UNIVERSITY OF GENOVA

MASTER'S DEGREE IN ROBOTICS ENGINEERING

Multi-body Dynamic Simulation of a Parallel Robot: Development and Validation

by

Roberto Albanese

Thesis submitted for the Master's degree of *Robotics Engineering*

March 2022

Prof. Ing. Matteo Zoppi — Supervisor

Phd Mohamed Sadiq Ikbal — Supervisor

Egon Carusi — External Co-superrvisor

Department of Mechanical, Energy, Management and Transport Engineering

Acknowledgements

This journey has finally come to and end. Throughout this journey, I've been lucky enough to have friends by my side who have filled my days with joy, and I'd like to dedicate this space to thanking them for all of our shared experiences.

To my pals from "La Tana," Cea, Erri, Albi, Jack, Pica, and Giulio, for being my family for all these years and for always inspiring me to have higher and higher ambitions.

For my lifelong pals, who I know will always be by my side and that I can always count on their presence, Ste, Marino, Erik, Delco, Fil, Leo, Compa, Sticcia, Ico, Fela and Simon.

To my collegues from "Il Banano", Titti, Luca, Fra, Sere, and Isa. I met real friends in you, and we had a great time together between music, alcohol, studying, and evenings spent together.

To my family, who have always been supportive of me during my university career and who represent home and love to me. It is only because of you and your unwavering faith in me that I am receiving all of this today.

To Sadiq, who has been a mentor throughout my thesis work and has always managed to motivate me to achieve better and better despite the challenges I've faced.

And, as is frequently stated, the best is saved for last.

To my beloved parter Cecì, you have been an invaluable presence in helping me reach my objectives. You've been my cheerleader, encouraging me to get back up after every fall. I received love, attention, and passion from you and I can't think how my life would be without you.

Abstract

Industry and academics have utilized motion simulators extensively to teach pilots, perform psychological research on drivers, investigate how people perceive motion, and cater to the expanding gaming industry, among other things. Motion simulators are sophisticated equipment that require the expertise of skilled engineers to develop and operate. The building of a dynamic simulation model that emulates the motion simulator in all aspects can make the development of projects involving it easier. Multibody dynamics is one of applied mechanics' fastest growing fields. Robots, mechanisms, chains, cables, space structures, and biodynamic systems are all examples of multibody systems that are increasingly being used as models. This thesis presents a solution to the problem of modeling a parallel robot using a graph-based development environment, with declarative and acasual programming paradigms in order to obtain a multi-domain cyber-physical counterpart of the existing parallel robot SP7. The model is built using a top-down programming approach, which allows the developer to break down the system into smaller parts. The functionality of the proposed model is validated comparing its accuracy and precision with the SP7, taken as ground truth system, and performing statistical hypothesis tests to get a statistical result of the simulation performed.

Keywords: *Simulation Modeling, Parallel Robots, Modelica, Multi-domain modeling.*

Table of contents

List of figures **v**

List of tables **viii**

Nomenclature **ix**

1 Introduction **1**

1.1 Multi-body dynamic simulation 3

1.2 Problem Definition 4

1.3 Problem Approach 6

1.4 Structure of the Thesis 6

2 Literature Review **8**

2.1 System Modeling in the literature 8

2.2 Feature Criteria Selection 9

2.3 Simulation environments 10

2.4 Final Comparison 12

3 Implementation **14**

3.1 OpenModelica and Modelica 19

3.1.1 Components of Modelica Standard Library 20

3.1.1.1 Blocks 20

3.1.1.2 Mechanics 25

3.1.1.3 Electrical 29

3.2 External C Functions 32

3.2.1 Example 33

3.3 System Architecture 35

3.4 SP7 Model 36

3.4.1 SP7 dynamic model 37

3.4.1.1 Base . . . 37
3.4.1.2 Six_rss_legs . . . 38
3.4.1.3 Rss_leg . . . 39
3.4.1.4 Platform . . . 41
3.4.1.5 Six_rss_closedloop . . . 43
3.4.2 Simulation Control Unit . . . 44
3.4.2.1 DCPositionAxis . . . 45
3.4.3 Full Model Implementation . . . 46
3.5 UDP Socket . . . 47
3.6 Windows Application . . . 49

4 Testing Phase and Results 50
4.1 Calibration of Model Parameters . . . 53
4.2 Signal Flagging . . . 55
4.3 Data Postprocessing . . . 57
4.4 Accuracy Analysis . . . 59
4.4.1 Design of Experiments . . . 59
4.4.2 Results . . . 62
4.5 Precision Analysis . . . 68
4.5.1 Design of Experiments . . . 68
4.5.2 Results . . . 69
4.6 Conlusions . . . 71

References 73

Appendix A Step by Step Experimental Procedure 78

List of figures

1.2.1 CAD Model of the 6RSS-R parallel manipulator 5

3.0.1 MotorPower - Duet FLexi 80 2.0 15
3.0.2 MotorPower - Duet FLexi 80 2.0 Dimensions 15
3.0.3 SPINEA - Drive Spin 110-119 15
3.0.4 SPINEA - Drive Spin 110-1190 Dimensions 16
3.0.5 SP7 Base and Platform location of legs 17
3.0.6 Workspace envelope projections of SP7 generated in Maple. 18
3.1.9 MSL Component - RealInput and RealOutput - Icon 21
3.1.10 MSL Component - LimPID - Icon 21
3.1.11 MSL Component - LimPID - Diagram View 22
3.1.12 MSL Component - Sources components - Icon 22
3.1.13 MSL Component - CombiTimeTable - Icon 23
3.1.14 MSL Component - CombiTimeTable - textfile 24
3.1.15 MSL Component - TerminateSimulation - Conditional Code 24
3.1.16 MSL Component - Gain - Icon 25
3.1.17 MSL Component - Frame_a and Frame_b - Icon 25
3.1.18 MSL Component - Flange_a and Flange_b - Icon 26
3.1.19 MSL Component - World - Icon 26
3.1.20 MSL Component - Revolute joint - Icon 27
3.1.21 MSL Component - Spherical joint - Icon 27
3.1.22 MSL Component - SphericalSpherical joint - Icon 28
3.1.23 MSL Component - SphericalSpherical joint - Animation View 28
3.1.24 MSL Component - Fixed - Icon 28
3.1.25 MSL Component - FixedTranslation - Icon 29
3.1.26 MSL Component - FixedTranslation - Animation View 29
3.1.27 MSL Component - BodyShape - Icon 29
3.1.28 MSL Component - BodyShape - Diagram View 29

3.1.29 MSL Component - Positive and Negative Pins - Icon 30
3.1.30 MSL Component - SignalVoltage - Icon 30
3.1.31 MSL Component - DC_PermanentMagnet - Icon 31
3.1.32 MSL Component - DC_PermanentMagnet - Diagram View 31
3.1.33 MSL Component - DCPM_CurrentControlled - Diagram View 32
3.2.34 Modelica Models Translation Process 33
3.2.35 Modelica External C Function Declaration 33
3.2.36 Modelica External C Function - Example Result Plot 35
3.3.37 OpenModelica software Architecture 36
3.4.38 Modelica Import Component - Code 37
3.4.39 Base model - Icon 37
3.4.40 Base model - Diagram view 37
3.4.41 Base model - Parameters 38
3.4.42 Six_rss_legs - Icon 39
3.4.43 Six_rss_legs - Diagram View 39
3.4.44 Rss_leg First Implemented Model - Diagram View 39
3.4.45 Fourbar Closed Loop Model - Diagram View 40
3.4.46 Rss_leg Final Implemented Model - Diagram View 41
3.4.47 Platform - Icon 42
3.4.48 Platform - Diagram View 43
3.4.49 Six_Rss_ClosedLoop Final Implemented Model - Diagram View . . . 44
3.4.50 DC_PositionAxis - Diagram View 45
3.4.51 Full Model Final Implemented Model - Diagram View 46
3.4.52 Full Model 47
3.5.53 Software Architecture with UDP Socket 48

4.0.1 SP7 Control Architecture 51
4.0.2 SP7 Experimental Architecture 52
4.1.3 DC Motor PID Calibration Model - Diagram View 53
4.1.4 PID Tuning - Proportional Gain 54
4.1.5 PID Tuning - Integral Gain 55
4.2.6 Trajectory Flagging - Head Flag 56
4.2.7 Trajectory Flagging - Tail Flag 57
4.3.8 SP7 Control Application User Interface 58
4.4.9 SP7 Platform Desired Trajectory - Cartesian Space 61

4.4.10 SP7 Joint Positions 63
4.4.11 Modelica Model Joint Positions 64
4.4.12 Accuracy Analysis - Joint Error - j1 65
4.4.13 Accuracy Analysis - Joint Error - j2 65
4.4.14 Accuracy Analysis - Joint Error - j3 65
4.4.15 Accuracy Analysis - Joint Error - j4 65
4.4.16 Accuracy Analysis - Joint Error - j5 66
4.4.17 Accuracy Analysis - Joint Error - j6 66
4.4.18 Accuracy Analysis - Joint Error - j7 66
4.4.19 Accuracy Analysis - Boxplot of Joint Errors 67
4.5.20 Precision Analysis - Joint Error - j1 69
4.5.21 Precision Analysis - Joint Error - j2 69
4.5.22 Precision Analysis - Joint Error - j3 70
4.5.23 Precision Analysis - Joint Error - j4 70
4.5.24 Precision Analysis - Joint Error - j5 70
4.5.25 Precision Analysis - Joint Error - j6 70
4.5.26 Precision Analysis - Joint Error - j7 71

List of tables

2.1 MultiBody Dynamic Tools Comparison 12

3.1 Parallel Robot Geometry 17
3.2 Base Position of Leg Housing 17
3.3 Platform Position of Leg Housing 17
3.4 SP7 Workspace Bounsaries 18
3.5 Control Unit - Gains value mapping 45
3.6 Nominal parameter for MP DUET Flexi 80 2.0 46
3.7 Control Unit - Gearbox Ratio 46
3.8 Modelica configuration variables 49

4.1 Control PID parameter - Control Unit 55
4.2 Accuracy Analysis - Hypothesis test 59
4.3 Trajectory combinations 60
4.4 Accuracy Analysis - Statistical Results 67
4.5 Accuracy Analysis - Hypothesis Test 67

Nomenclature

General Kinematics/Dynamics

$\vec{\tau}$	Joint torques
$\vec{f}$	End effector applied force
$\ddot{\vec{\theta}}$	End effector angular acceleration
$\ddot{\vec{x}}$	End effector linear acceleration
$\dot{\vec{\theta}}$	End effector angular velocity
$\dot{\vec{x}}$	End effector linear velocity
$\vec{\theta}$	End effector angular position
$\vec{x}$	End effector linear position
R	Revolute joint
S	Spherical joint

Non-SI Accepted Unit

°	Degrees [°]

SI Units

m	Meters [m]
rad	Radiants [rad]

Statistical Quantities

α	Statistical Significance index

$\bar{d}$ Mean Error of a Sample

δ_{avg} Null Hypothesis Average Error

δ_{max} Null Hypothesis Maximum Error

d_{max} Maximum Error of a Sample

d_{min} Minimum Error of a Sample

s^2 Standard Deviation of a Sample

Z_α Z-test Score at α

Acronyms / Abbreviations

AET Average error test.

DOF Degree of Freedom

ee End effector

MBS Multi Body Simulation

MM Modelica Model

MPET Maximum permissible error test

MSL Modelica Standard Library

PWM Pulse Width Modulated

Chapter 1

Introduction

Parallel robots are closed-loop mechanisms presenting very good performances in terms of accuracy, rigidity and ability to manipulate large loads [43]. The first appearance of a parallel manipulator concept was in 1934, with the spray painting machine designed by Willard PollardJr. [49], a 3 DOF mechanism consisting of an electrical control system and a mechanical manipulator driven by two rotary motors. In 1965, the advent of the Stewart Platform [55] marks a turning point in the development of robots: the mechanism consists of six prismatic actuators which drive a moving platform, or end-effector, resulting on a *six-axis* motion (6 DOF). The legs are connected to the fixed base with universal joints and to the platform with spherical joins. This system is typically applied in flight simulator, machine tool technologies, crane applications, marine technology, the Hexapod-Telescope, robots, and others. Throughout history, research has produced various types of parallel robots that vary in mechanics, freedom of movement and field of application. Another common design is the Delta robot, originally created in the 1980s, that is a parallel robot with three arms that are joined at the base by universal joints and that due to its design, can control the end effector in order to perform pure translating motions in the space. With this architecture, the Delta has a very low inertia and can manipulate light pieces within a very short cycle time. Delta robots can be commonly found along many assembly lines across numerous industries with applications that focus on pick and place, material handling, part transfer, 3D printing and packaging robotic applications. Among the remaining possible parallel robot designs we can mention the 5 Bar Parallel Robot, a two degree-of-freedom mechanism that is constructed from five links that are connected together in a closed chain mostly use for pick and place applications, drawings and academic teaching, the 3 RRR Planar Parallel Robot, a special symmetrical mechanism composed of three planar kinematical chains with identical

topology, all connecting the fixed base to the moving platform, and many other variations of the mentioned designs.

Due to its constructive robustness, the Stewart platform was meant to be employed typically in flight simulations: the pilot can benefit from this mechanism by training his skills in a complete safe environment receiving the most realistic feedback from the maneuvers he is carrying out during the simulation. Motion platform were employed in flight simulators since the early 1920s for military purpose only and they were largely used as a training procedure for the pilots. Motion simulators can be subdivided as passive ride simulators, i.e. theme park rides, where the user has no control on the motions, or occupant-controlled motion simulators, like flight simulators or driving simulators, where the user drives the simulating environments using controllers to interact with it. In both cases the occupant receives motion feedback from the platform and visual feedback either from a screen mounted in front of him, or from a virtual reality system combined with the simulation. An example of virtual reality applied to flight or driving simulations can be found in the Yaw VR motion simulator by Yaw VR ltd, a compact virtual reality motion simulator that encapsulates all the above mentioned features.

An important factor in obtaining the most realistic response from the platform is to describe the relationship between forces and motion to increase system stability and precision and to obtain small amplitudes of vibration [60]. It is therefore decisive to design the most accurate dynamic model in order to take into account the kinematics and the limitations of the robot employed. As a direct consequence, it is important to rely on simulation tools either to work out any flaws in design before the system is installed or to analyze the current mechanism to predict and prevent any malfunction fault. Simulation tools are built to replicate real-world applications as closely as possible, taking every environmental and physical factor into account, and to produce tests for all possible variables. The result of such a process is to have an accurate estimation of the equipment's condition through a virtual representation that serves as the real-time digital counterpart of the physical object.

The aim of this thesis is to develop a full-fledged simulation model for the already existing SP7 motion simulator that incorporates all facets of the robotic manipulator from the actuator to kinematics. The proposed work is divided between an initial phase with a comparison of software architectures for creating a multi-body dynamic simulation model, and a second phase in which it will be developed and fully tested the simulation model generated in the process. By integrating the model in a simulation environment it is possible to improve the operating processes involved with the platform and to evaluate the feasibility of any motion at the actuator level before experimenting with the actual platform. In general a simulation

approach brings out benefits on the majority of the task since it provide an accelerated, safe, and fully controlled virtual testing and verification environment. It should also be noted that the development of such a model can play a substantial role in the eventuality of a redesign of the mechanism by not affecting the overall kinematics.

Nevertheless, the proposed thesis is a encouraging opportunity to revisit the skills acquired during these years of study by working in a transversal and complete project under the aspects of mechanics, physics, electronics and robotics.

1.1 Multi-body dynamic simulation

Classical mechanics is the basis of multibody dynamics. Its engineering applications range from mechanisms to gyroscopes, satellites, robots, as well as into biomechanics. A multibody system can be defined as a collection of rigid and/or flexible bodies inter-connected by kinematic joints and possibly some force elements. A body is said to be rigid when its deformations are assumed to be small such that they do not affect the global motion produced by the body. Such a body can translate and rotate, but it can not change its shape. In the three-dimensional space, the motion of a free rigid body can be fully described by using six generalized coordinates associated with the six degrees of freedom, three for translations and three for rotations. The interaction may be realized by a joint, by a force actuator or a sensor and thus, the interaction is characterized by two types of information: the frames to be connected and the connecting element itself. In addition, specific correlations between the parameters and variables may be imposed. These connections (also known as restrictions) are included in the system description. With the assignment of values to each parameter and beginning conditions to each variable, the model is complete. It's important to remember that the starting circumstances must meet the limitations.

The dynamic simulation of multi-body systems becomes very important when introducing robotics into human environments [25, 39, 44] where the success will not depend only on the capabilities of the real robots but also on the simulation of such systems. For example, in applications like virtual prototyping, teleoperation, training, collaborative work, and games, physical models are simulated and interacted with both human users and robots. Dynamic modeling of a manipulator describe the relation between the actuation and acting contact forces, and the resulting acceleration and motion trajectories and it plays an important role in robot control. It mathematically describes the dynamic behavior of the structure and it can be addressed in two different approaches:

- **Inverse dynamics**: given motion variables (e.g. $\vec{\theta}, \dot{\vec{\theta}}, \ddot{\vec{\theta}}$ or $\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}$), find what joint torques ($\vec{\tau}$) or end-effector forces ($\vec{f}$) would have been the cause;
- **Forward dynamics**: given joint torques ($\vec{\tau}$) or end-effector forces ($\vec{f}$) , find what motions (e.g. $\vec{\theta}, \dot{\vec{\theta}}, \ddot{\vec{\theta}}$ or $\vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}$) would result.

This mathematical representation provides the essential information to the model based controller design. The mostly used approach is the Lagrange–Euler formalism [28], as well as the Newton-Euler equations [21, 33]. Further more, the principle of virtual work [41, 30], the Hamilton's principle [34], the Kane method [37] and the skew theory have been used in different works. For the SP7 motion simulator, a different approach is carried out as we will see in the following section.

1.2 Problem Definition

A parallel manipulator is a mechanical system formed by two linked platforms, namely, the fixed platform and the moving platform. The moving platform is connected to the fixed platform by at least two independent computer-controlled serial chains or limbs working in parallel. Compared with their serial counterparts, parallel manipulators are essentially more accurate and rigid [31], but possess smaller workspace and dexterous maneuverability. Parallel robots are very attractive for several applications because the manipulated load is shared by several legs of the system and each kinematic chain carries only a fraction of the total load [26].

The robot involved in this thesis is a 6RSS-R parallel manipulator: the Stewart-Gough platform in Figure 1.2.1 consists a moving platform connected to the base by means of 6 legs, where each leg is a serial chain composed of a crank linked to the base and a spherical-spherical rod. 6 revolute actuators at the base are active joints, which are actuated to perform and allow six dimensional motion of the platform. An independent revolute actuator positioned under the moving base drives the platform to move along its yaw axis. SP7 is located in the PMAR lab of the DIME department of the University of Genova and it was center of research and studies of the previous years. The development of a motion generation and planning system for a virtual reality-based motion simulator was the primary project in which the platform was employed. PMAR Robotics laboratory in collaboration with the virtual reality gaming company Singular Perception s.r.l worked together to obtain a virtual reality scenario application with custom visual, audio, and motion cues. Nevertheless, Moreover, a simulation environment for an obstacle avoidance motion

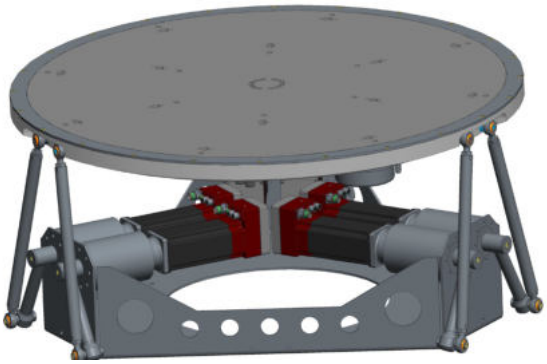


Figure 1.2.1 CAD Model of the 6RSS-R parallel manipulator

feedback for teleoperated drones is currently being developed and includes the use of the platform as a flight simulator, the HTC VIVE HDM device as a virtual reality system and Unreal Engine 4 as the development environment for the drone flight dynamics. The work proposed by this thesis project arises from the lack of a simulation system that can replace the real platform during the development process of works such as those just mentioned: the SP7 can in fact be controlled through the use of an application developed at PMAR laboratory and inverse kinematics studies have previously been performed for the bivalent generation of a control trajectory in joint space or in Cartesian space. What is missing, however, is a system that can mimic the behavior of the platform taking into account the dynamics of the mechanism at the simulation level. A direct kinematics algorithm is currently missing so the model implemented during this work can be validated though an accuracy analysis with the data coming from the encoders of each actuator. The requirements of the projects are the following:

- Implement a dynamic simulation model of the SP7;
- Perform a literature review on software architectures for creating a multi-body dynamic simulation for a robot;
- Implements the actuators models to analyze torque and power consumption;
- Validate the developed model.

In the following section it will be analyzed all the possible solutions to solve this problem and the applied final methodology will be explained.

1.3 Problem Approach

This section explains the key phases of the systematic modeling technique based on multibody system theory that is used to construct a dynamic model for a parallel robot. The initial step in this project will be to find a trustworthy simulation environment in which to simulate the mechanics and, eventually, the control of a parallel kinematic structure. As a result, the ability to develop models for implementation across several domains, such as mechanical, electrical, and physical, will be required. Once the development environment has been chosen, it will be required to determine the parameters for the model's development: the model will be separated into sub-components in order to perform a modular development, which will allow for a better examination of each multi-domain sub-component. This can shorten development time and allow for focused and universal changes. You should ideally choose a development environment that allows you to work using an object-oriented programming approach to subdivide the parallel mechanics into three parts: base, legs, and platform. After the model's mechanics have been developed, we'll move on to developing the control unit that will allow us to drive the model along pre-determined paths. As a result, it will be required to appropriately implement the components offered by the chosen development environment in order to build a control model that behaves identical to that of the real platform. The model required can be fed in different ways: the most recommended solution is to replicate the architecture of the already existing system in order to replace it in the projects that will follow. For this use case, the results of the inverse kinematics problem will be used and desired input of the model.

