time > Tpoint(i)) && (time <= Tpoint(i+1))
            pathRot = (time - Tpoint(i))*(tcp(i+1,4:5) -
tcp(i,4:5))/(Tpoint(i+1) - Tpoint(i)) + tcp(i,4:5);
        end
    end
end

% Once the time reaches the last Tpoint, maintain the value
if time >= Tpoint(t+1,1)
    pathTrans = tcp(t+1,1:3);
    pathRot = tcp(t+1,4:5);
end
% ---------------------------------------------------------------
% ---------------------------------------------------------------

% ---------------------------------------------------------------
% Total time of the path
% ---------------------------------------------------------------
totalTime = Tpoint(t+1,1);
% ---------------------------------------------------------------
% ---------------------------------------------------------------
```

Appendix D

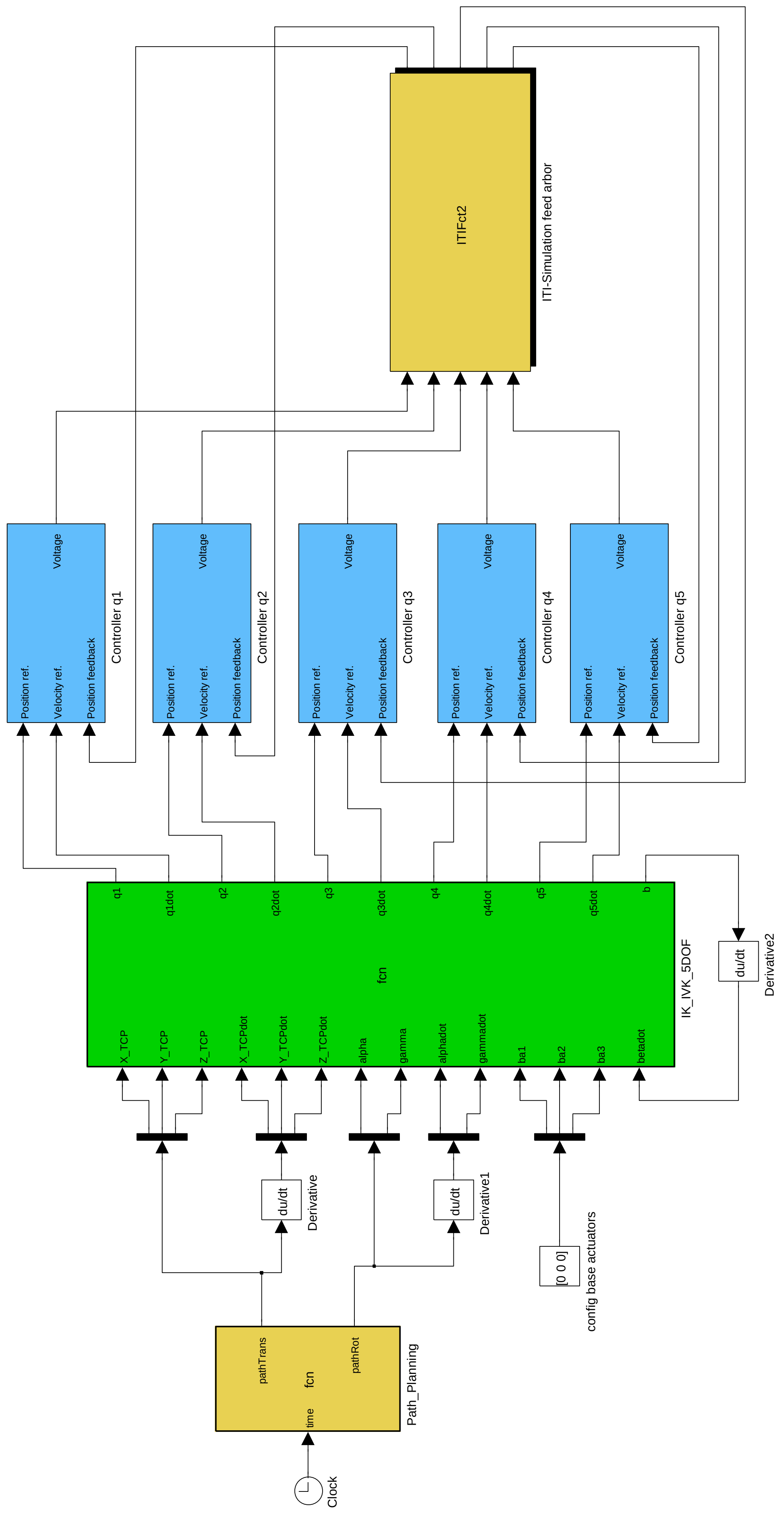
SimulationX IK model of the Gantry-Tau