The model will be developed at the PMAR laboratory. The following stage will be to determine how the validation trials will be carried out. In order to do this, a defined set of experiment will be chosen and the analysis of the resulting measurements will held to a final approval.

1.4 Structure of the Thesis

The thesis is structured in four chapters organized as follows: **Chapter 2** describes the study of the art for the research of the most suitable software architecture for the development of the dynamic model of the motion simulator. The software currently most used by the scientific community are analyzed here until the final selection is made. **Chapter 3** begins with a description of the SP7 motion simulator, which includes the robot's shape and actuators. The remainder of the chapter concentrates on the model's software implementation: following a

thorough discussion of the development environment chosen, the produced components are explained, and the architectural design of each component of the model is provided. Lastly, **Chapter 4** evaluates the experiment design for the validation phase of the model: the first section of the chapter describes how data is post-processed in order to compare the results of the simulations, and the second section displays the analysis and the final assessment based on the results.

Chapter 2

Literature Review

2.1 System Modeling in the literature

In this section there will be evaluated different solutions among a list of simulation tools with the aim of finding the most suitable software to implement the dynamic model of the SP7 platform. It is important to understand the nature of a dynamic model and to find objective evaluation criteria in order to catalogue and discard the solutions less fitting for our goal by deploying a real-time model capable of reducing the overall complexity of the robot and increase the accuracy of the simulation, avoiding unnecessary data processing with the goal of fault prevention and redesign application. There have been a number of general-purpose simulation modeling paradigms and languages established in the research field: graph-based vs language-based paradigms, procedural versus declarative models, multi-domain versus single-domain models, continuous versus discrete models, and functional versus object-oriented paradigms are some of the classification criteria [54]. Graph-based modeling abstractions are designed to provide a consistent framework for capturing modeling features that appear in a variety of engineering disciplines while also visually simplifying the development process. Declarative or equation-based languages do not impose a causality constraint on the model: a set of equations defines the model, which establishes connections between states, derivatives, and time. The implemented solver is in charge of converting these equations into software methods that can be evaluated by computers. Modelica [4] and VHDL-AMS [27] are two prominent recent attempts aimed at developing sophisticated simulation languages capable of simulating large multidisciplinary systems. Both languages have a similar goal and expressiveness in general: they both provide continuous and discrete temporal modeling in a variety of energy domains, as well as declarative modeling and hierarchical encapsulation. Modelica also has the benefit of being true object-oriented, with

support for model inheritance and sub-typing. It also facilitates the creation of aggregate connections which integrate several signal variables from various energy domains. Defining the development design of a dynamic model can be achieved in different way, and for this purpose it is useful to assess the analysis carried out by Aivaliotis at al. [19]: this paper presents a methodology for advanced physics-based modeling with the intent of define, create and utilize the digital model of a resource. In the paper, the modelling process of the DT is divided in three phases:

- **Phase 1**: Dynamic behavior;
- **Phase 2**: Virtual sensors;
- **Phase 3**: Modeling parameters.

In the first phase the selection of model components occurs, as long as the definition of a model leveling (in *white/grey/black boxes*) determined by the available data for the parametrization of the model. After the composition of all model components, in the phase two it is required the definition,the selection and the integration of proper virtual sensors to monitor and gather data from the machine's model during the simulation. Finally, in the last phase will occur the definition of the modelling updateable parameters in order to adjust the model to the real machine's behavior based on the controller and sensor data. What has been explained so fare give us a hint on how to proceed for the software architecture selection: the most fitting solution for our problem will be a block diagram graph-based modeling tool which implements a declarative programming paradigm and provides multi-domain approach which is a decisive feature for the behavior of mechatronic systems.

2.2 Feature Criteria Selection

The methodology aforementioned helps to find the criteria for the software selection. Among all the important parameter and conditions, the following has been chosen as feature selection criteria, in a scale of importance:

1. **Multi-domain environment**: for a complete simulation it is unavoidable the use of simulation components that replicate actuator dynamics and consumption. The need for a multi-domain environment is straightforward;
2. **Graph-based Paradigm**: since it may not be possible to describe all the real components with complete fidelity, it is necessary to be able to add a degree of abstraction in

the composition of the model. Graph-based paradigm confers the needed amount of abstraction for the system modeling;

3. **Real robot interfacing**: is the need to use the same input data for the real robot and for the virtual model and to guarantee the interaction between them. Furthermore, the possibility to interface the developed model with external devices, e.g. joystics of spacemouse, can be required for future implentations;
4. **Sensor integration**: it is important to use virtual sensors to acquire simulation data which will be compared with the real one, e.g. motor applied torques, joint positions, current consumption, etc.;
5. **Data postprocessing**: for a correct tuning and evaluation of the output of the simulation it is required the possibility to postprocess the data. The interaction with other tool (i.e. MATLAB) for the postprocessing step is also accepted.
6. **Dynamic and Kinematic modelling**: it is necessary to be able to analyze the direct and inverse dynamic of the model for a correct study of the forces involved on the executions. The model will be fed with the desired positions either of the platform or of the joints; in both cases, it required to analyze both platform and joins position in the result files;
7. **Solver type**: it is necessary to work with a numeric solver, to avoid simplifications and to extract the exact value of each variable at any time instant. symbolic translation can be used for the translation of the model before the simulaiton;
8. **Import CAD**: it is convenient that the software could import CAD models from other tools since it already exists and it doesn't have to be made from scratch. This can be used for animating the simulation of each trajectory to have both a data measurements and a graphical view of the results;

2.3 Simulation environments

In this section will be shown the list of software analyzed for the comparison. The research of such tools was based on the requirements selected: the goal was to find software capable of multibody dynamic modelling for mechatronics systems. The search was conducted entirely online, employing keywords such as *multibody*, *multi-domain*, *CAD*, *mechanics*, and *models*, as well as a thorough examination of the software's official websites and a search for case studies in which these programs were utilized to locate the most similar application in comparison to the one offered in the thesis. The software discovered differs

from one another in a number of ways, and it satisfies the demands given in the preceding part on multiple levels. Furthermore, not every software supports multi-domain modeling, and a free solution was picked among the many license alternatives. The softwares that requires a commercial license also provides academic free editions for limited amount of time: students editions were mainly avoided since restriction on the model size and on the computational load are made by the producers. Each simulation software program was designed to implement its model in a variety of methods, but the most frequent were those with a graphical interface, those based on diagram-blocks, and a small number that needed programming in a programming language. As for modeling using the Modelica language, there are a variety of development environments that support it. For the reasons stated above, it was decided to exclude software having a commercial license, such as Dymola, and among the possible Modelica-based solutions, it was decided to consider OpenModelica in the final comparison. Tus, the software selected for the comparison are the following:

OpenModelica, an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage [56, 50, 19, 58, 29, 20]. OpenModelica can be employed to model a big variety of models thanks to the standard libraries developed for it. Modelica-based softwares can be used to model a multi-domain cyber-physical robotics systems as it is done for YouBot from Kuka in [14]. ***Recurdyn***, an interdisciplinary, commercial CAE software package whose primary function is the simulation of Multi-Body Dynamics, with cutting-edge finite element technology for modeling flexible bodies [24, 38, 40]. ***Ansys Motion***, a third-generation engineering solution based on an advanced multibody dynamics solver [48, 42] that works as an extension of the pure mechanical simulation environment Ansys Mechanical, mainly used for finite element analysis, vibration and fatigue life analysis. ***COMSOL***, a general-purpose simulation software based on advanced numerical methods, with fully coupled multiphysics and single-physics modeling capabilities [59, 52]. ***Simulia Simpack***, a general multibody system simulation (MBS) software enabling analysts and engineers to simulate the non-linear motion of any mechanical or mechatronic system [2]. ***Maplesim***, a Modelica-based, multi-domain modeling and simulation which generates model equations, runs simulations, and performs analyses using the symbolic and numeric mathematical engine of Maple [32]. ***FreeDyn***, a free simulation software designed for solving industrial problems in multibody dynamics with systems consisting of flexible bodies. ***MBDyin***, a free general purpose Multibody Dynamics analysis software, released under GNU's GPL 2.1 [23]. ***SimMechanics***, a multibody simulation environment for 3D mechanical systems, such as robots, vehicle suspensions, construction equipment, and aircraft landing gear [45, 46, 35]. ***Robotran***, a powerful tool that analyze mechanical systems using

a symbolic multibody approach [53]. ***MSC ADAMS***, a multibody dynamics simulation software system widely used in analysis of vehicle structure and suspension [57, 51].

2.4 Final Comparison

In Table 2.1 it is presented the final comparison among the mentioned simulation tools: the table presents in the rows all the software and in the columns all the criteria: The table

Table 2.1 MultiBody Dynamic Tools Comparison

SOFTWARE	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
MSC ADAMS	✓	✗	✓	✓	✓	✓	✓	numeric	✓
Recurdyn	✗	✗	✓	✗	✓	✓	✓	numeric	✓
SimMechanics	✓	✗	✓	✓	✓	✓	✓	numeric	✓
OpenModelica	✓	✓	✓	✓	✓	✓	✓	numeric	✗
MapleSim	✓	✗	✓	✓	✓	✓	✓	symbolic	✓
Ansys Motion	✗	✗	✓	✗	✗	✓	✓	numeric	✓
COMSOL MBD Module	✓	✗	✓	✓	✓	✓	✓	numeric	✓
Simpack	✗	✗	✗	✓	✓	✓	✓	numeric	✓
Freedyn	✗	✓	✗	✗	✗	✓	✓	numeric	✓
MBDyn	✗	✓	✗	✓	✓	✓	✓	numeric	✓
Robotran	✗	✓	✓	✓	✓	✓	✓	symbolic	✓

[1] Multi-domain environment
[2] Licence payment (✓free - ✗commercial)
[3] Graph-based paradigm
[4] Real robot interfacing
[5] Sensor integration
[6] Data postprocessing
[7] Dynamic and Kinematic modelling
[8] Solver type
[9] CAD import

summarizes what was found during the research, that is how many softwares allow to model cyber-physical system on multiple domains and how some focus only on a mechanical analysis of them. Despite having found a software that fully satisfies the defined criteria,

OpenModelica is the one that boasts all the most important features: it can be used for modeling mechatronic systems and the presence of standard libraries will simplify the development process. OpenModelica is widely used many different purpose and it is possible to find a good amount of case stydies in the literature [56, 50, 19, 58, 29, 20]. The simulation environment will be described in the next chapter, followed by an examination of the model's implementation in all of its components.

Chapter 3

Implementation

The following chapter will describe how the selected simulation environment has been used to develop a working model for the SP7. After a complete description of the system architecture and the tools used for the project, we will analyze the implementation of the mechanical and geometric structure and the development of the control unit that controls the actuators.

SP7 is a parallel robotic manipulator with 7 DOF. The design of the mechanism is similar to the 6RSS-R parallel manipulator commonly known as the “Stewart platform” [55] with an additional yaw axis actuator making it a redundant axis. The mechanical design, kinematics, control strategies, workspace analysis, actuator selection process, etc. have been developed at the PMAR laboratory team. The actuators installed in the platform are basically of two types: one implemented as driving unit of each leg and one used to drive the plate of the platform. The DUET FLexi 80.2.03 actuator from MotorPower (Fig 3.0.1 is selected for the control of the six parallel legs: this servomotor has been selected since it has a compact design, CANopen interface for integration in automation systems, it can be used in torque, speed and position control and ensures high precision nominal torque at low speed and low voltages. At the motor connection, The DUET FL servo positioning controller supplies the synchronous machine with a PWM, symmetrical, 3-phase rotating field with variable frequency, current and voltage. The DUET FL is designed for a continuous torque, speed and position control in typical industrial applications such as positioning and feeding drives in machines. n the other hand, the DriveSpin 110-119 from SPINEA has been chosen for the vertical rotation axis because the unit provides rotary motion and torque transfer with a high radial-axial load capability and high moment overload capacity. This becomes crucial for application like the SP7 motions simulator, since all the load of the platform plus the additional weights of the user and the pilot seat strains on the axial direction of the installed actuator; in order to ensure

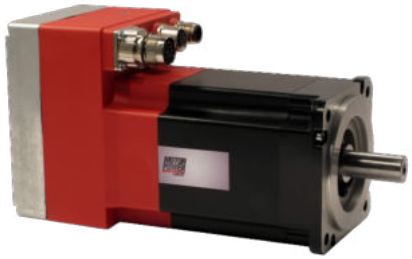


Figure 3.0.1 MotorPower - Duet FLexi 80 2.0

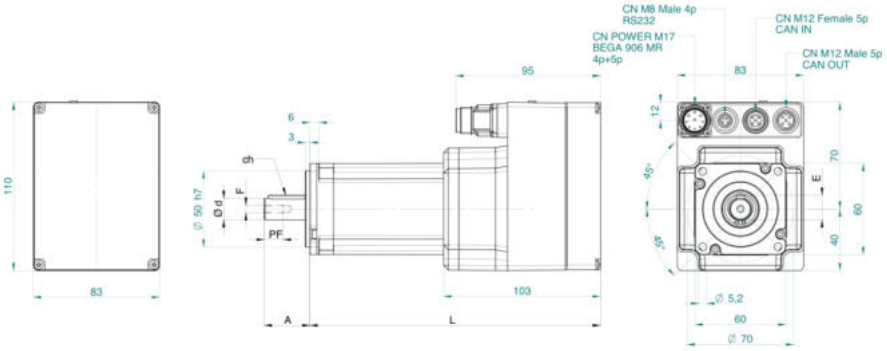


Figure 3.0.2 MotorPower - Duet FLexi 80 2.0 Dimensions

the best motion experience it is therefore necessary to select an actuator with the mentioned features. here are several works of literature on the various development aspects of this

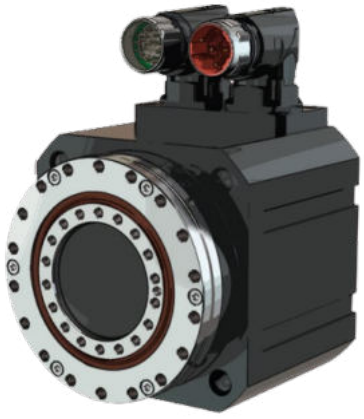


Figure 3.0.3 SPINEA - Drive Spin 110-119

type of manipulator. For instance, [15] provides an implementation procedure for the inverse kinematics of the manipulator, [36] describes the actuator control and parameter tuning

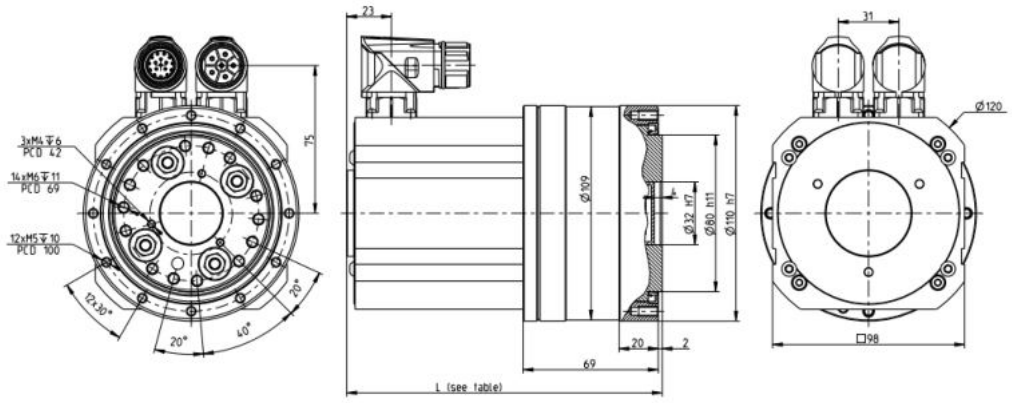


Figure 3.0.4 SPINEA - Drive Spin 110-119 Dimensions

strategies, etc. Thus, the specifics of the SP7 platform are not added in this research. What is relevant for the development of the model is the geometry of the platform, the weights of each component and the specifics of the actuators installed. As already mentioned in the introduction, the platform in connected to the base by means of six legs: each leg is a serial chain with one revolute joint, referred as j_i with $i = 1,2\ldots,6$, a crank attached to it and one spherical-spherical rod. In Fig. 3.0.5 it is illustrated how the six legs are positioned in the platform and in the base: $B_{1,6}$ represents the positions of the revolute joints in the base and are defined as shown in Table 3.1; $P_{1,6}$ represents the housing of the spherical joints of each leg in the platform. Legs are numbered from 1 to 6 in a counterclockwise order starting from the positive direction on the x axis of the robot reference frame, which is centered at the base and oriented as show in Fig. 3.0.5. The radius of the platform and the radius of the base are equal and they are referred as R in the geometry illustration. Table 3.1 shows the lengths of the crank, rod and radius R. O_p is the relative reference frame of the platform and represents the position and orientation of the *ee* of the platform. The platform zero pose is: $ZeroPose = [\ 0\ 0\ 0.401\ 0\ 0\ 0\ ][m]$ and represent the position of the reference frame O_p w.r.t. the platform coordinate frame O_b projected in O_b.

In previous works at the PMAR lab, a complete analysis of the platform workspace was conducted with the goal to select a prescribed workspace for the motion simulator platform. In order to ensure continuous workspace, i.e. a set of poses reachable by the *ee* without passing through any singularities, the prescribed workspace has been identified inside the workspace envelope of the manipulator. Fig. 3.0.6 shows the workspace envelope of the SP7 generated in the Maple application. Table 3.4 lists the boundary limits of SP7 at each axis with the whole workspace and the reduced workspace capability. showfig

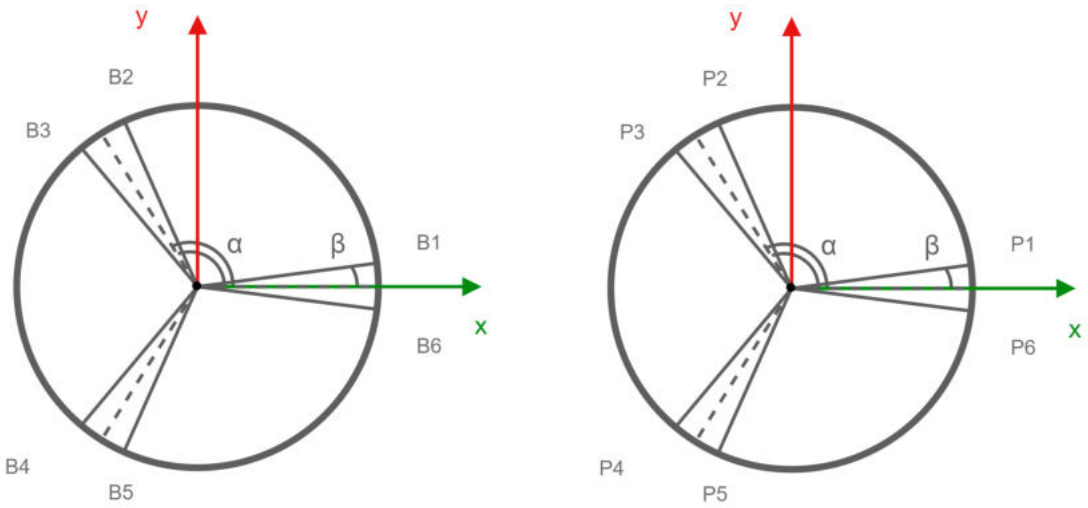


Figure 3.0.5 SP7 Base and Platform location of legs

Table 3.1 Parallel Robot Geometry

	Dimensions	**Units**
α	120	[°] degree
β	6,3794	[°] degree
R	0.5735	[m] meter
Cranck	0.19	[m] meter
SS Rod	0.45	[m] meter

Table 3.2 Base Position of Leg Housing

$B1_x = Rcos(\beta)$	$B1_y = Rsin(\beta)$
$B2_x = Rcos(\frac{2\pi}{3} - \beta)$	$B2_y = Rsin(\frac{2\pi}{3} - \beta)$
$B3_x = Rcos(\frac{2\pi}{3} + \beta)$	$B3_y = Rsin(\frac{2\pi}{3} + \beta)$
$B4_x = Rcos(\frac{4\pi}{3} - \beta)$	$B4_y = Rsin(\frac{4\pi}{3} - \beta)$
$B5_x = Rcos(\frac{4\pi}{3} + \beta)$	$B5_y = Rsin(\frac{4\pi}{3} + \beta)$
$B6_x = Rcos(-\beta)$	$B6_y = Rsin(-\beta)$

Table 3.3 Platform Position of Leg Housing

$P1_x = Rcos(\beta)$	$P1_y = Rsin(\beta)$
$P2_x = Rcos(\frac{2\pi}{3} - \beta)$	$P2_y = Rsin(\frac{2\pi}{3} - \beta)$
$P3_x = Rcos(\frac{2\pi}{3} + \beta)$	$P3_y = Rsin(\frac{2\pi}{3} + \beta)$
$P4_x = Rcos(\frac{4\pi}{3} - \beta)$	$P4_y = Rsin(\frac{4\pi}{3} - \beta)$
$P5_x = Rcos(\frac{4\pi}{3} + \beta)$	$P5_y = Rsin(\frac{4\pi}{3} + \beta)$
$P6_x = Rcos(-\beta)$	$P6_y = Rsin(-\beta)$

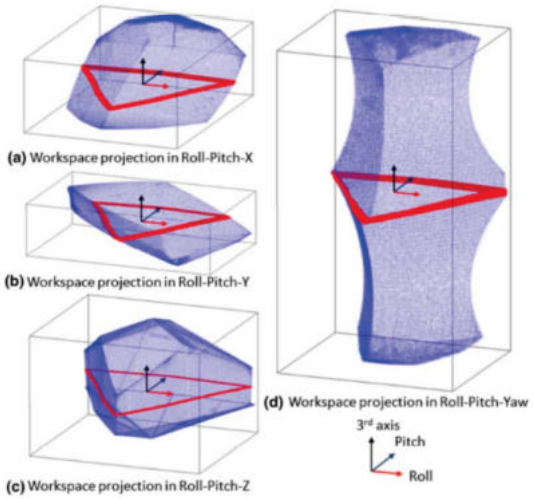
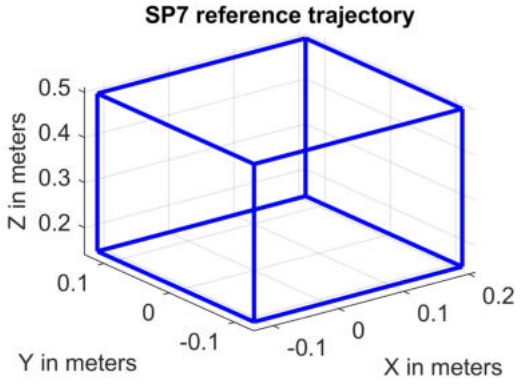
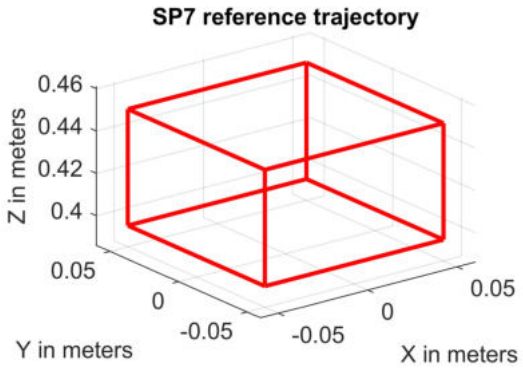


Figure 3.0.6 Workspace envelope projections of SP7 generated in Maple.



(a) SP7 Workspace - Workspace Boundaries



(b) SP7 Workspace - Prescribed Boundaries

Table 3.4 SP7 Workspace Bounsaries

DOF	**Entire Workspace**	**Prescribed Workspace**
Roll angle	$\pm 15^\circ$	$\pm 5^\circ$
Pitch angle	$+20/-13^\circ$	$\pm 7^\circ$
Yaw angle	$\pm 180^\circ$	$\pm 180^\circ$
Surge translation	$+200/-120mm$	$\pm 50mm$
Sway translation	$\pm 120mm$	$\pm 55mm$
Heave translation	$+150/+500mm$	$+396/+451mm$

Before starting with the presentation of the implementation of the model, it is necessary to dedicate a section of this paper to the development environment and programming language chosen to have a mature vision of the context in which the development will proceed.

3.1 OpenModelica and Modelica

(a) Modelica Language Logo

(b) OpenModelica Logo

OpenModelica is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage [9]. The Modelica Language is a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems. It is widely used in modeling applications since it has the following characteristics [5]:

- **Declarative language**: allowed acausal modeling, high level specification, increased correctness;
- **Multi-domain modeling**: Combine electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc...;
- **Everything is a class**: Strongly typed object-oriented language with a general class concept, Java & MATLAB-like syntax;
- **Visual component programming**: Hierarchical system architecture capabilities;
- **Efficient, non-proprietary**: Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations.

In the literature it is possible to find multiple use cases of this simulation tool for a multi-domain simulation purpose, to name a few:

Desai et al. [29] presented the study of a small size pressure control system modeled in *OpenModelica*. In the paper it is highlighted the effectiveness of using this tool to develop a DT which is similar for most of the components to the real one, but still not exactly the same.

Vathoopan et al. [58] proposed a modular corrective maintenance method by using *OpenModelica* to reduce the complexity of the model of the resource.

Aivaliotis et al. [20] propose an extension of the methodology proposed in [19] describing a predictive maintenance methodology by calculating the Remaining Useful Life (RUL) of machinery equipment by utilising physics-based simulation models and Digital Twin concept.

Oñederra et al. [47] described a MV cable model aimed at emulating its behavior and lifetime in order to carry out a preventive maintenance, reducing the costs due to downtime and increasing the reliability of the elements that intervene in the conversion of wind energy to electric power. Designing an accurate DT acquires a lot of relevance when it comes to remote locations such as offshore wind farms.

As it can be seen from the literature, *OpenModelica* is a versatile tool used in any possible scenario that can be adopted at multiple stages in the life of a product, from the pre-production design up to the predictive troubleshooting once the product is already in operation. From the point of view of this thesis, *OpenModelica* will be used to describe the electronics, the mechanics and the physics of the SP7 motion simulator, by developing and testing the model designed in the environment. In the next section will be shown the component and package names exactly as are found in the OpenModelica environment.

3.1.1 Components of Modelica Standard Library

The Modelica Standard Library is a free library developed by the Modelica Association. It includes the fundamental components for modelling mechanical (1D/3D), electrical (analog, digital, machines), thermal, fluid (1D), control systems and hierarchical state machines The components that have been used within this library for the development of the platform belong to the Blocks, Mechanics and Electrical packages.

3.1.1.1 Blocks

Interfaces.RealInput & **Interfaces.RealOutput**

Interface package contains several elements that can be used to simplify dialog between models or components.The components used for modeling in this package are RealInput and RealOutput (Fig. 3.1.9). These are connector with one input or output signal of type Real.

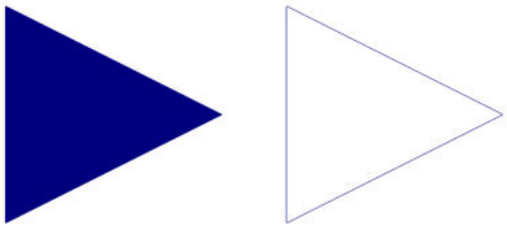


Figure 3.1.9 MSL Component - RealInput(left) and RealOutput(right) - Icon

Continuous.LimPID

Compared with the basic PID, the following controller has some advanced and performance-

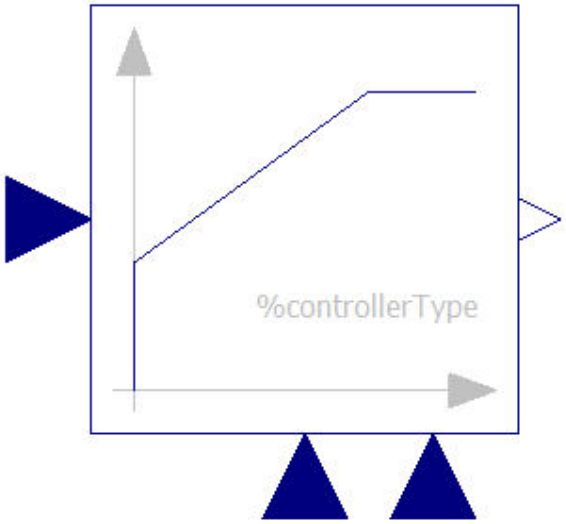


Figure 3.1.10 MSL Component - LimPID - Icon

enhancing features. P, PI, PD, or PID can be specified through the parameter *controllerType*. If PI is chosen, for example, all D-part components are eliminated from the block. Besides the additive proportional, integral and derivative part of this controller [22], the following features are present [6]:

- The output of this controller is limited. If the controller is in its limits, anti-windup compensation is activated to drive the integrator state to zero.
- The high-frequency gain of the derivative part is limited to avoid excessive amplification of measurement noise.
- Setpoint weighting is present, which allows to weight the setpoint in the proportional and the derivative part independently from the measurement. The controller will

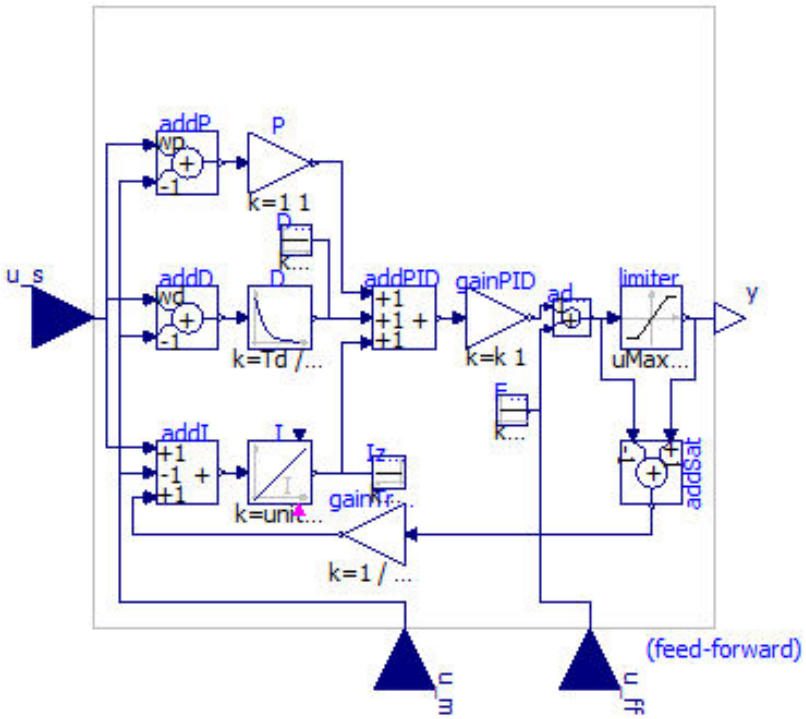


Figure 3.1.11 MSL Component - LimPID - Diagram View

respond to load disturbances and measurement noise independently of this setting (parameters wp, wd). However, setpoint changes will depend on this setting. For example, it is useful to set the setpoint weight wd for the derivative part to zero, if steps may occur in the setpoint signal.

- Optional feed-forward. It is possible to add a feed-forward signal. The feed-forward signal is added before limitation. (via conditional declarations)

Sources

This package includes source components, which are blocks that only have output signals.

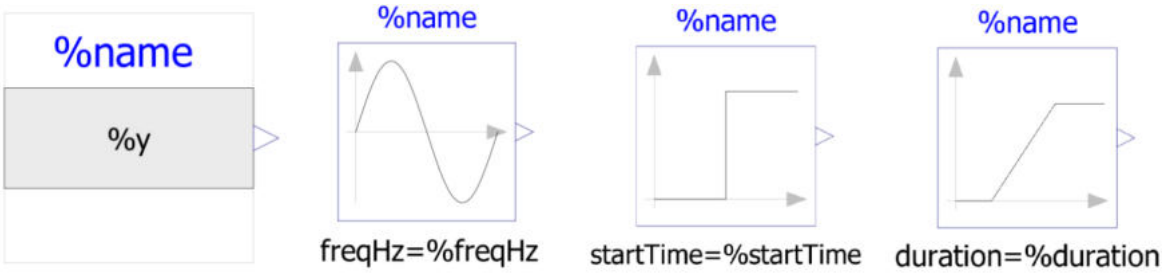


Figure 3.1.12 MSL Component - Sources components - Icon

Signal generators for Real, Integer, and Boolean signals are used in these blocks. All Real source signals have at least the following two parameters:

- **startTime**: Start time of signal. For time < startTime, the output y is set to offset;
- **offset**: Value which is added to the signal.

Source blocks was really helpful during the tuning phase of PID parameters.

Sources.CombiTimeTable

This block generates an output signal $y[:]$ by constant, linear or cubic Hermite spline

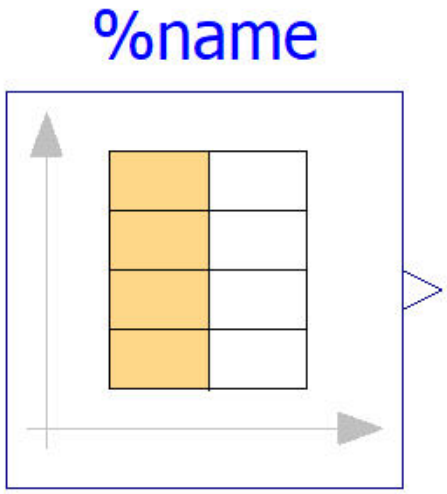


Figure 3.1.13 MSL Component - CombiTimeTable - Icon

interpolation in a table. The data to be interpolated and the time points are kept in a matrix table[i,j], with the former in the other columns and the latter in the first column table[:,1]. Data can be loaded though a text file indicating its location in the *filePath* parameter with the following syntaxt: *"Modelica.Utilities.Files.loadResource("modelica://path/to/file.txt"*). If tables are read from a text file, the file needs to have the following structure: Note, that the first two characters in the file need to be "#1" (a line comment defining the version number of the file format). Afterwards, the corresponding matrix has to be declared with type (= "double" or "float"), name and actual dimensions. Finally, in successive rows of the file, the elements of the matrix have to be given. More info in [7].

```
-------------------------------------------------------
#1
double tab1(6,2)   # comment line
  0   0
  1   0
  1   1
  2   4
  3   9
  4  16
double tab2(6,2)   # another comment line
  0   0
  2   0
  2   2
  4   8
  6  18
  8  32
-------------------------------------------------------
```

Figure 3.1.14 MSL Component - combitimetable - textfile

Logical.TerminateSimulation

This logical block is used to autonomously stop the simulation when the condition in the *condition* parameter is verified. In the parameter menu, a time varying expression can be defined to stop the simulation whenever a time instant is passed, as it is later done in the final implementation of the dynamic model.

```
equation
  when condition then
    terminate(terminationText);
  end when;
  annotation (Icon( ...);
end TerminateSimulation;
```

Figure 3.1.15 MSL Component - TerminateSimulation - Conditional Code

Math.Gain

This block computes output y as product of gain k with the input u: $y = k * u$

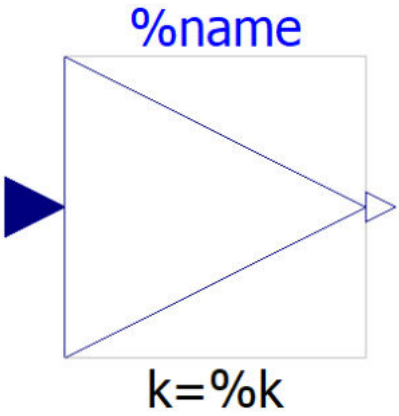


Figure 3.1.16 MSL Component - Gain - Icon

3.1.1.2 Mechanics

The MultiBody library is a free Modelica package providing 3-dimensional mechanical components to conveniently model mechanical systems, such as robots, mechanisms, or vehicles. This library is used to model rotational and translationsal movement of 1-D and 3-D in the space for mechanical systems. For the Mechanics package, the following components have been used:

MultiBody.Interfaces.Frame

These connectors represents basic definition of a fixed coordinate system on a mechanical

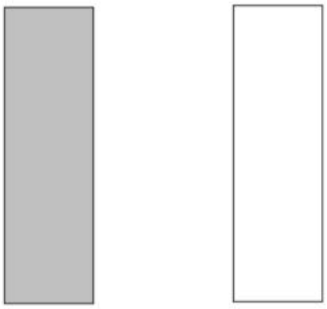


Figure 3.1.17 MSL Component - Frame_a and Frame_b - Icon

component. At the origin of the coordinate system, the cutting force and the cutting torque operate. The two components functionally correspond, but they are visually different for a better representation of models in the diagram viewer.

Rotational.Interfaces.Flange

This is a connector for 1D rotational mechanical systems and defines the angle phi of

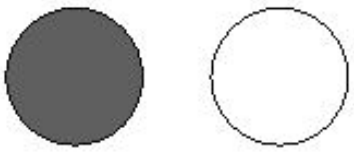


Figure 3.1.18 MSL Component - Flange_a and Flange_b - Icon

absolute rotation in [rad] and tau as the cut-torque in the flange in [Nm].

MultiBody.World

Model *World* represents a global coordinate system fixed in ground. This model can be used

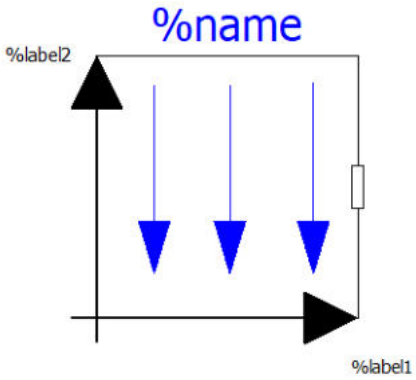


Figure 3.1.19 MSL Component - World - Icon

as an inertial system for all the mechanical components involved in the model, as a gravity field with gravity acceleration $\vec{g}$ and a visual representation for the animation view. In order to implement the gravity field function on all bodies with mass, and ensure that its default settings apply to nearly every component, an instance of model *World* must be present on the top level of every model. *Inner* and *outer* declaration parameter must be used to declare a world component on the top-level model and refer to it within the child models linked to it.

MultiBody.Joints.Revolute & **MultiBody.Joints.Spherical**

Revolute joint component can be used to add a revolute joint with one rotational DOF and two potential states. Joint where frame_b rotates around axis n which is fixed in frame_a. The two frames coincide when the rotation angle "phi = 0". Optionally, it is possible to drive the joint position through 1-dimensional mechanical flanges by enabling the parameter *useAxisFlange*. On the other hand, the *Spherical* joint is a component with three constraints

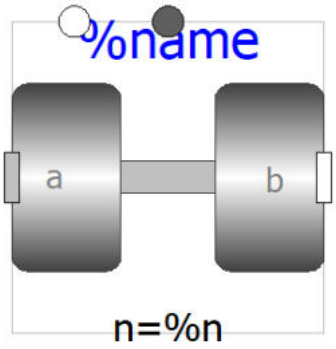


Figure 3.1.20 MSL Component - Revolute joint - Icon

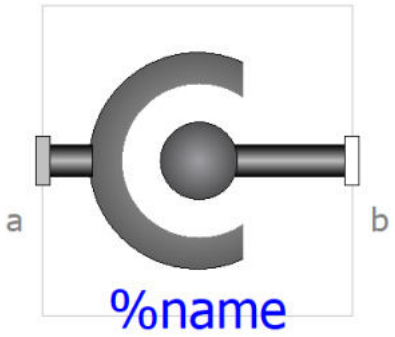


Figure 3.1.21 MSL Component - Spherical joint - Icon

that define that the origin of its frame_a and the origin of frame_b coincide. This component can be used both in chain and in loop structures, as is indicated in the documentation of Modelica library, but to do this, you must be aware of how state selection works during model translation (see Section 3.2) in order to correctly choose how to set *enforceStates* and *useQuaternions* parameters. If the model requires the use of many *Spherical* joints, it can suffer of higher time-consuming simulations since more complexity will be added to the system . A solution to this problem might be to use *Assemblies* instead. The model developed by this thesis used the *SphericalSpherical* component to reduce the number of equations created by translation and thus improve performance.

MultiBody.Joints.SphericalSpherical

Joint that has a spherical joint on each of its two ends that adds one constraint and no potential states to the system. A point mass in the middle of the rod approximates the rod connecting the two spherical joints. In the Fig. 3.1.23 representing default animation figure, the two spherical joints are represented by two red spheres, the connecting rod by a grey cylinder and the point mass in the middle of the rod by a light blue sphere. As mentioned in the Modelica library documentation, this joint should be used as often as possible in loops, as this greatly increases the efficiency due to the smaller non-linear algebraic equation systems.

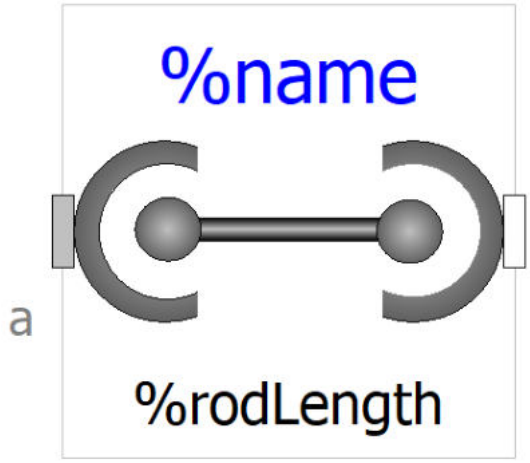


Figure 3.1.22 MSL Component - Spherical-Spherical joint - Icon

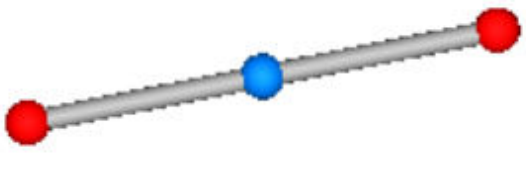


Figure 3.1.23 MSL Component - Spherical-Spherical joint - Animation View

MultiBody.Parts.Fixed

The following component forms a constraint so that the part connected to it turns out to be

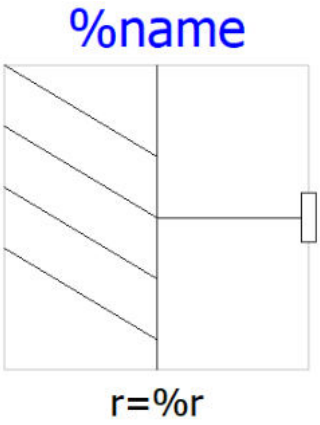


Figure 3.1.24 MSL Component - Fixed - Icon

fully bound both in translations and in rotations. The pose defined by parameter vector *r* defines how the component is positioned w.r.t. the world frame.

MultiBody.Parts.FixedTranslation

This components represents a rigid body translation of the *frame_b* w.r.t. the *frame_a*. This component is involved during the *Platform* modeling in order to rigidly connect the center of the plate with the housing of the spherical-sphrical rods.