SimulationX results	
X_toolpoint	0.6548 m
Y_toolpoint	0.7965 m
Z_toolpoint	1.0886 m
q1	0.0000 m
q2	0.0000 m
q3	0.0000 m
q4	0.0000 m
q5	0.0000 m

Appendix E

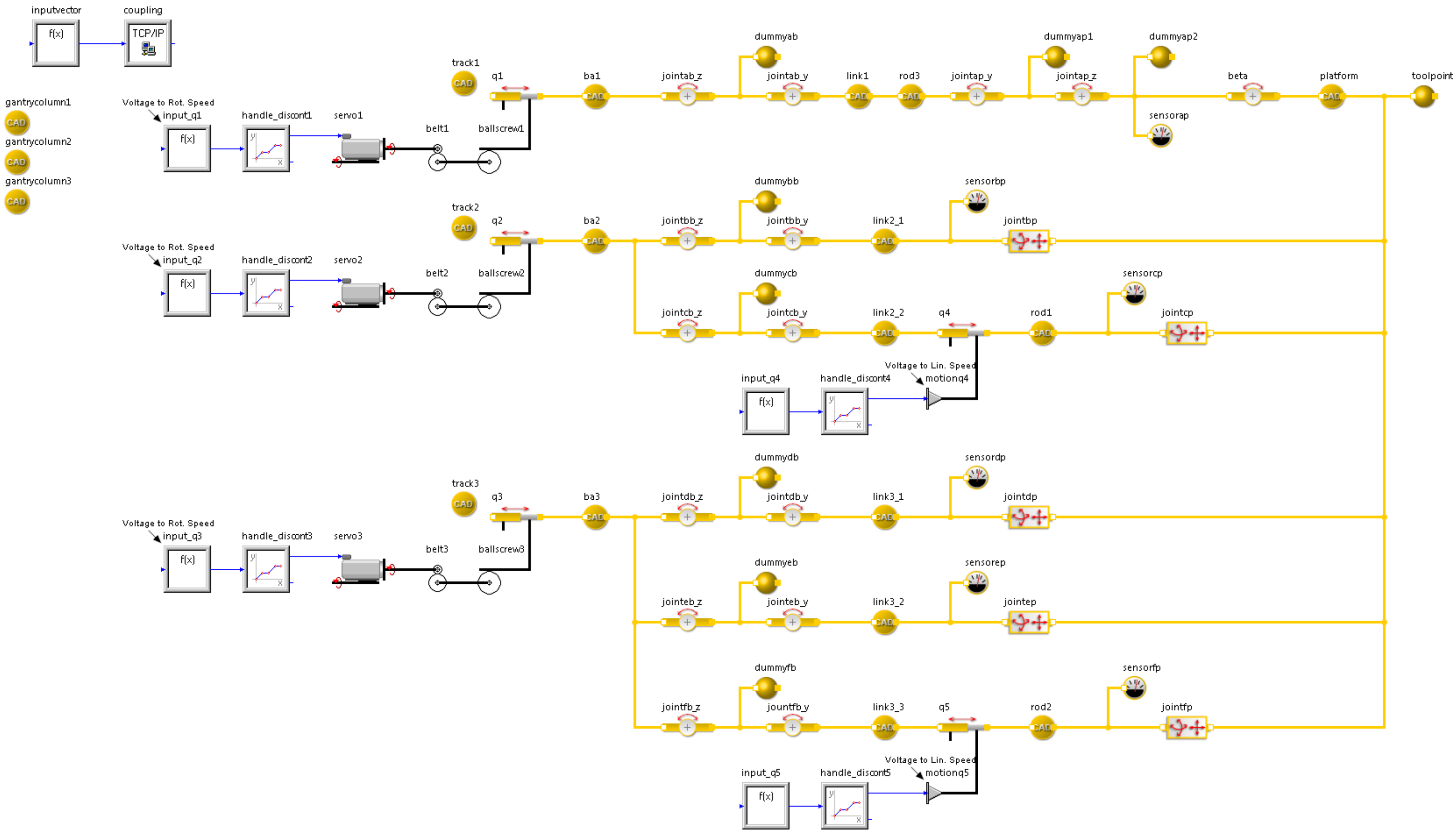
Simulink Model for Co-Simulation



Clock
time
fcn
pathTrans
pathRot
Path_Planning
du/dt
Derivative
du/dt
Derivative1
[0 0 0]
config base actuators
X_TCP
Y_TCP
Z_TCP
X_TCPdot
Y_TCPdot
Z_TCPdot
alpha
gamma
alphadot
gammadot
ba1
ba2
ba3
betadot
fcn
q1
q1dot
q2
q2dot
q3
q3dot
q4
q4dot
q5
q5dot
b
IK_IVK_5DOF
du/dt
Derivative2
Position ref.
Velocity ref.
Position feedback
Voltage
Controller q1
Position ref.
Velocity ref.
Position feedback
Voltage
Controller q2
Position ref.
Velocity ref.
Position feedback
Voltage
Controller q3
Position ref.
Velocity ref.
Position feedback
Voltage
Controller q4
Position ref.
Velocity ref.
Position feedback