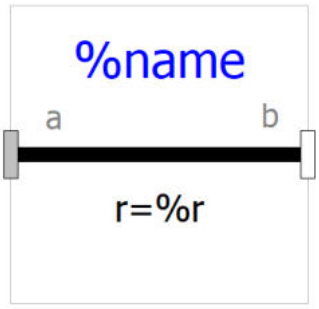


Figure 3.1.25 MSL Component - FixedTranslation - Icon

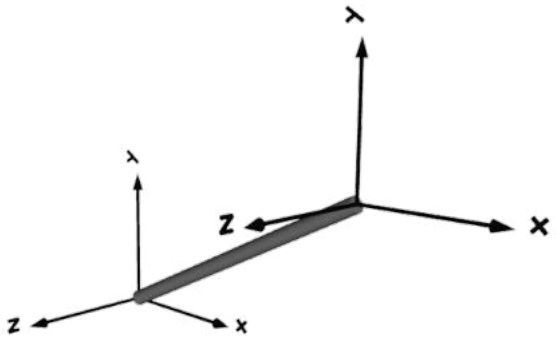


Figure 3.1.26 MSL Component - FixedTranslation - Animation View

MultiBody.Parts.Body / MultiBody.Parts.BodyShape

This models represent a rigid body with mass, inertia tensor, different shapes for animation,

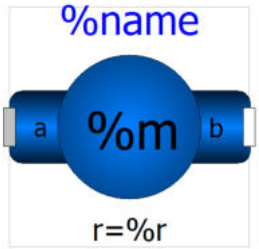


Figure 3.1.27 MSL Component - BodyShape - Icon

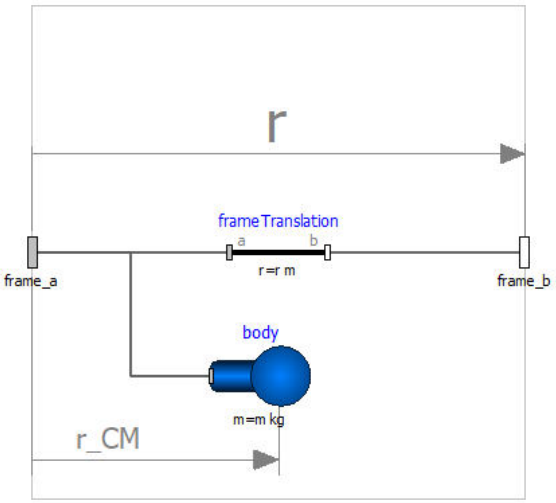


Figure 3.1.28 MSL Component - BodyShape - Diagram View

and two frame connectors. Specifying the diameter of the cylinder, the density of the material and the height as a vector all physical properties of the body are calculated.

3.1.1.3 Electrical

For the Electrical package, the following components have been used:

Analog.Interfaces.Pin

The basic electrical connector is the pin. This is used to define connecting equations for

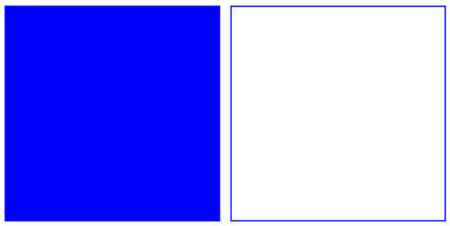


Figure 3.1.29 MSL Component - Positive and Negative Pins - Icon

voltages and currents between electrical components. Usually *PositivePin* and *NegativePin* are used for a better diagram comprehension. The connection between two pins is defined as:

$$\begin{aligned} v &= p.v - n.v; \\ 0 &= p.i + n.i; \\ i &= p.i; \end{aligned}$$

Analog.Sources.signalVoltage

This component represents an ideal voltage generator capable of providing the voltage

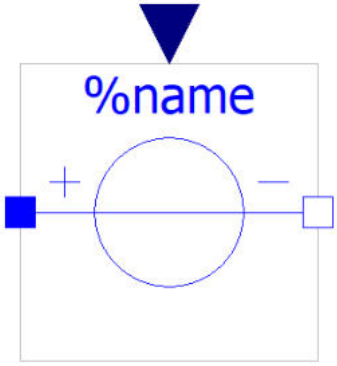


Figure 3.1.30 MSL Component - signalVoltage - Icon

specified through the appropriate port.

BasicMachines.DCMachines.DC_PermanentMagnet

This component represents a model of a DC Machine with permanent magnets. Modeling of resistance and inductance is directly based on the armature pins, and then using a *AirGapDC*

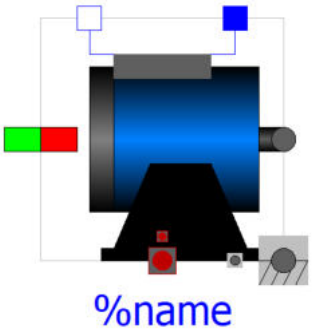


Figure 3.1.31 MSL Component - DC_PermanentMagnet - Icon

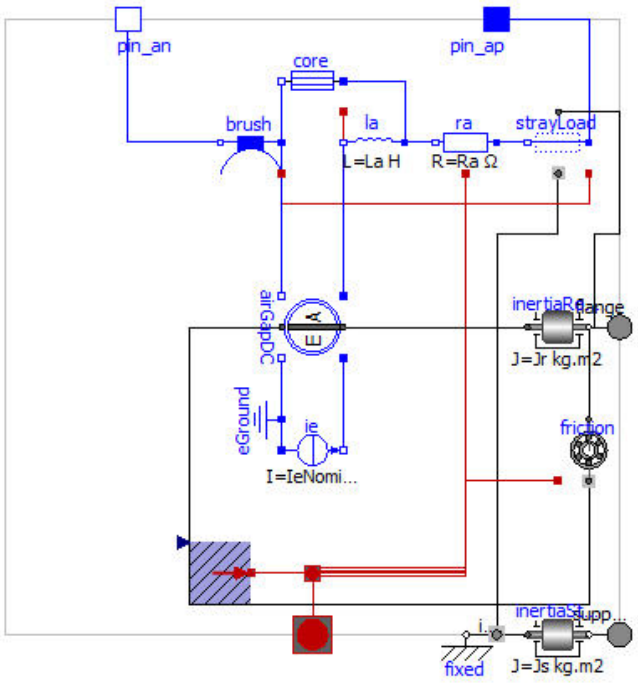


Figure 3.1.32 MSL Component - DC_PermanentMagnet - Diagram View

model. This component represents an ideal model of a DC motor and it can be replaced later with a more complete realistic model.

Machines.Examples.DCMachines.DCPM_CurrentControlled

Each section of the MSL contains an example package, with models architectures to test

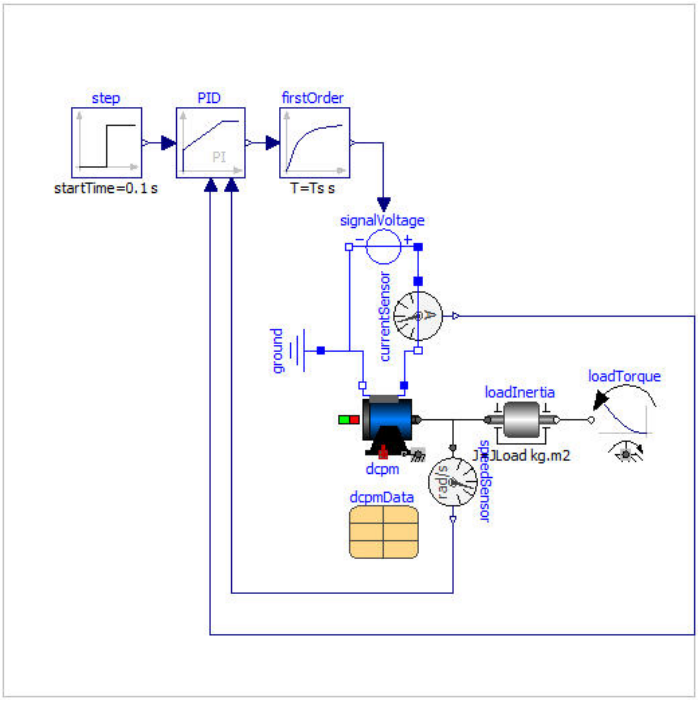


Figure 3.1.33 MSL Component - DCPM_CurrentControlled - Diagram View

various systems. This also applies to the electrical package, where examples of electric machines can be found. By examining the model presented here, you can get a better idea of how a permanent magnet DC motor works with current control. A dynamic model control unit based on this model was implemented for SP7. The *angleSensor* and *currentSensor* components are used as feedback for the PID controller and the *Step* source signal has been replaced with the desired joint position coming from the input trajectory file.

3.2 External C Functions

Each time a model is developed in Modelica, it must go through a series of processes before it can be simulated. Modelica model must be translated and systems of linear and non-linear equations are generated. Symbolic processing is performed in the first phase

of the translating process to obtain a "flat" model and after an optimization phase, the model is translated in C code and finally compiled. Fig. 3.2.34 represents an architectural overview of the aforementioned process and a detailed description can be found in [16]. Modelica allows developers to model their systems with a high degree of customization by

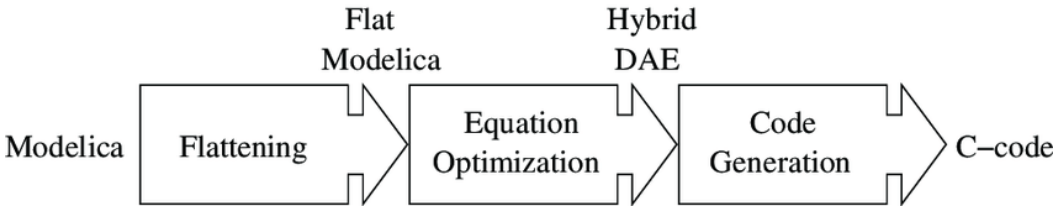


Figure 3.2.34 Modelica Models Translation Process

providing numerous open libraries and giving no limitation on the modification of these libraries. Another fundamental aspect of development with Modelica is the possibility of including functions not present in libraries within the model by developing external functions to the development environment and easily importing them into it. This is possible since, as previously explained, each model must be translated in C code and then compiled by a C compiler. A detailed guide can be found in [1]. OpenModelica provides support for:

- C89 and FORTAN 77;
- mapping of argument types;
- array handling.

The community have suggested two main ways of getting external C code compiled into the simulation: one is by separately compiling the code into an object file and then instruct the Modelica compiler to link against it; the second one is by telling the Modelica compiler to directly include your source as an input file. For both the cases, the format declaration of external functions in Modelica language is as follows: A simple example of what can be

Figure 3.2.35 Modelica External C Function Declaration

achieved with external functions is shown in the following sub section.

3.2.1 Example

Requirements: implement and external C function that takes an integer number n as input parameter and returns -1 in $n < 2$ or returns $n * 4$ otherwise. This behaviour is implemented with the following source code "*ext_func.c*":

```
-------------------------------------------------------------------------------
        int funcname(int n)
    {
        if (n < 2)
            return -1;
        else
            return n * 4;
    }
-------------------------------------------------------------------------------
```

The modelica model is structured as follows:

```
-------------------------------------------------------------------------------
    model test_external_c

    function ext_funcname
      input Integer n;
      output Integer result;
      external "C" result = funcname(n); // Must be the same
                                            name of the function
      annotation(Include = "#include \"C:/Path/To/Source/ext_func.c\"");
    end ext_funcname;

    output Integer a;
    equation
        a=if time<2 then ext_funcname(1) else ext_funcname(5);
    end test_external_c;
-------------------------------------------------------------------------------
```

The resulting behaviour of the model is shown in the following plot:

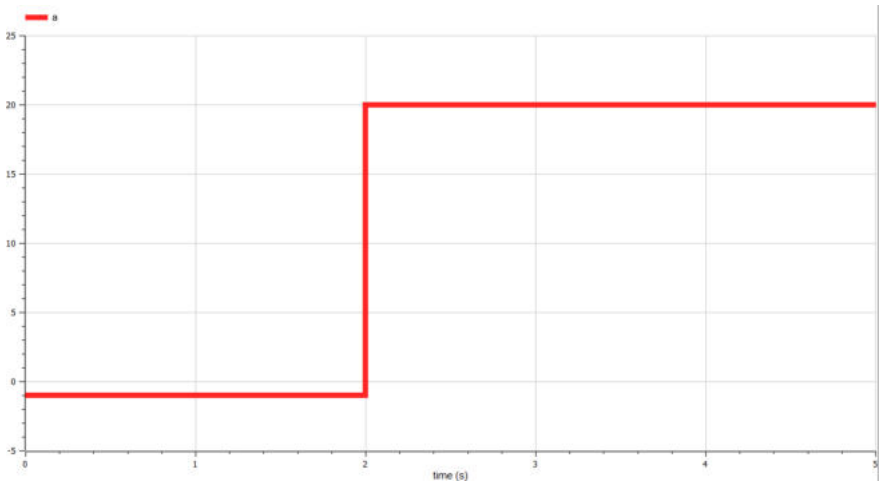


Figure 3.2.36 Modelica External C Function - Example Result Plot. As expected $a = -1$ for $time < 2$ and $a = 20$ for $time > 2$

3.3 System Architecture

Software Architecture represents the global view of software systems by abstracting out the complexities of low-level design and implementation details. The goal of the project is to obtain a simulation model that can be used for testing trajectories coming from independent works before actually feeding the platform with them. Therefore, it is necessary to implement the model in such a way as to be able to use the same input data for both the platform and the model itself. The SP7 can be controlled with the use of SP7App in three different ways:

- pre-established circular trajectory, tunable using scaling factors;
- using a space mouse as controller [11];
- using a trajectory from a file.

The last options is the one that is generally used. The file can be generated with any different procedure and the data in it must be written in the following format:

$$[\ timestamp\ \ x\ \ y\ \ z\ \ roll\ \ pitch\ \ yaw\ \ v_x\ \ v_y\ \ v_z\ \ \omega_x\ \ \omega_y\ \ \omega_z\ ]$$

A sequence of the so generated data represents the desired positions and velocity, w.r.t. the reference frame of the platform, in each time instant. Once the file has been created and the workspace boundaries have been checked, the platform is fed with the data throught the UDP socket. In run-time, the Inverse Kinematics algorithm implemented within the platform drivers, computes the joint position and velocity for each motor and the actuators

are then controlled. The goal of the system architecture is to replicate the behaviour just explained. Given that an Inverse Kinematics algorithm written in C++ was already existing, it was decided to develop the MM in order to accept as input the computed joint's trajectories. According to what just explained, the final System Architecture looks like this: where the

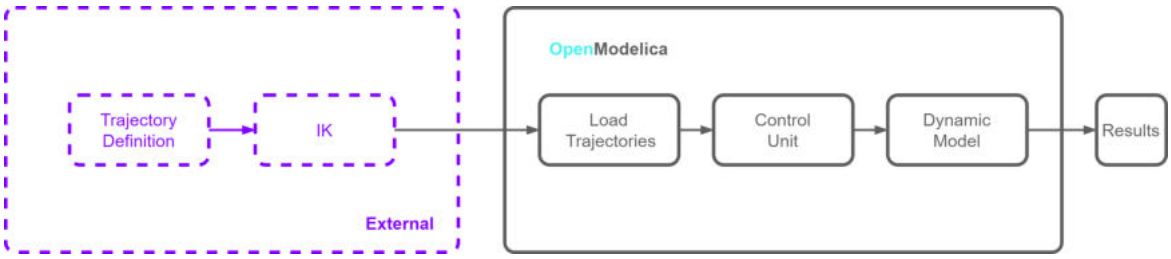


Figure 3.3.37 OpenModelica software Architecture

purple block represents the trajectory generation and the IK computations in order to feed the model with the platform position represented in joint space, and the right block represents the software architecture of the MM which receives the input data, loads them to send to the controller the set positions and finally drives the dynamic model with which the resulting data will finally be collected.

3.4 SP7 Model

In this sections the package organization of the SP7 model will be described. During the modeling of the different components and systems of the parallel robot, they were used several components already modeled and contained in the Open Source libraries available on OpenModelica. As we have already seen in the previous chapter, Modelica allows you to interface different components in order to obtain complex multi-physics systems. The models contained in the libraries are however fully editable once duplicated in a personal Modelica package. Finally, you can create new non-component models still contained in the libraries offered. During the development of the model of the motion platform, it was possible to refer only to the components found in the Modelica Standard Library, which we have extensively described in the Section 3.1. Including a component of a library within the model under development is needed to include this library and the path to access that component. In Fig. 3.4.38 we see two different ways to include a component: the first method includes a single component, the second includes all the components contained in the package.

```
import Modelica.Mechanics.MultiBody.Joints.Revolute;
import Modelica.Blocks.*;
```

Figure 3.4.38 Modelica Import Component - Code

3.4.1 SP7 dynamic model

The full model developed during this project is composed both of the mechanics and the control unit of the robot. In this sub section the mechanics of platform will be described, leaving the overview of the controller model for the following section. It was decided to use a software design aimed at object-oriented development and the structure of the parallel robot was divided into three blocks connected to each other namely *base*, *six_rss_legs* and *platform*. This was possible thanks to the Modelica feature of dividing a system into a hierarchical architecture using connectors to establish a connection between models. A detailed description of the developed block follows in the next paragraphs.

3.4.1.1 Base

Figure 3.4.39 Base model - Icon

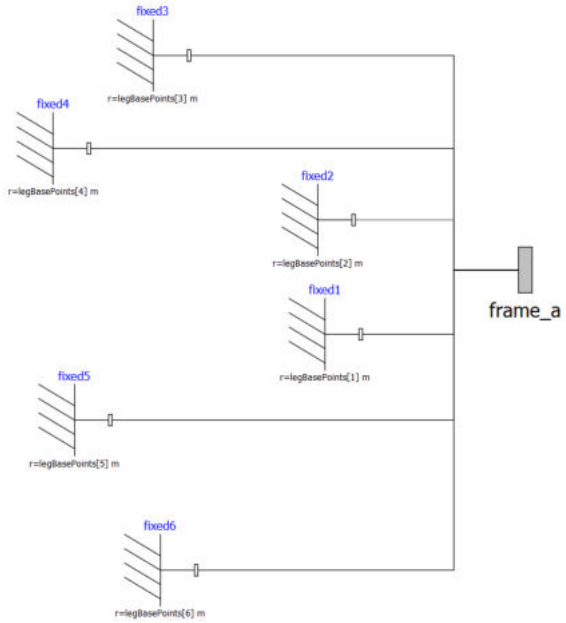


Figure 3.4.40 Base model - Diagram view

This component represents the positions of the six legs in the bottom plate of the mechanism and therefore the machine bed. The spatial position (x, y, z) are received as parameter

of the model from the array *legBasePoints[6,3]* and the exact value is hard-coded using the geometry of the robot. The value so obtained are used as position vectors for the *Fixed* components. *Frame_a* is used to connect the base to the *six_rss_legs* model. The parameters used for this model are shown in Fig. 3.4.40 where:

- *legBasePoints[6,3]* is the vector distance from the center of the base to the housing of each leg;
- *l* is the distance from the center of the platform to the connection point between platform and leg in polar coordinates w.r.t. the *x-y* axis;
- *alpha* is the position angle in polar coordinates of the motor pairs w.r.t. the *x-y* plane;
- *betha* is the positive/negative offset angle of each leg from alpha.

```
  parameter Real legBasePoints[6, 3] = {{l * cos(0 * alpha + betha ), l * sin(0 * alpha  + betha ),0},
{l * cos(1 * alpha + (-betha)), l * sin(1 * alpha + (-betha)),  0}, {l * cos(1 * alpha + betha), l *
sin(1 * alpha + betha),0}, {l * cos(2 * alpha + (-betha)), l * sin(2 * alpha + (-betha)),0}, {l * cos(2
* alpha + betha), l * sin(2 * alpha + betha),0},{l * cos(0 * alpha + (-betha) ), l * sin(0 * alpha  +
(-betha) ),0}};
  constant SIunits.Length l = 0.5735;
  parameter SIunits.Angle alpha = Modelica.SIunits.Conversions.from_deg(120);
  parameter SIunits.Angle betha = asin(60.0 / 540.0);
```

Figure 3.4.41 Base model - Parameters

3.4.1.2 Six_rss_legs

This model represents the second block of architecture. Each kinematic chain is in turn encapsulated within the *rss_leg* model that will be described soon. The *six_rss_legs* model is used to connect each leg to the respective base location with *Frame_a* and to the respective platform location using *Frame_b*. The torque and the angle generated in the Control Unit can drive the position of the revolute joint of each leg through *Flange_a*.

Figure 3.4.42 Six_rss_legs - Icon

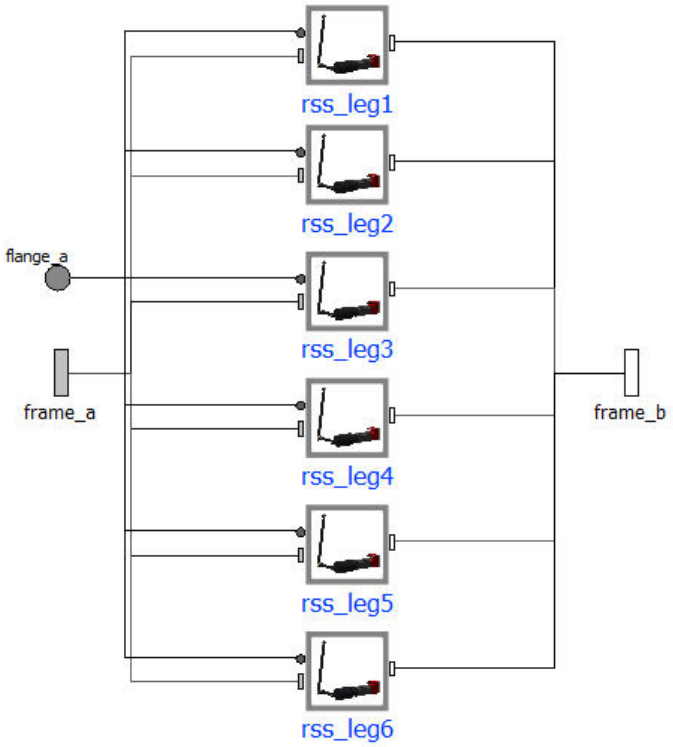


Figure 3.4.43 Six_rss_legs - Diagram View

3.4.1.3 Rss_leg

Getting a working leg model to successfully close multiple kinematics loops was the most time-consuming part of the project. As already explained, each leg of the SP7 is composed by a revolute joint driven by the actuator, a crank attached to the joint lying on a plane perpendicular to it and a spherical-spherical rod which close the kinematic chain with the platform. The first attempt was to model the leg using atomic components given by the MSL: in Fig. 3.4.44 it is shown the first model implemented for the RSS leg.

Figure 3.4.44 Rss_leg First Implemented Model - Diagram View

Now, one of the downsides of the acausal programming approach of a system had to be faced: in fact, even if at first sight the model seems to be correct, the use of these components in this specific configuration for the development of parallel closed kinematic chains resulted in a redundant and overdetermined system. Farther more, the translation of the model and generation of the equations that govern the system takes place in the backend of the simulation environment, thus it was impossible to attempt a debug to understand what the breaking point of the model was. A similar situation was encountered during the

modeling phase of the best known planar four-bar closed-chain linkage: if a planar loop is present, e.g., consisting of 4 revolute joints where the joint axes are all parallel to each other, then there is no unique mathematical solution if all revolute joints are modelled with *Joints.Revolute* and the symbolic algorithms will fail. The reason is that, e.g., the cut-forces in the revolute joints perpendicular to the planar loop are not uniquely defined when 3-dim. descriptions of revolute joints are used. Usually, an error message will be printed pointing out this situation. In this case, one revolute joint in the loop, namely *j2*, has to be replaced by the model *Joints.RevolutePlanarLoopCutJoint* as shown in Fig. 3.4.45. The effect is that

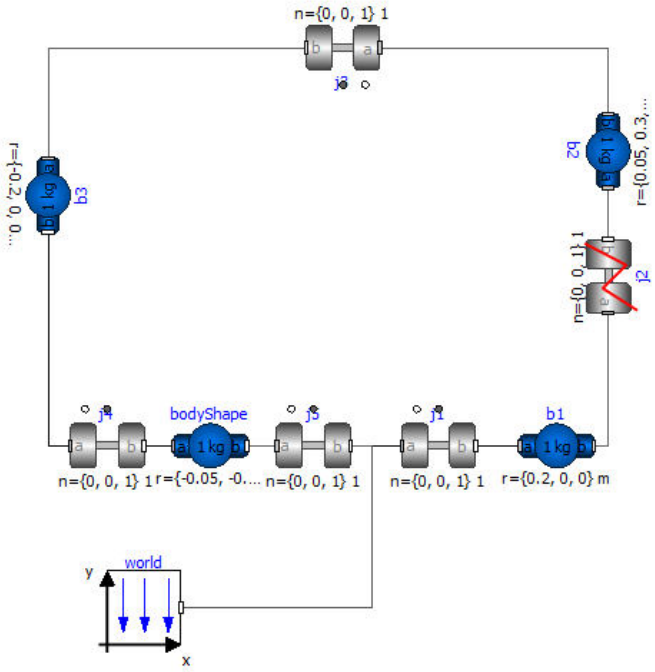


Figure 3.4.45 Fourbar Closed Loop Model - Diagram View

from the 5 constraints of a 3-dim. revolute joint, 3 constraints are removed and replaced by appropriate known variables (e.g., the force in the direction of the axis of rotation is treated as known with value equal to zero; for standard revolute joints, this force is an unknown quantity). The declared variables remove the redundant equations and the system becomes finally solvable. It soon became clear that the leg model had to be modified to eliminate the surplus equations. To do this, it was decided to replace the Spherical-bodyShape-Spherical sequence with the *SphericalSpherical* component as recommended in the model information box of the latter, since this enhances the efficiency considerably due to smaller systems of non-linear algebraic equations. Indeed, differently form *Spherical* joint that introduces 3 potential states, this joint component introduces one constraint (defining that the distance

between the origin of *frame_a* of the model and the origin of *frame_b* is constant) and no potential states. The number of potential states in a model defines the amount of equations of the system, therefore, using *SphericalSpherical* resulted in deleting the redundant equations of the system. The final implementation of the *rss_leg* is shown in Fig. 3.4.46. As for the base, the geometry of each component, i.e. the direction of the revolute joint axis of rotation and the vector *r* defining the crank model, is hardcoded starting from the geometry of the robot. The parameter of this model are therefore the following:

- *legPositionConfig[2]* is a structure indicating the motor pair with the first value and the right/left leg with the second value;
- *alpha* same as base model;
- *h* is the length of the crank represented in meter.

The revolute joint can be driven through the *flange_a* since the *useAxisFlange* flag is set to true.

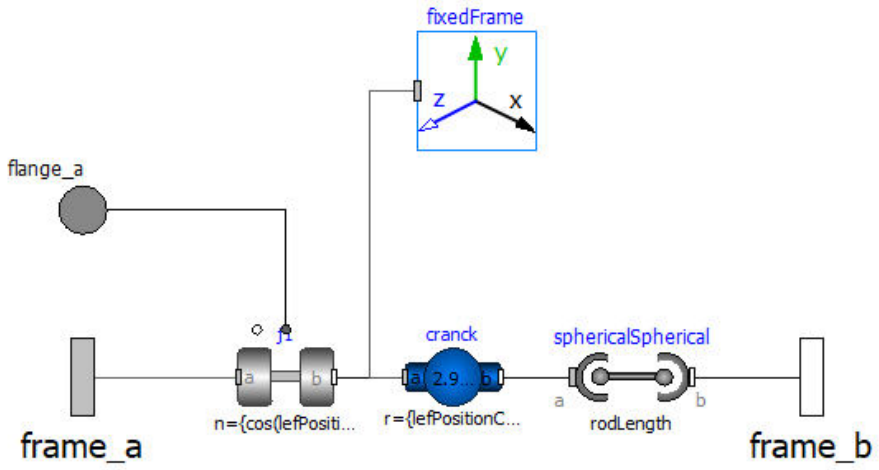


Figure 3.4.46 Rss_leg Final Implemented Model - Diagram View

3.4.1.4 Platform

As we can see from Fig. 3.4.48 the Platform component consists of a central body of the *BodyShape* type, which will define the geometric and mass properties of the platform. Six FixedTranslation components will be correctly oriented to position the interfaces that will

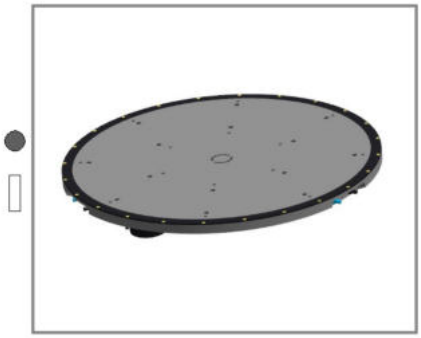


Figure 3.4.47 Platform - Icon

house the legs of the mechanism using *legPlatformRelativePositions* parameter. The revolute joint *j_vertical* is controlled using *flange_a* input with the torques and angles generated by the control unit. The parameters used for this model are:

- *legPlatformRelativePositions[6,3]* is the vector distance from the center of the platform to the housing of each leg;
- *l* is the distance from the center of the platform to the connection point between platform and leg in polar coordinates w.r.t. the *x*-*y* axis;
- *alpha* same as base model;
- *betha* same as base model.

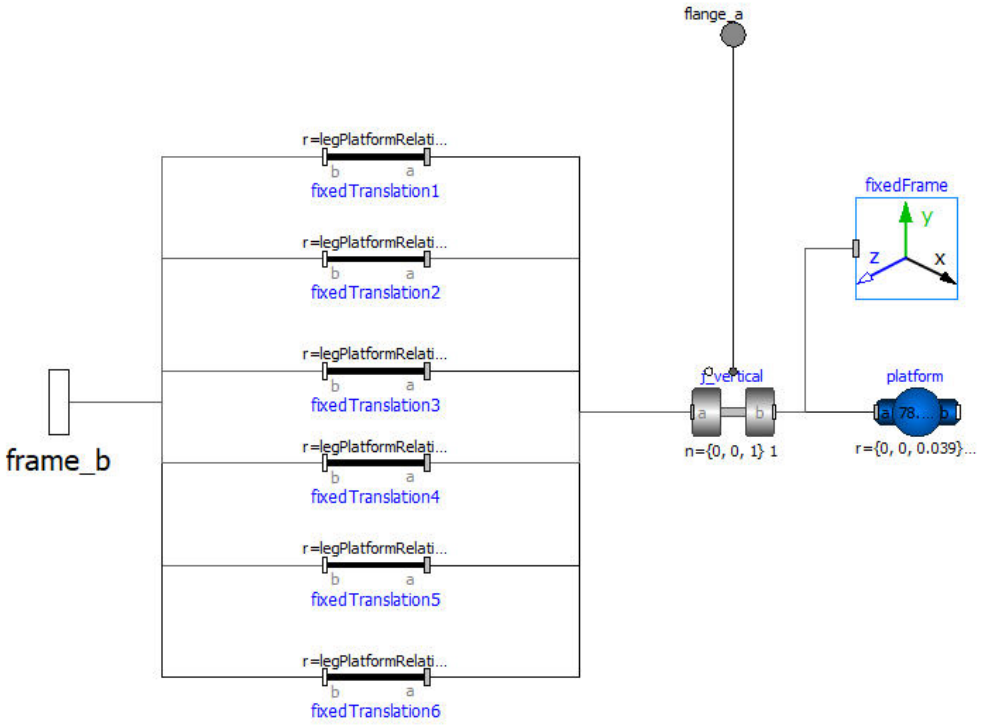


Figure 3.4.48 Platform - Diagram View

3.4.1.5 Six_rss_closedloop

We can see in Fig. 3.4.49 that the dynamic model is basically composed of the three elements connected to each other, i.e. the base (*Base*), the legs (*Six_rss_legs*) and the platform (*Platform*). Through the *flange_a* input it is possible to attribute forces to the legs and to the vertical motor positioned under the platform and observe the movements of the mechanism through a correct analysis of the output file of the simulation. The dynamic model is finally developed but a correct implementation of the Control Unit is needed and will be the described in the next section.

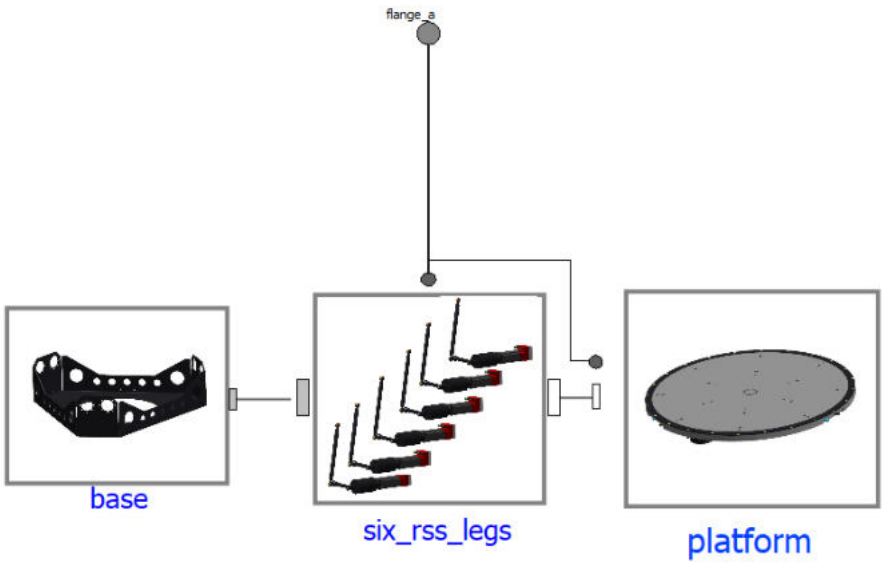


Figure 3.4.49 Six_Rss_ClosedLoop Final Implemented Model - Diagram View

3.4.2 Simulation Control Unit

It has been shown in Section 3.3 that the MM receives as input the trajectory of the platform in joint space (rad), i.e. the results of the Inverse Kinematics through the whole trajectory. At this point, it was necessary to model a Control Unit in order to drive the legs through the requested motion. Among the multiple possibilities for implementing a controller of this kind, it was decided to replicate the system already used in the real robot, and therefore one components was developed to model a tunable PID and generate a torque using the *DC_PermanentMagnet* model described in Section 3.1.1.2.

In Fig. 3.4.50 it is visible the diagram view of the Control Unit: *jointSetPosition* is a RealInput array of dimension seven and receives the joint position values from the outer *CombiTimeTable* component. *DCPositionAxis* is the model encapsulating the PID and the servomotor model. *Flange_b* is an array of dimension seven and connect the torques and angles resulting from the single controller with the respective revolute joint of the dynamic model of the robot. The use of gains is required since the angles computed for each pair of motor has the same sign, as shown in [15], e.g. a desired positive motion along the z axis would generate positive angles for any of the six legs. This would result in a correct movement if and only if the motors in the pairs rotated in the opposite direction, one clockwise and the other counterclockwise. Since in the implemented model the axis of rotation of the joins is always directed outwards, it becomes necessary a $+1/-1$ gain to invert the signs of three of

the six angles calculated by IK. Gains are tunable and their value depends on the input that is used for the model: it follow the mapping of gains value for this project.

Table 3.5 Control Unit - Gains value mapping

	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5	Leg 6
Gain value	1	-1	1	-1	1	-1

3.4.2.1 DCPositionAxis

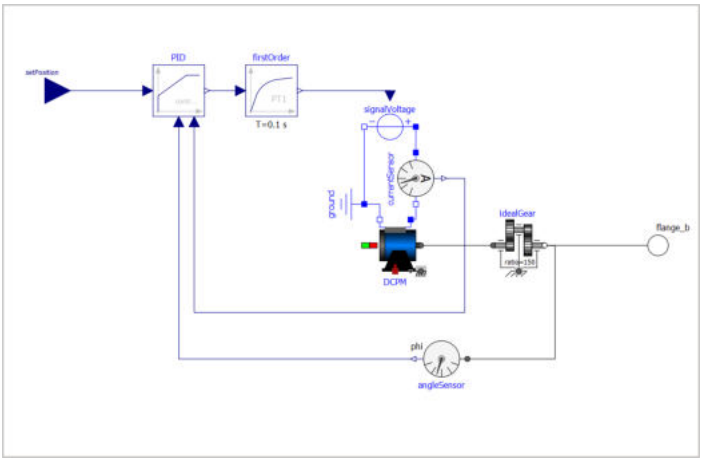


Figure 3.4.50 DC_PositionAxis - Diagram View

In order to develop this block of the model, it has been followed the *DCPM_CurrentControlled* model found in the examples of the *Electrical.Machines* package. *setPosition* represents the set point value i.e. the desired angle of the actuator for each time instant. *LimPID* is a controller with limited output, anti-windup compensation, setpoint weighting and optional feed-forward and takes as input *setPosition*. With the use of *currentSensor* and *angleSensor* it is possible to compose the feedback of the system. A detailed description of the PID parameter calibration will follow in Section 4. The specifics of the motors are directly taken from [13] and [12] and are summarized in the Table 3.6. In the following table it is possible to see the gearbox ratio used in this model.

Table 3.6 Nominal parameter for MP DUET Flexi 80 2.0

		MP DUET Flexi 80 2.0	SPINEA DS 110-119
Nominal Voltage	[V]	48	48
Nominal Current	[A]	14.4	4.9
Nominal Temperature	[C°]	40	40
nominalRPM	[rev/min]	2500	3000
armatureResistance	[Ω]	0.12	1.4
armatureInductance	[!htbp]	0.00031	0.0074

Table 3.7 Control Unit - Gearbox Ratio

GearBox	
	Ratio
Horizontal motor	150
Vertical motor	119

3.4.3 Full Model Implementation

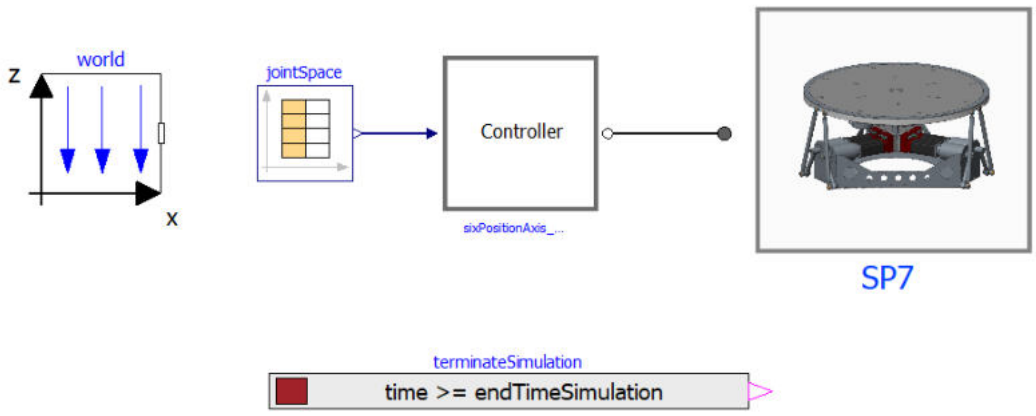


Figure 3.4.51 Full Modelica Model Final Implemented Model - Diagram View

This is the final implementation of the system architecture described in 3.3, where the developed sub models are connected. *World* component is used to define an inertial coordinate

system and an uniform gravity field pointing downwards. *jointSpace* is a *CombiTimeTable* used to load the position vector and timestamp of the platform trajectory from an external file: the output of this component is a *RealOutput* vector of dimension seven with the actuator positions mapped in the following way:

$$[\ l_1\ \ l_2\ \ l_3\ \ l_4\ \ l_5\ \ l_6\ \ j\_vertical\ ]$$

terminateSimultion is used to stops the simulation whenever the time represented by the last timestamp value of the file has passed (Fig. 3.4.52); this resulted really helpful in the synchronization of results data during the test phase.

```
Real endTimeSimulation = jointSpace.t_maxScaled;
Modelica.Blocks.Logical.TerminateSimulation terminateSimulation(condition = time >= endTimeSimulation)
```

Figure 3.4.52 Full Model - terminateSimulation code

3.5 UDP Socket

Modelica is a powerful programming language and libraries of all kind has been developed to implement a whole variety of cyber-physical systems. Both offline or online simulations can be run in Modelica and this gives the user a large set of possibilities. Run-time simulation can be achieved by working on the synchronization of the simulation time and implementing if necessary a real-time communication behaviour in order to exchange information with agents external to the Modelica development environment. These features can be implemented either by using Modelica libraries or by implementing custom source code as it will be described below. *Modelica_DeviceDrivers* [17] library is a Modelica package which add a whole set of components to interface external devices as keyboard, mouse or joystick controllers with the model that you are working with. The library also has components to synchronize the simulation time (which depending on the implemented model can be really short compared to the real-time) and to setup socket communications. *SynchronizeRealtime* is a component from *Modelica_DeviceDrivers.Blocks.OperatingSystem* package that can synchronize the simulation time of the simulation process with the operating system real-time clock. The behaviour of this component is described as "soft" real-time synchronization and indeed it is highly not recommended in any safety relevant application where precise timing is mandatory [8]. The library also provides a set of components to setup a UDP socket communication: after some initial tests, the communication resulted to be limited and poorly

customizable, so it was necessary to find a different solution. To overcame these limitations, an external C function was developed in order to establish a fast and fully customizable socket communication. A detailed overview on how to develop and implements an external function can be found in Section 3.2.

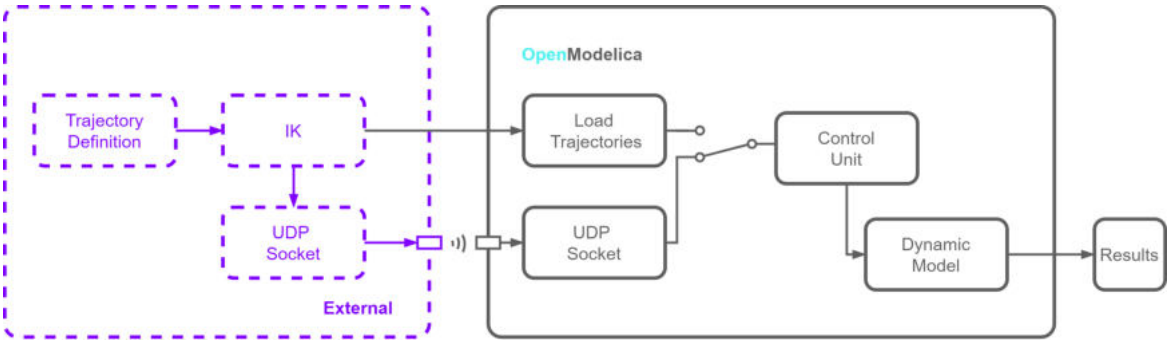


Figure 3.5.53 Software Architecture with UDP Socket

3.6 Windows Application

In Section 3.2 it is also shown how an executable simulation program is created by translating high-level MMs into a flat set of hybrid differential-algebraic equations. This translation process is highly complex, involving multiple phases and analyses. Using the application generated once the model has been built, it is possible to simulate different trajectories without using the OpenModelica tool set (including OpenModelicaEditor). Detailed instructions on how to use the application will follow. Anytime a model is built within OMEdit, these files are generated inside the temporary system folder
"C:\\UserName\AppData\Local\Temp\OpenModelica\OMEdit\ModelName":

- ModelName.exe
- ModelName_prof.intdata
- ModelName_info.json
- ModelName_prof.realdata
- ModelName.log
- ModelName_init.xml
- ModelName_visual.xml

In order to run the application, it is necessary to link the needed windows dynamic libraries with the executable: in fact, by running the *.exe* file, a window containing a *"Missing libraries error"* will pop up. All the necessary libraries can be found in the setup directory of OpenModelica, e.g. *"C:\Program Files\OpenModelica<version>\bin"* and need to be moved in the executable directory to successfully run the simulation. From now on, it is possible to deliver the application and run it in a OpenModelica-free Windows machine. In the "ModelName_init.xml" file it is possible to configure the following info:

Table 3.8 Modelica configuration variables

StartTime	Simulation starting time
StopTime	Simulation stopping time
OutputFormat	Supported extensions for output fire are "csv", "mat" and "plt"
StepSize	Simulation time resolution
Tolerance	Tolerance value for DAE solver
Solver	ODE solver [10]
VariableFilter	Regular expression used to filter variables from the result file [3, 18]. In MBS become useful filtering variables since the output file dimension increase drastically. Example: `VariableFilter = "varA.phi|varB.phi"` for varA and varB angles.