Voltage
Controller q5
ITIFct2
ITI-Simulation feed arbor

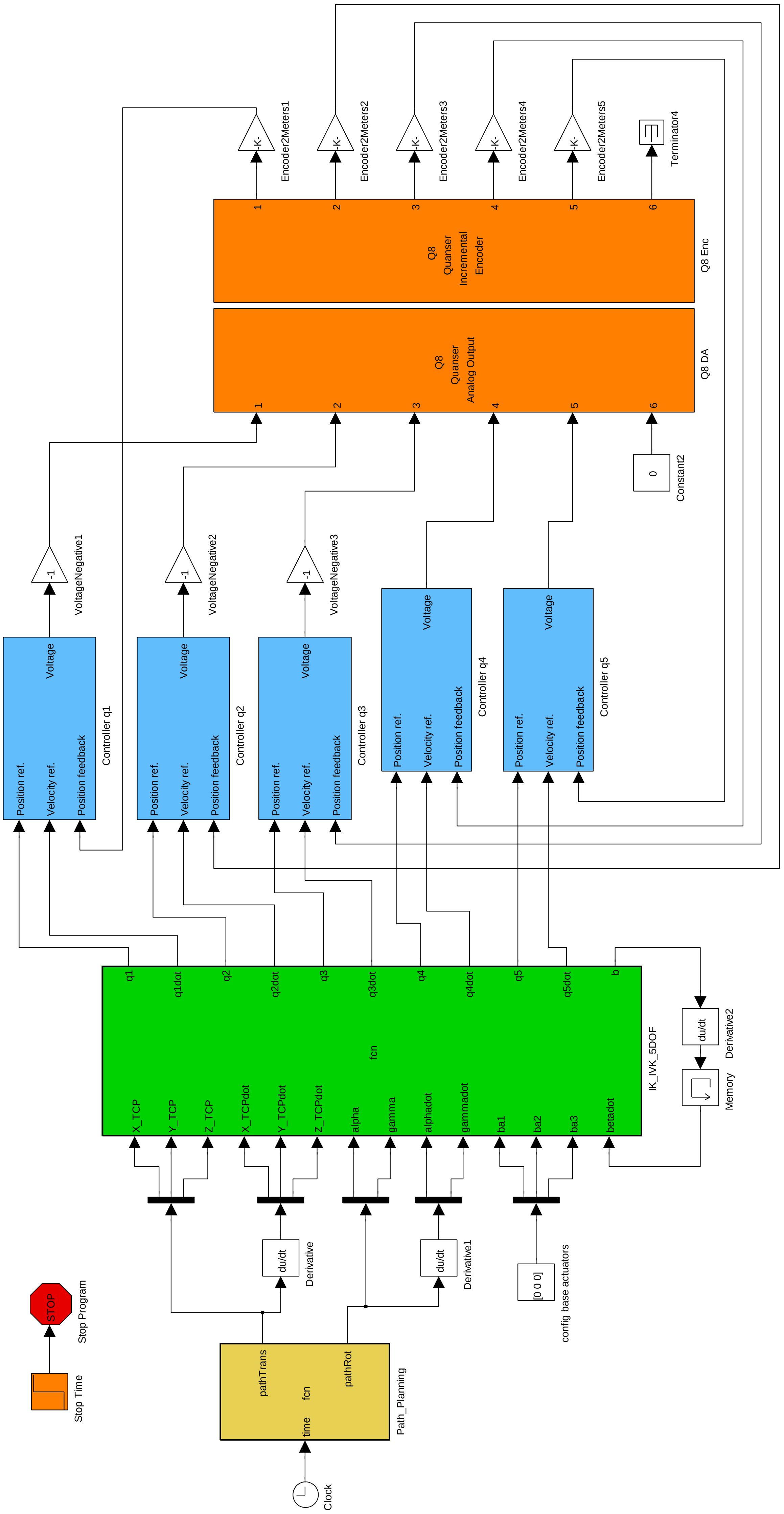
Appendix F

SimulationX FK model for Co-Simulation


inputvector
f(x)
coupling
TCP/IP
gantrycolumn1
gantrycolumn2
gantrycolumn3
CAD
Voltage to Rot. Speed
input_q1
handle_discont1
servo1
belt1
ballscrew1
track1
q1
ba1
jointab_z
dummyab
jointab_y
link1
rod3
jointap_y
dummyap1
jointap_z
dummyap2
sensorap
beta
platform
toolpoint
Voltage to Rot. Speed
input_q2
handle_discont2
servo2
belt2
ballscrew2
track2
q2
ba2
jointbb_z
dummybb
jointbb_y
link2_1
sensorbp
jointbp
jointcb_z
dummycb
jointcb_y
link2_2
q4
rod1
sensorcp
jointcp
input_q4
handle_discont4
Voltage to Lin. Speed
motionq4
Voltage to Rot. Speed
input_q3
handle_discont3
servo3
belt3
ballscrew3
track3
q3
ba3
jointdb_z
dummydb
jointdb_y
link3_1
sensordp
jointdp
jointeb_z
dummyeb
jointeb_y
link3_2