Chapter 4

Testing Phase and Results

The software architectures involved in the project are two: one describing the control procedure of the motion platform, and the second one describing tools and dataflow for the simulation model. For what concerns the real platform, Matlab is used to generate a trajectory within the workspace boundaries and at a framerate of 120 Hz, and the needed velocity are then numerically computed from the so obtained trajectory. For each motion in the space, velocities are calculated using numerical differentiation with the assumption of an enough small dt $(8,3ms)$, where $dt = t_{i+1} - t_i$, in the following way:

$$f'(x) = \frac{f(x+h) - f(x)}{h} = \frac{f(x_{t_{i+1}}) - f(x_{t_i})}{t_{i+1} - t_i} = vel$$

The resulting input file has the following format:

$$[\ timestamp\ \ x\ \ y\ \ z\ \ roll\ \ pitch\ \ yaw\ \ v_x\ \ v_y\ \ v_z\ \ \omega_x\ \ \omega_y\ \ \omega_z\ ]$$

where:

- $x, y, z, roll, pitch, yaw$ represent positions and orientations of the platform;
- $v_x, v_y, v_z, \omega_x, \omega_y, \omega_z$ represent linear and angular velocities of the platform.

Via the Ethernet and UDP socket connections, it is possible to send the generated desired Cartesian trajectory to the platform using the SP7 controller application. The controller of the SP7 receives the Cartesian pose and the velocity of the end-effector, which are then processed into the joint angles with the IK implemented in C++. To control the actuators, the joint angles are then processed via the PID controller and driven by a PWM signal. After each simulation, a log file containing the positions of the joints is generated through the

records of the encoders mounted in the motors. As we will see soon, the log of each motion will be processed to generate the inputs data of the Modelica dynamic model.

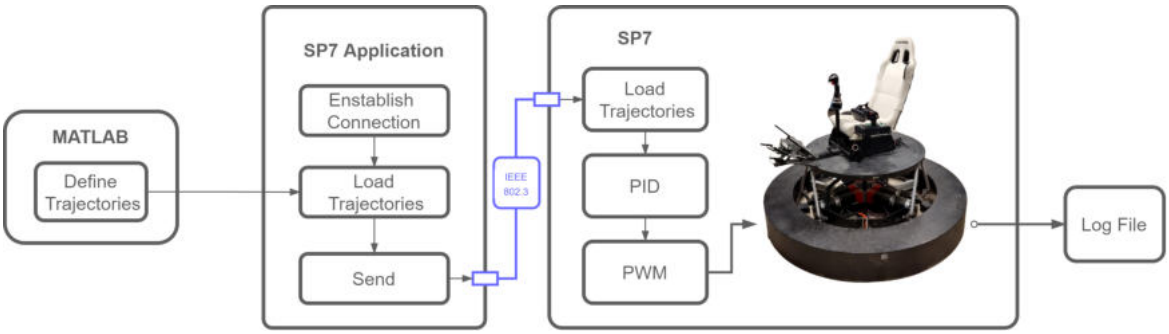


Figure 4.0.1 SP7 Control Architecture

As regards the simulation model, the system architecture has found several changes compared to the initial definition: initially, the idea was to decouple the platform and the model as much as possible to better accredit the validation phase of the model itself. To do this, an intermediate block of Inverse Kinematics was developed between the trajectory generation block and the OM model using pre-existing C ++ source code that should have replicated the one used by the platform control module: in this way, the only common block between the two architectures would have been the one that generates the trajectories. During the development phase, an architectural modification was necessary as the IK block did not work as expected. Since a fully working IK code fells beyond the scope of this research, it was decided to eliminate it. As a result, the final architecture became the one shown in Fig. 4.0.2. As previously mentioned, each time the platform is controlled with the use of the SP7Application, the positions of the seven joints during all the motion are recorded with the use of the encoders in a log file. This data will be eventually processed in order to produce the corresponding input joint trajectories for the MM.

The experiments for this project involve the use of two main equipment: the SP7 motion platform will be used as ground truth system, the MM is the component under examination. In Fig. 4.0.2 it is illustrated the experimental architecture that is been developed for this project. All the set of experiments will be taken offline and the purpose of the test is to define a certain amount of desired trajectories for the SP7 and then compare the platform motions with the model performances to assess the accuracy and precision of the model by comparing the joint positions during all the motions. The results of the simulation will be analysed through statistical analysis test in order to give a statistical proof of the capabilities of the model.

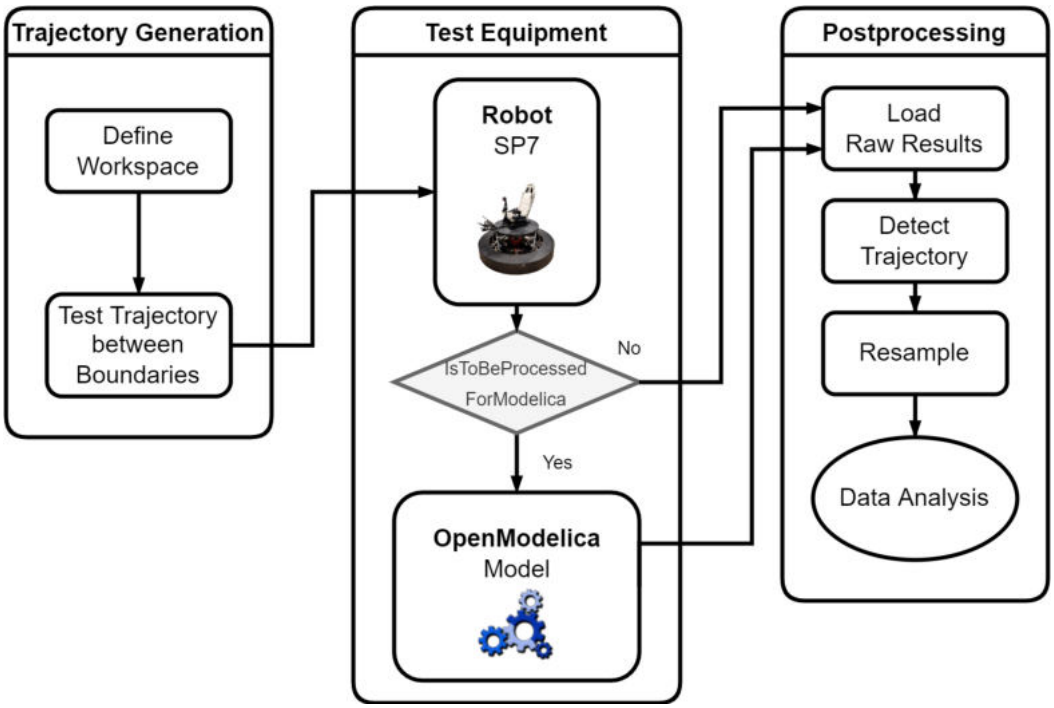


Figure 4.0.2 SP7 Experimental Architecture

The first block of the architecture represents the generation of the tested trajectories: using Matlab, the workspace boundaries are initially loaded and a structured algorithm is used to develop one-axis, two-axis or three-axis combined tragectories. An axample of trajectory could be a combined three-axis motion with translations along x axis, y axis and z axis between minimum and maximum boundary value of each direction. The second block represents the devices and the software involved during the experiments:

- **SP7**: data is read from the encoders of the motors and it is stored in a log file saved by the SP7 control Application;
- **Modelica**: each run of the model simulation generates an output file in *csv* format with joint positions and platform reference frame poses over time.

The postprocessing of the collected data is done within the Matlab environment: for each subsystem, the data relative to the trajectories are extrapolated and the sequence are resampled to synchronize the motions during the simulation. A better description of the Matlab logic will be presented in the Subsection 4.3.

4.1 Calibration of Model Parameters

In Section 3 the model design has been extensively described in each of its components and a complete description was given. There was explained how it was possible to obtain a working dynamic model and a detailed description of all the possible parameters involved in it was given. Developing a model requires the input of certain information, such as the geometry of the platform or the specifications of the actuators, which belong to a whole set of fixed parameters of the model. Once the mentioned parameters has been implemented, what remains to do is to define all those set of variables in the model that have to go through a tuning process before conducting the real experiments. In this project, most of the parameters were fixed and what remains to be tuned were the PID gains: in order to do this, starting from the model shown in Fig. 3.1.33, the model shown in the following figure was implemented.

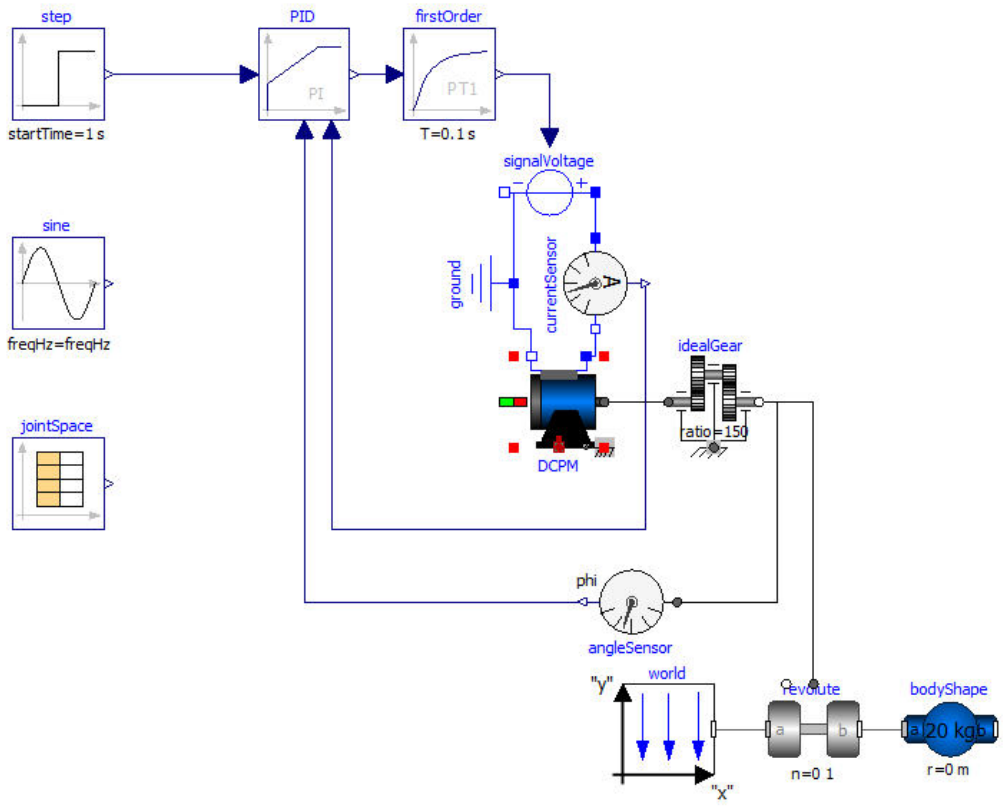


Figure 4.1.3 DC Motor PID Calibration Model - Diagram View

The models implements a simplified version of the architecture of the parallel robot, where each leg can be considered as a unique system composed of PID, actuator and load.

The reference set positions used in the process were sequentially updated in order to find the best fitting parameters for PID: the first set of tuning was carried out having a Step Source component as *setPosition*; once suitable gain values were selected, it has been used a sinusoidal source signal and finally a sample of real input data from SP7 recordings for the final evaluation. The sizing of the gains of a PID is an extremely discussed matter and for our case, we will refer to what is indicated in [22]. The PID component implemented in the model is described in Section 3.1.1.1 and its behaviour is described by the formula:

$$u(t) = K_P\left(e(t) + \frac{1}{T_1}\cdot\int_0^t e(\tau)\,d\tau + T_D\cdot\frac{de(t)}{dt}\right)$$

Standard tuning operation for PIDs is to first select a proportional gain value that allows us to achieve good controller reactivity without having to deal with too high overshooting values. In Fig. 4.1.4 it is shown how different values of K_P affect the reactiveness of our model for the leg's actuators: it is possible to see that for our application, by increasing the gain value the performances improve. Once a satisfactory value for the proportional gain has been chosen, the next step will be to select the best value for the integral part T_I. Fig. 4.1.5 shows how different values affects the response of the controller for the leg's actuators.

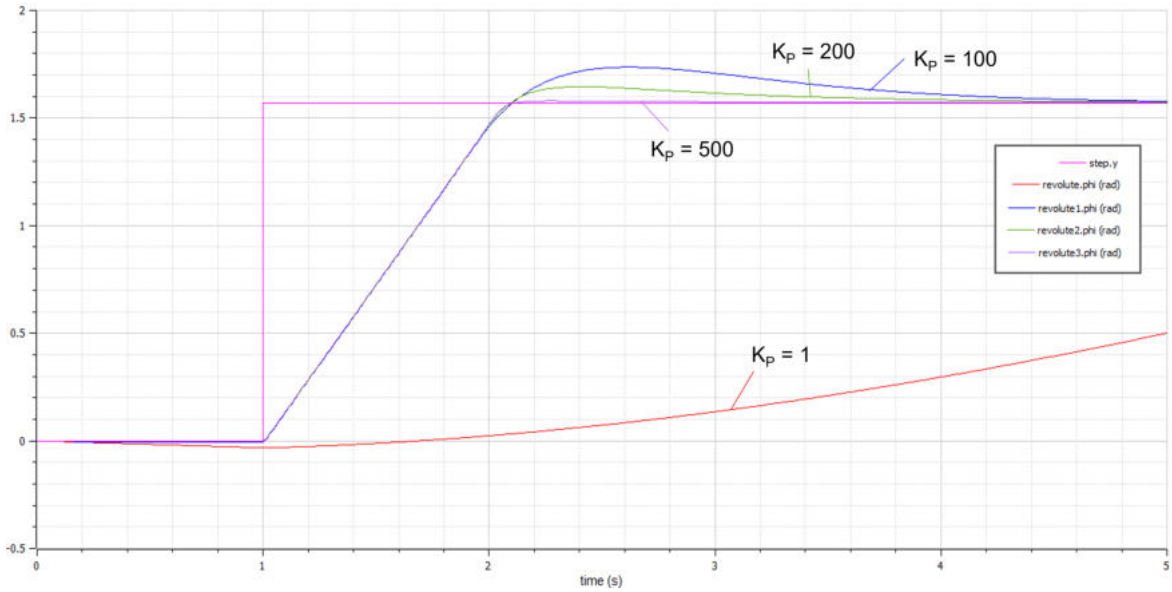


Figure 4.1.4 PID Tuning - Proportional Gain

The same procedure has been followed also for the vertical actuator and in the following table it is possible to see the final parameter used in this model for the PID controllers.

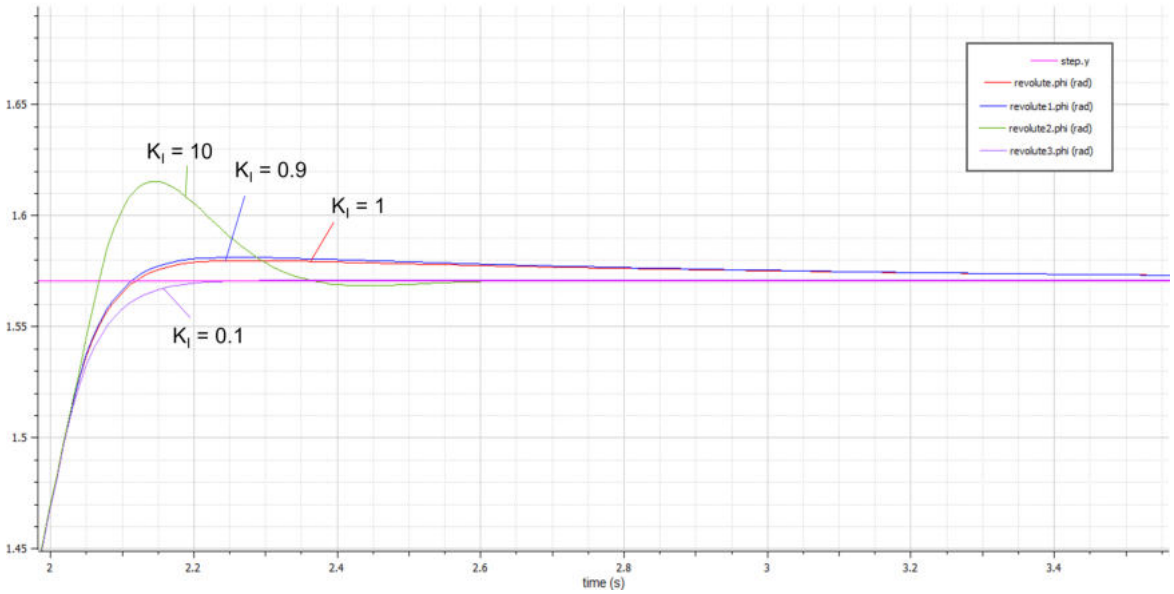


Figure 4.1.5 PID Tuning - Integral Gain

Table 4.1 Control PID parameter - Control Unit

	PID				
	PID type	P	T_i	T_d	Max Current [A]
Horizontal motor	PI	500	0.1	N.A.	39.8
Vertical motor	PID	300	10	0.001	17

4.2 Signal Flagging

As explained in Section 4, a set of Matlab functions and scripts have been developed to postprocess the data coming from the two systems namely SP7 and MM. Since the data comes with independent timestamps and time rates, it is necessary to process them in order to obtain a final set of records with a synchronized timestamp and a common resampling rate. Due to the offline testing proceedings, another relevant issue arises, that is finding a robust way to uniquely and repeatedly detect the relevant trajectory among the whole set of data recorded by the systems. In order to solve this, it was decided to use the same platform motions to append at the beginning and at the end of each input trajectory a motion flag that

encompasses the trajectory itself. According to this, it has been selected a set of two motions along a unique axis to be used as head flag and tail flag in the following way:

- **Head Flag**: A pure translations along the z axis that starts from the ZeroPose of the platform, reaches the lower boundary in 1 s, goes back to the ZeroPose in 1 s and stays there for 2 s;

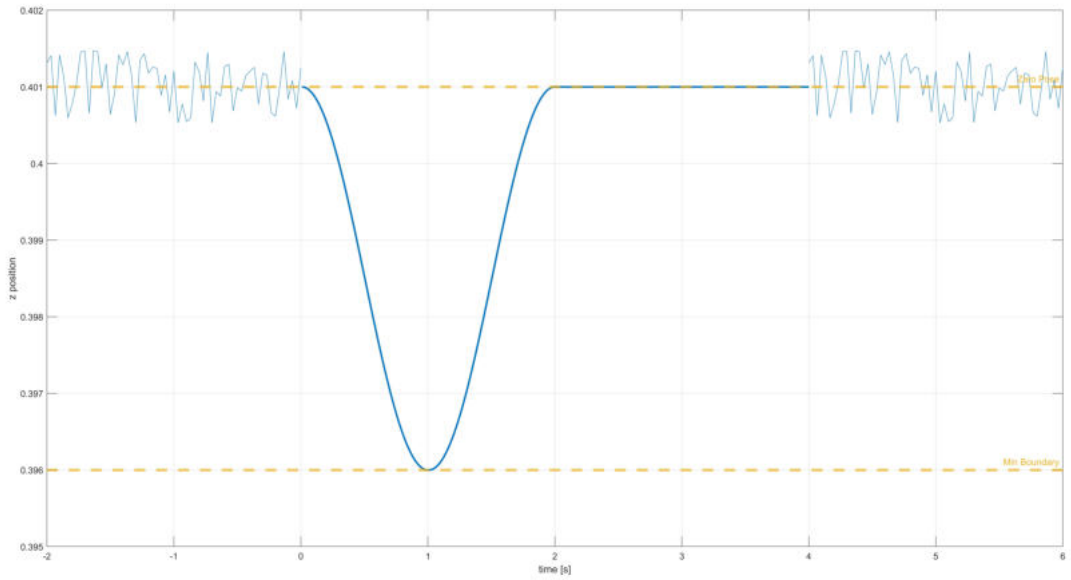


Figure 4.2.6 Trajectory Flagging - Head Flag

- **Tail Flag**: A pure translations along the z axis that starts from the ZeroPose of the platform, stays there for 2 s, reaches the higher boundary in 1 s and goes back to the ZeroPose in 1 s;

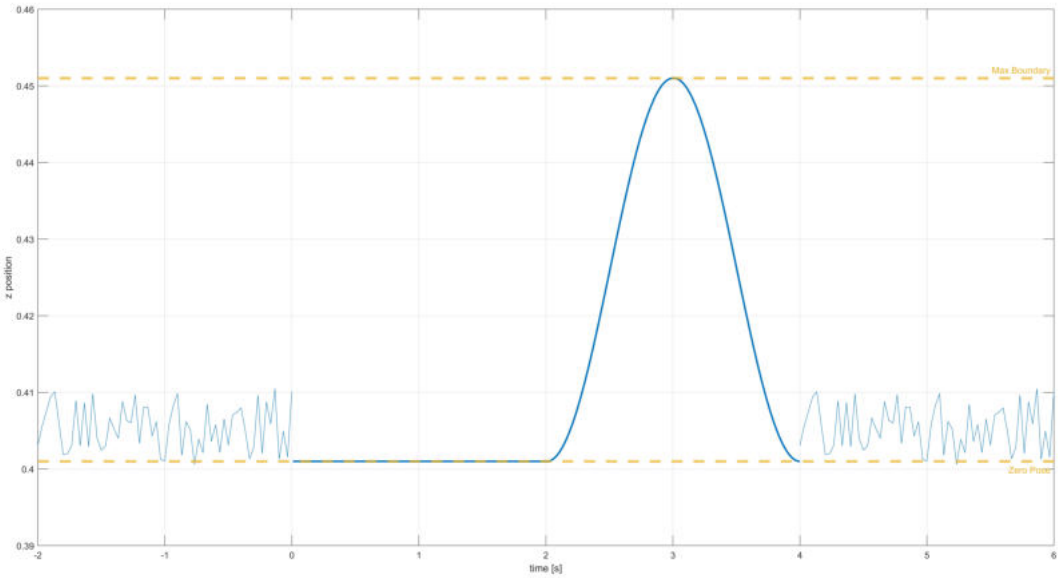


Figure 4.2.7 Trajectory Flagging - Tail Flag

Using a pure translation along one single axis results on a fast and easy solution for the detecting phase of the trajectory. Detection was conducted by smoothing curve generated by the recorded data and finding local maxima and minima in the resulting curve (smoothing is a destructive process with a consistent loss of information, but for what concerns the flags position identification, such a loss can be accepted; the raw data will still be the ones used for the final test analysis).

4.3 Data Postprocessing

Each of the two systems collect the data in different formats, so the description of the program logic will be given separately. The SP7 motion platform is controlled through the SP7 Application as explained in Section 3.3, and each time a simulation is run a log file is generated: the procedure to use the application is the following:

1. Enstablish the connection between the application and the platform through the UDP socket by selecting the correct IP and by pressing **Connect**;
2. Select the preferred input configuration;
3. If file, browse through the file manager and select the desired input file;
4. Press **Start Feeder** to launch the simulation;

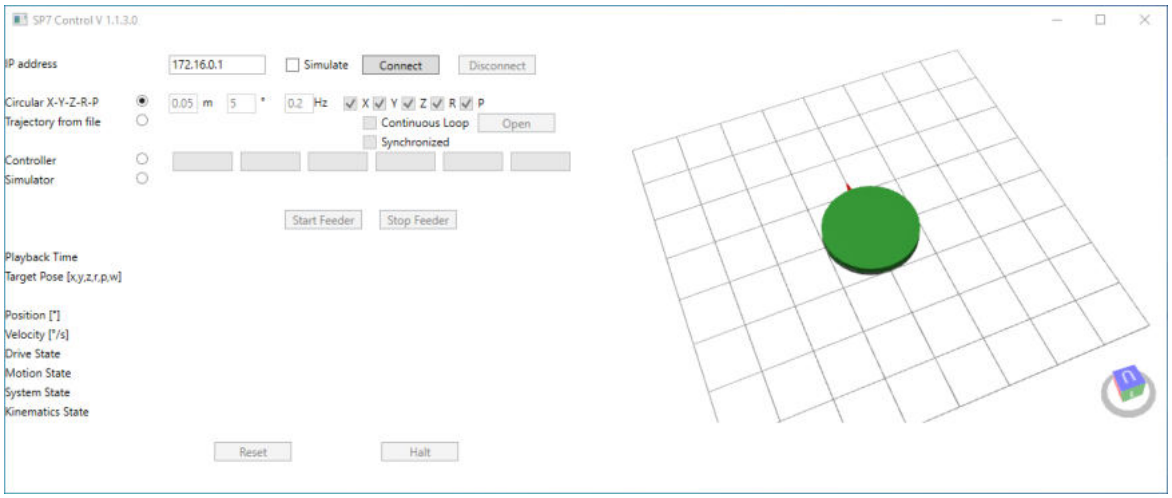


Figure 4.3.8 SP7 Control Application User Interface

5. Once the simulation ends, the application automatically disconnects;

6. To launch a new simulation, repeat from Step 1.

The data is recorded starting with the initial connection to the socket and ending with the final disconnection. Therefore, a whole set of irrelevant data are saved between Step 1 and Step 4. Signal flagging supports the removing process of these unwanted data by detecting the required trajectory regardless of what has been recorded before and after it. So, once the desired information has been extrapolated from the log file, this is resampled at $60Hz$ and the output of this process is a structure with the following fields:

- **t** [# timesteps x 1] : it is a column vector with the processed timestamp values;
- **j** [# timesteps x 7] : joint trajectories;

The same strategy is used to process the data coming from the MM simulation, but a different code is developed since the output file is in the *csv* extension and measurements units are different. Once the trajectory has been extrapolated, data is resampled at $60Hz$ and the output structure is:

- **t** [# timesteps x 1] : it is a column vector with the processed timestamp values;
- **j** [# timesteps x 7] : joint trajectories;

Once the raw data has been processed, it will be possible to compare them and give a statistical evaluation on the accuracy analysis and precision analysis of the experiment carried out. The following section will describe the design of the experiments of each set of test and validate the developed model accordingly to them.

4.4 Accuracy Analysis

4.4.1 Design of Experiments

In system modeling analysis, accuracy represents the capability of a model to simulate a certain behaviour with the closest results to the real system it is replicating. In the Robotics field, the position accuracy of a system, whether it is a serial manipulator, a parallel robot or a mobile robot, has always been center of continuous studies and analysis. In this project, it was therefore decided to conduct a statistical analysis on the study of the accuracy of the model to obtain a systematic and uncertainty-free validation. In order to do this, two statistical hypothesis tests AET and MPET were conducted. With the former, MM was tested to determine if the average accuracy was lower than 0.5 degree for the joint positions. The latter measures whether the maximum deviation from the mean is less than 2 degrees for the joint positions. The null and alternative hypothesis are defined as shown in the Table 4.2.

Table 4.2 Accuracy Analysis - Hypothesis test

Test	**Null Hypothesis**	**Alternative Hypothesis**
AET	$H_0 : \bar{d} \leqslant \delta_{avg}$	$H_a : \bar{d} > \delta_{avg}$
MPET	$H_0 : d_{max} \leqslant \delta_{max}$	$H_a : d_{max} > \delta_{avg}$

To ensure that the average error follows a normal distribution according to the central limit theorem, a trajectory sample of size $N > 30$ has been selected. Z-test has been chosen as hypothesis test since the sample distribution is unknown and for our application, collecting a set of over 30 trajectory is not extremely time-consuming and resource-intensive. As described in Section 4, combined axis trajectories can be generated using Matlab: in particular a set of 41 trajectories (sum of maximum combinations of $1/2/3$ objects with sample size of 6) as been developed.

Table 4.3 Trajectory combinations

Sample Size	**Set of Objects**	**# of Combinations**	**Sum**
6	Single-axis (1 obj)	6	41
	Two-axis (2 obj)	15	
	Three-axis (3 obj)	20	

In Fig. 4.4.9 it is possible to see the joint trajectories of one example desired trajectory generated in Matlab: the figure show how the platform should move during the execution of a combined motion of a translation along x axis, a rotation along x axis and a rotation along z axis. In a statistical sense, the precision is less than δ_{avg} if the AET accepts the null hypothesis with a $p-value > 0.95$ i.e., if the following equation is not satisfied:

$$\frac{\bar{d}-\delta_{avg}}{\sqrt{s^2/N}} > Z_\alpha \tag{4.1}$$

where:

- $\bar{d}$ mean error of the sample;
- δ_{avg}: null hypothesis average error;
- s^2: standard deviation of the sample;
- N: number of samples;
- $Z_\alpha = 1.6449$ (Z score at 0.95).

For MPET, the null hypothesis i.e., the maximum deviation is less than a certain threshold if the following equation is false for a significance value of α.

$$\frac{\delta_{max}-d_{max}}{d_{max}-d_{min}} < \frac{\alpha}{1-\alpha} \tag{4.2}$$

where:

- δ_{max}: null hypothesis maximum error;
- d_{max}: maximum error of the sample;
- d_{min}: minimum error of the sample;
- $\alpha = 0.05$.

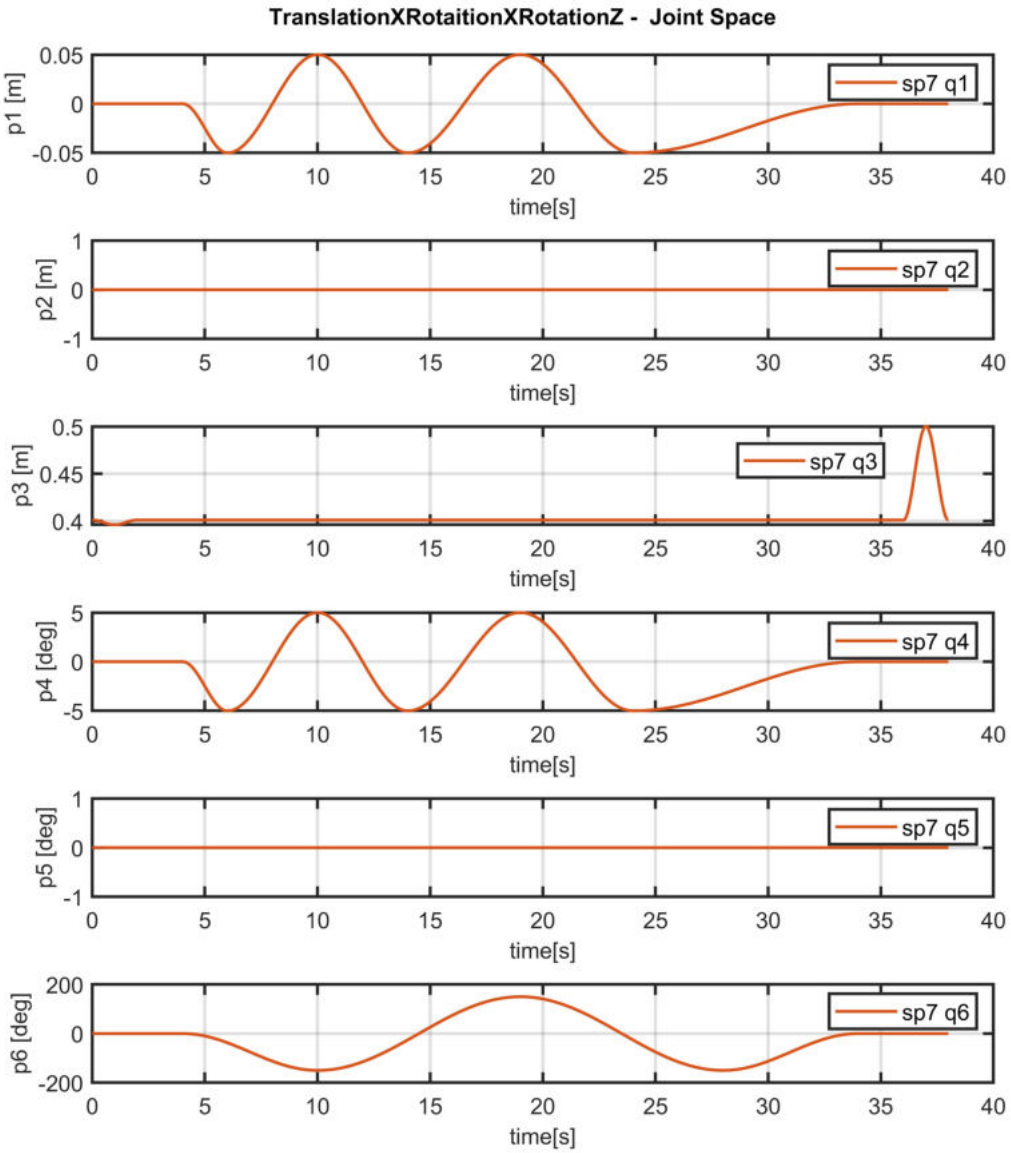


Figure 4.4.9 SP7 Platform Desired Trajectory - Cartesian Space: The plots show an example of trajectory namely *TransaltionXRotationXRotationZ* generated with Matlab. [p1, p2, p3, p4, p5, p6] represents [x, y, z, roll, pitch, yaw] of the platform. As it is expected to see, motions happen only in p1 (tranlsation X), p4 (rotationX) and p6 (rotationZ) and in p3 it is possible to see the presence of the *headFlag* and of the *tailFlag* at the beginning and at the end of the trajectory.

4.4.2 Results

All the experiments were conducted in the PMAR lab under the supervision of the thesis supervisors, and data were collected for the set of 41 trajectories composed of head flag (4 secs), desired trajectory (30 secs) and tail flag (4 secs). A Windows application (see Section 3.6) was built in order to test the MM and to collect data from the experiments and the following configuration parameters were used in the *Model_init.xml* file:

```
-------------------------------------------------------------------------
    <DefaultExperiment
        stepSize        ="0.011104617182934"
        outputFormat        ="csv"
        stopTime        ="50"
        tolerance        ="1e-06"
        solver        ="cvode"
        startTime        ="0"
        variableFilter      = "SP7.six_rss_legs.rss_leg1.j1.phi|
          SP7.six_rss_legs.rss_leg2.j1.phi|SP7.six_rss_legs.rss_leg3.j1.phi|
          SP7.six_rss_legs.rss_leg4.j1.phi|SP7.six_rss_legs.rss_leg5.j1.phi|
          SP7.six_rss_legs.rss_leg6.j1.phi|SP7.platform.j_vertical.phi|
          SP7.platform.platform.frame_a.R.T.*|
          SP7.platform.platform.frame_a.r_0.*" />
-------------------------------------------------------------------------
```

Figures 4.4.10 and 4.4.11 show the results of the MM simulation in comparison to the respective joint of SP7, demonstrating how the model has an optimum behavior even before moving on to error computation. For the sake of clarity, the figure depicts the findings obtained using the same trajectory as in Fig. 4.4.9. After the collected raw data were processed accordingly to what explained in Section 4.3, the errors between MM and SP7 joints were computed and the results were appended one after the other to prepare the sample for the statistical analysis. The computed error for each joint are shown in the following plots, where on the *y* axis the error value is represented and on the *x* axis the sample data are listed:

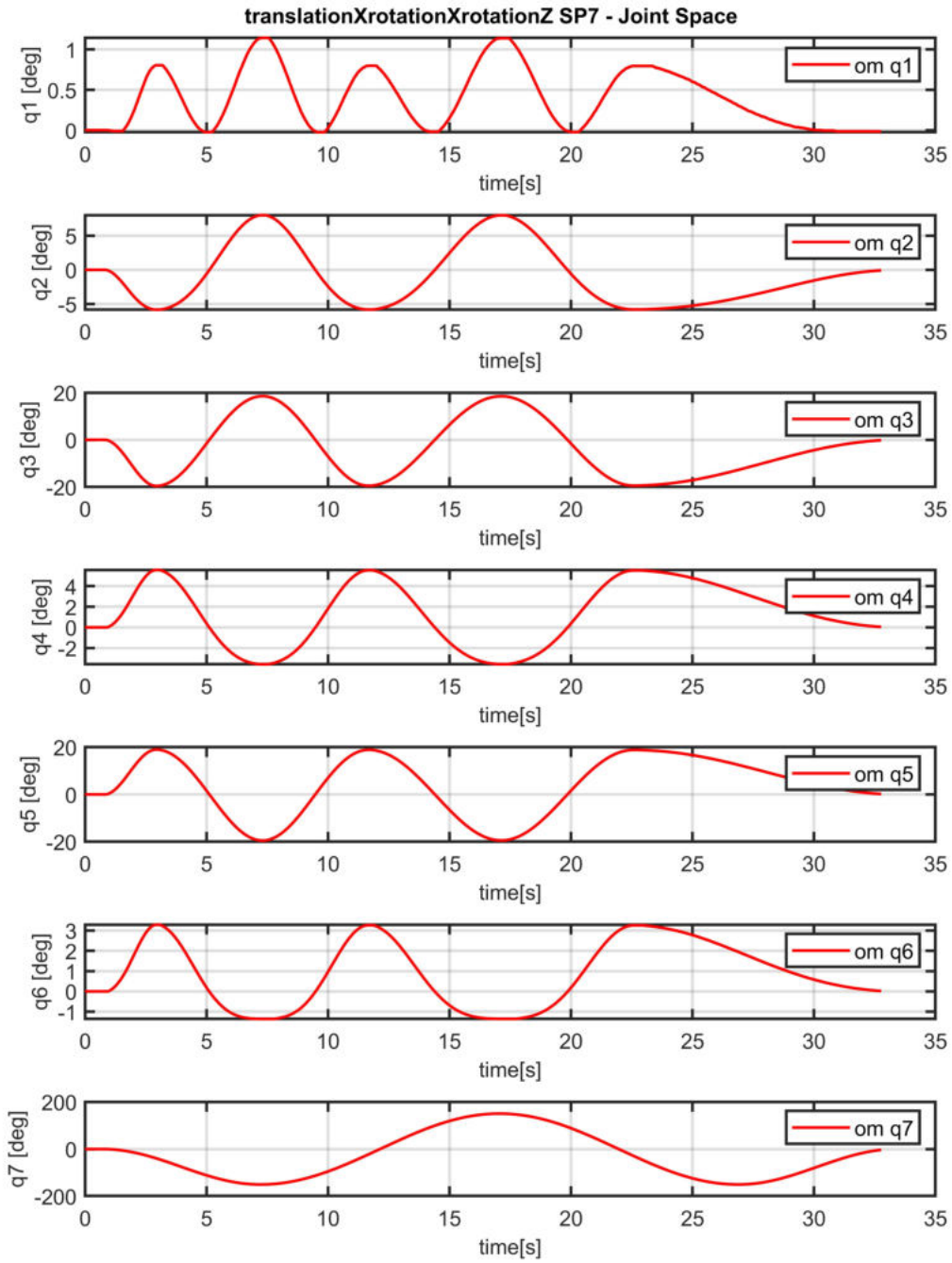


Figure 4.4.10 SP7 Joint Positions: the plots represent the actuator position of SP7 recorded for the *TranslationXRotationXRotationZ* trajectory (the *headFlag* and *tailFlag* motions has already been discarded from the recordings).

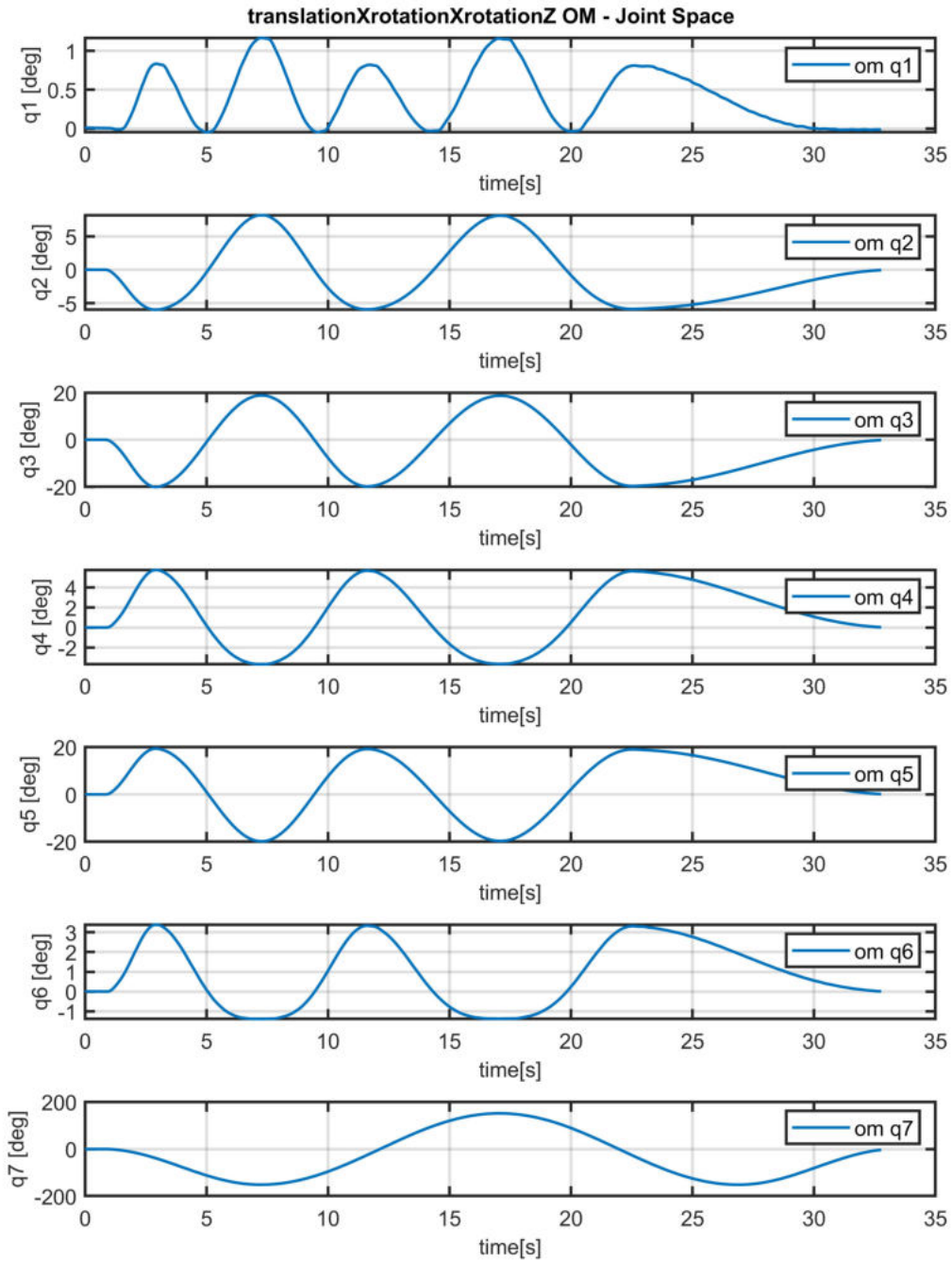


Figure 4.4.11 Modelica Model Joint Positions: the plots represent the joint position on the MM recorded for the *TranslationXRotationXRotationZ* trajectory (the *headFlag* and *tailFlag* motions has already been discarded from the recordings).