sensorep
jointep
jointfb_z
dummyfb
jountfb_y
link3_3
q5
rod2
sensorfp
jointfp
input_q5
handle_discont5
Voltage to Lin. Speed
motionq5

Appendix G

Robot's main program



Clock
time
fcn
pathTrans
pathRot
Path_Planning
Stop Time
STOP
Stop Program
du/dt
Derivative
du/dt
Derivative1
[0 0 0]
config base actuators
X_TCP
Y_TCP
Z_TCP
X_TCPdot
Y_TCPdot
Z_TCPdot
alpha
gamma
alphadot
gammadot
ba1
ba2
ba3
betadot
fcn
q1
q1dot
q2
q2dot
q3
q3dot
q4
q4dot
q5
q5dot
b
IK_IVK_5DOF
du/dt
Derivative2
Memory
Position ref.
Velocity ref.
Position feedback
Voltage
Controller q1
Controller q2
Controller q3
Controller q4
Controller q5
-1
VoltageNegative1
VoltageNegative2
VoltageNegative3
0
Constant2
Q8
Quanser
Analog Output
Q8 DA
Q8
Quanser
Incremental
Encoder
Q8 Enc
1
2
3
4
5
6
-K-
Encoder2Meters1
Encoder2Meters2
Encoder2Meters3
Encoder2Meters4
Encoder2Meters5
Terminator4