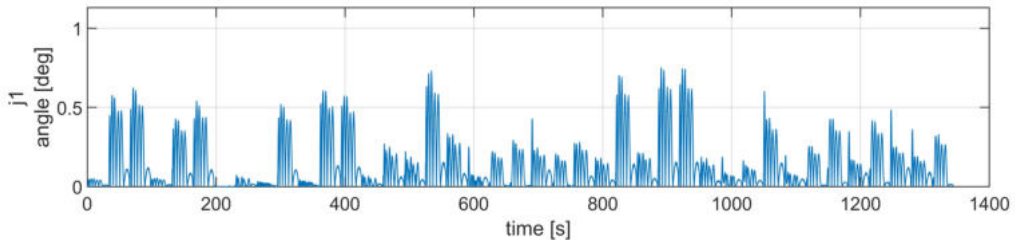


Figure 4.4.12 Accuracy Analysis - Joint Error - j1

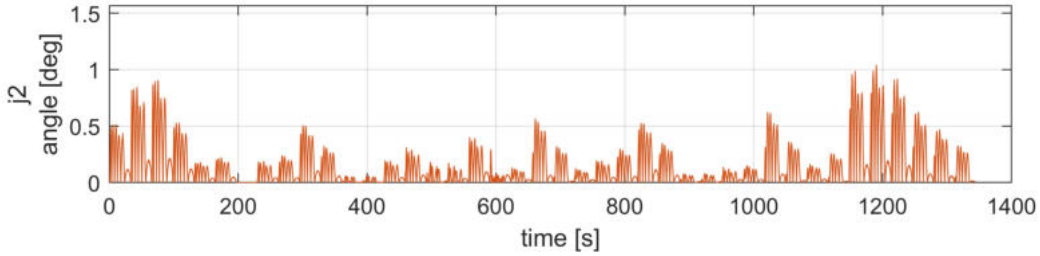


Figure 4.4.13 Accuracy Analysis - Joint Error - j2

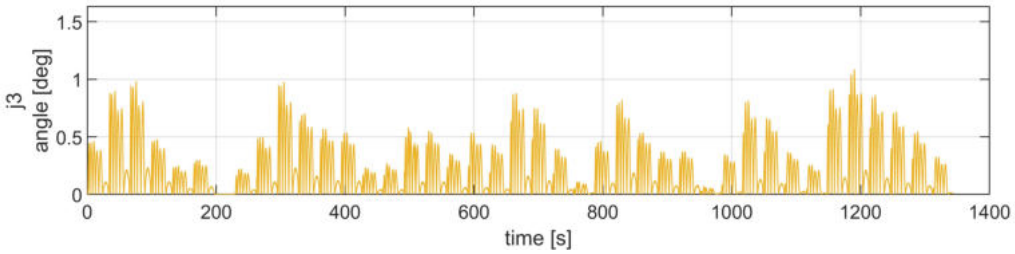


Figure 4.4.14 Accuracy Analysis - Joint Error - j3

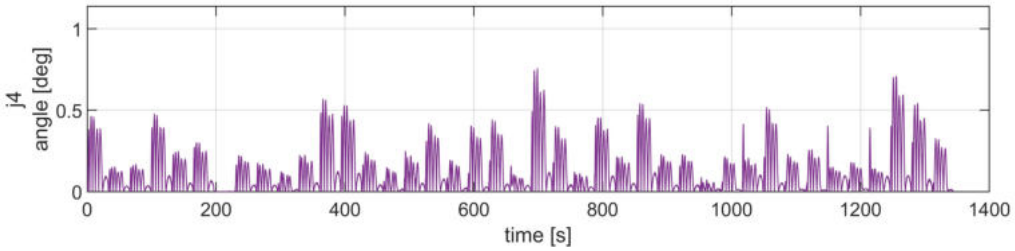


Figure 4.4.15 Accuracy Analysis - Joint Error - j4

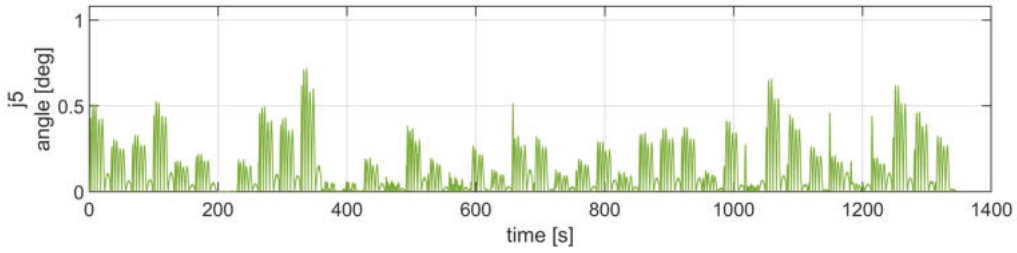


Figure 4.4.16 Accuracy Analysis - Joint Error - j5

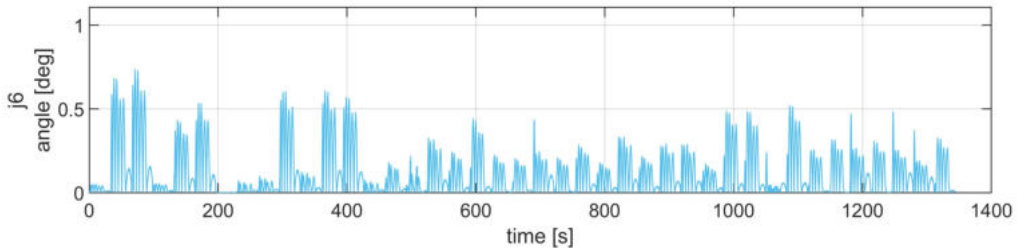


Figure 4.4.17 Accuracy Analysis - Joint Error - j6

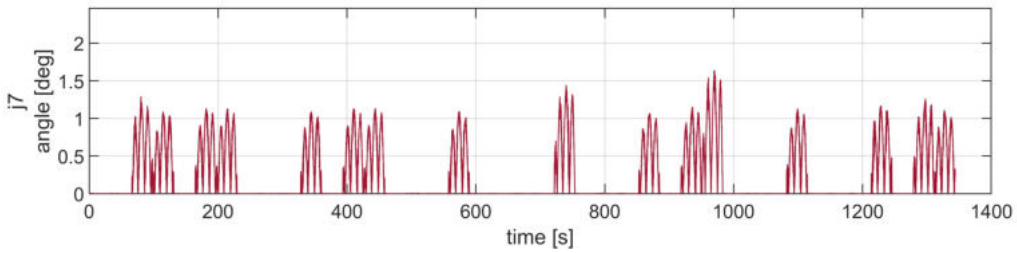


Figure 4.4.18 Accuracy Analysis - Joint Error - j7

The Figures from 4.4.12 to 4.4.18 show the inaccuracy in joint position between the SP7 and the MM: as you can see, this error, measured in degrees, stays very low and replicates each other for all of the trajectories studied. This highlights the durability and efficiency that a parallel mechanism may give during simulations since no trajectory is apparently performed with more complexity than others, in addition to providing an initial evidence of the established model's accuracy. The plots highlights optimal performances for all the seven joint positions with errors that rarely are greater than 1 degree, and a quantitative result is presented in Table 4.4 where the RMSE is found to be in the sub-degree range and mean error and standard deviation are shown. Fig. 4.4.19 shows how error is distributed across the experiments and how it tends to be concentrated nearer zero than farther away.

Table 4.4 Accuracy Analysis - Statistical Results

	RMSE	**E(50)**	**E(95)**	**E(99.7)**	$\bar{e}$	s^2	**Max e**
J space N = 564347	0.2514	0.0695	0.5777	1.0961	0.1488	0.0410	1.6408

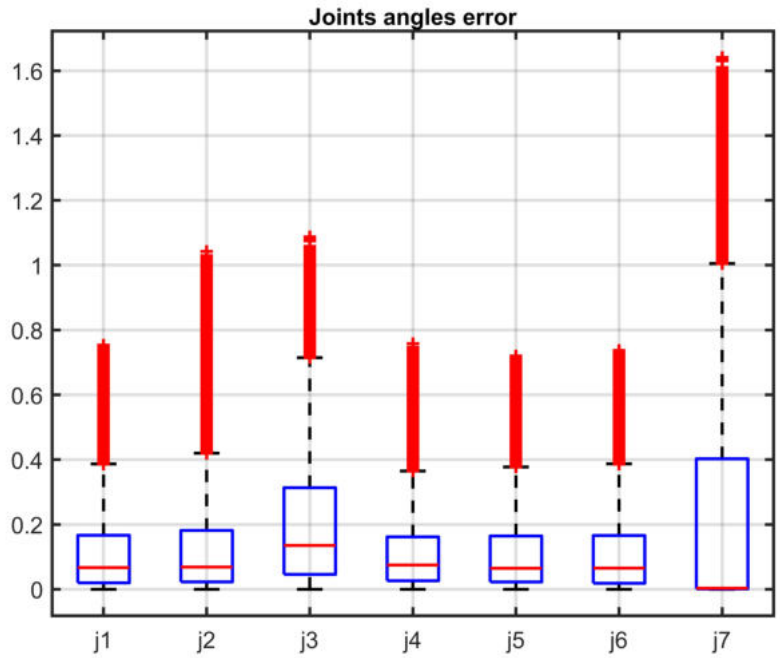


Figure 4.4.19 Accuracy Analysis - Boxplot of Joint Errors

Table 4.5 Accuracy Analysis - Hypothesis Test

	AET value	H_0	**MPET value**	H_0
J space N = 564347	-174.6907	*Accepted*	0.2189	*Accepted*

In Table 4.5, the results of the accuracy analysis both for AET and MPET is shown. AET and MPET values are computed as anticipated with equations (4.1) and (4.2) and thus we can statistically infer the following for the Modelica model:

- Joint Position Error:
  - The average deviation from mean is less than 0.5 degree;
  - The maximum deviation from mean is less 2 degree.

This study highlights the quality of the simulation models and the reliability offered by a development environment such as the one provided by Modelica. In the next test, the precision of this model and the description of the procedural method will be given.

4.5 Precision Analysis

4.5.1 Design of Experiments

Precision represents the closeness of agreement between test results obtained under prescribed conditions from the measurement process being evaluated. During experimental tests, two types of measurements errors can be experienced: systematic errors and random errors. A systematic error has to do with the execution of the experiment: these are errors related to the imperfect technique of the experiment and can be caused from several factors, i.e. the environmental setup, the completeness of the experiment, the calibration phase of the measurement tools or the experimental procedure, and can be spread over the whole set of measurements. A random error is related to the instrument precision: these are inherent errors that cannot be eliminated without changing the instrument and they reflects a degree of intrinsic uncertainty of the instrument. While a systematic error decreases the accuracy of the system more and can be removed through a careful analysis of the aforementioned factors, random error mostly afflict the precision of the systems. Therefore, precision can be considered as an expression of random errors, and thus is a function of the standard deviation of the data that has been observed. The less the standard deviation, the more precise the measurement system is. The test aims to study the precision of the simulation model in performing motions compared to the real one. In order to do this, one desired trajectory has been selected and the Modelica simulation has been launched to obtain a sample size of $N > 30$. Since the system under consideration is a simulation model implemented with an equation-based language, the hypothesis to be verified is that given an unique set of data as

input, the simulation result would always produce the same output. In the result section this concept will be investigated and a both qualitative and quantitative results will be given.

4.5.2 Results

To study the precision of the simulation model, a trajectory was arbitrarily selected which was simultaneously composed of a translation along the x axis of the platform, a rotation along the x axis and another rotation along the y axis with a harmonic motion between the maximum and minimum boundaries of the prescribed workspace of each motion. For each joint, the 30 resulting trajectory errors are appended one after the other and are shown in Fig. 4.5.20 - 4.5.26. Now looking at the comparisons, for each joint, it is possible to notice what appears to be a repeated result on all 30 trajectories, as if to form a patter. This aspect suggests that the developed model enjoys very high, if not maximum, precision since the computed error seems to have the same trend for all the experiments. What has been visually described in the image, can also be quantitative analyzed computing the standard deviation of the joint position for the 30 trajectories in each timestamp. The mentioned analysis results on a computed standard deviation equal to zero between the experiments, which can only be observed when inspecting a simulation model as the one developed. The hypothesis test produced during the design of the experiment is successfully verified since by feeding the model with the same input, the results will never change.

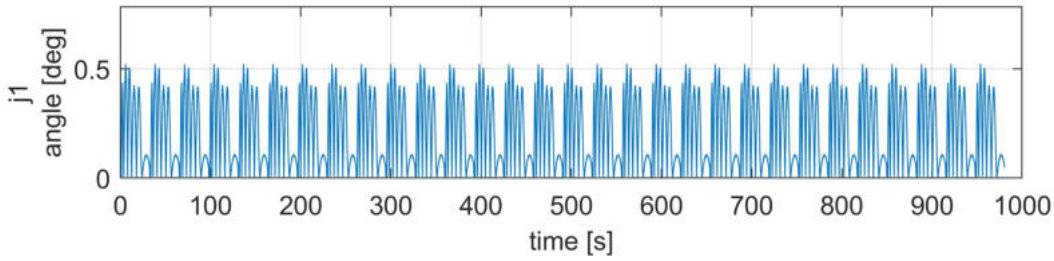


Figure 4.5.20 Precision Analysis - Joint Error - j1

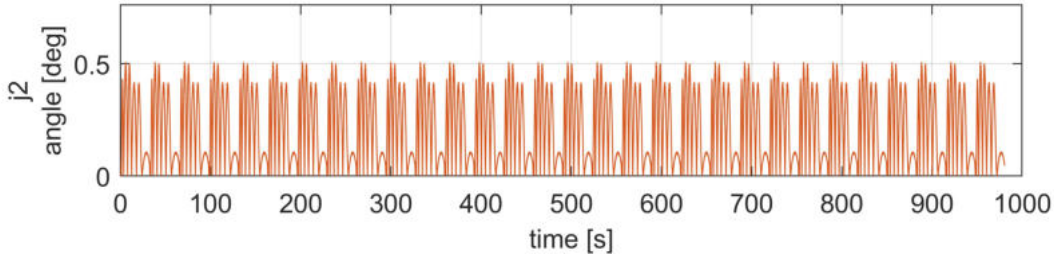


Figure 4.5.21 Precision Analysis - Joint Error - j2

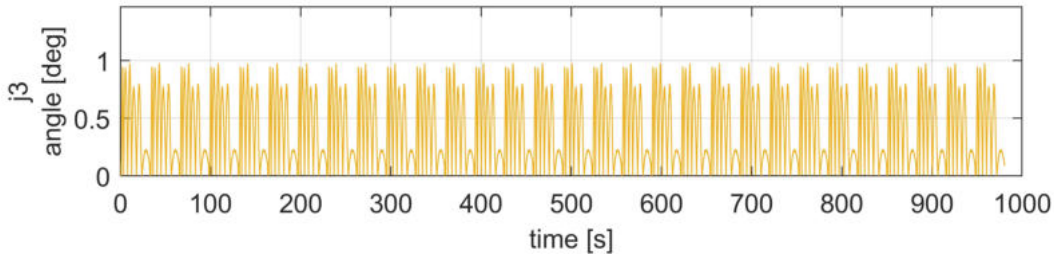


Figure 4.5.22 Precision Analysis - Joint Error - j3

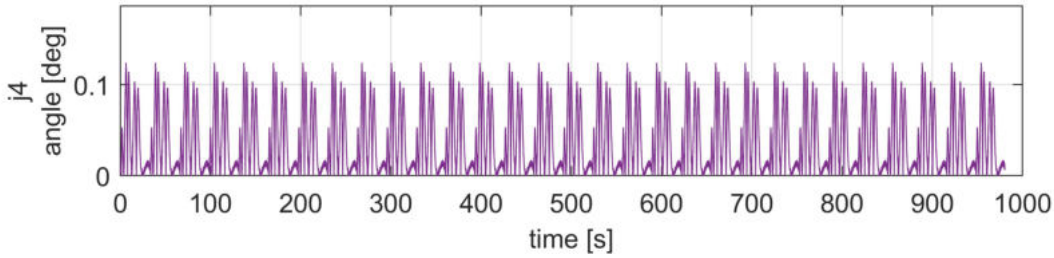


Figure 4.5.23 Precision Analysis - Joint Error - j4

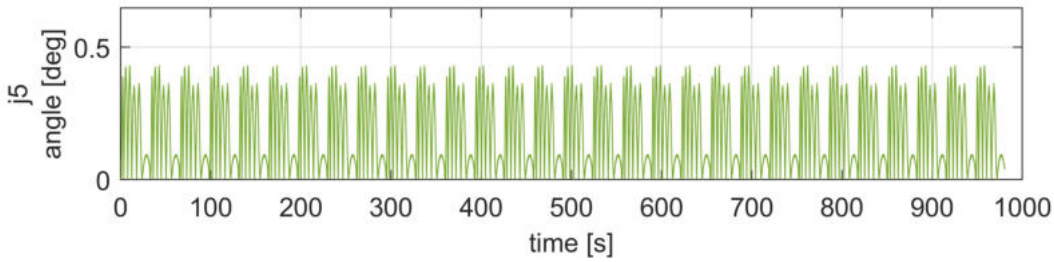


Figure 4.5.24 Precision Analysis - Joint Error - j5

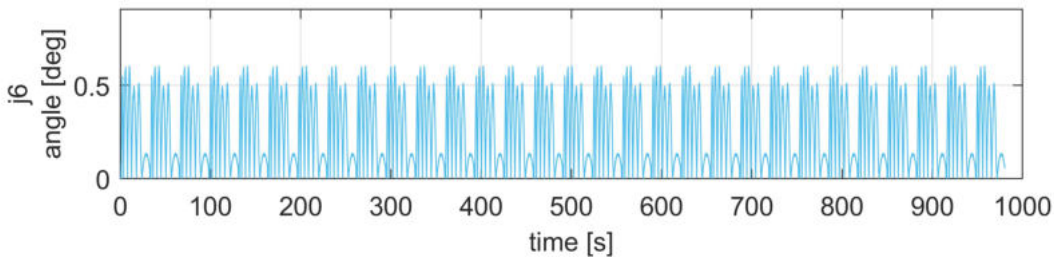


Figure 4.5.25 Precision Analysis - Joint Error - j6

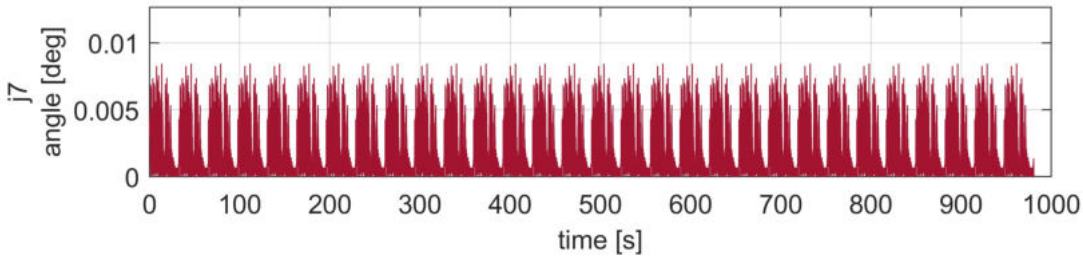


Figure 4.5.26 Precision Analysis - Joint Error - j7

4.6 Conlusions

This thesis presents the development of a dynamic model for the SP7 parallel robot, starting from the definition of the requirements, passing through the study of the possible solutions, the actual implementation and the test procedure for the validation of the model. The SP7 is a motion simulator located in the PMAR laboratory of the DIME mechanics department at the University of Genoa. It is a 7 DOF parallel platform with extraordinary precision and accuracy and it is mainly used in virtual reality projects for creating simulation environments with motion feedback for flight simulators, drone teleoperation and driving simulators. The dynamic model was created with OpenModelica, an open-source environment based on Modelica. The most challenging issue was to model closed chains in space, and how this problem was solved is fully explained in the implementation section. Subsequently, a control unit based on PID controllers was developed to be able to guide the joint present within the dynamic model. After having appropriately calibrated the gains of the PID controls, it was possible to proceed to the validation phase. The experiments were carried out to test the accuracy and precision of the model developed, and as can be seen from the results, it was possible to reach a more than satisfactory level of accuracy, with average errors for the position of the joints below the degree and maximum errors lower than two degrees.

The research developed in this thesis can continue in different ways: first of all it would certainly be necessary to implement the complete model of both the motors and the control in order to have a more detailed insight of the behavior of the actuators. Performances of the system may be improved and a new general purpose architecture can be designed to use it as a real-time motion simulator. With the advent of Industry 4.0, the need to monitor one's own structures to prevent malfunctions has increased more and more: another possible future work is to modify the model developed in order to make it a Digital Twin of the robot: this will bring the development to a new deeper level, since the model will have to go through

a series of structural changes to be able to replicate in all respects the behavior of all the players involved.

References

[1] 1 Introduction> Modelica® - A Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.4. https://specification.modelica.org /v3.4/Ch1.html.

[2] Investigation on the Effects of Structural Dynamics on Rolling Bearing Fault Diagnosis by Means of Multibody Simulation. https://www.hindawi.com/journals/ ijrm/2018/5159189/.

[3] Issue with simulation flag -override OMEdit :: OpenModelica. https://www.openmodelica.org/forum/ default-topic/2116-issue-with-simulation-flag-override-omedit.

[4] Modelica. https://doc.modelica.org/Modelica%204.0.0/ Resources/helpDymola/Modelica.html.

[5] Modelica Language — https://modelica.org/ modelicalanguage.html. https://modelica.org /modelicalanguage.html.

[6] Modelica.Blocks.Continuous.LimPID. https://build.openmodelica.org/Documentation/ Modelica.Blocks.Continuous.LimPID.html.

[7] Modelica.Blocks.Sources.CombiTimeTable. https://build.openmodelica.org/Documentation/ Modelica.Blocks.Sources.CombiTimeTable.html.

[8] Modelica_DeviceDrivers.Blocks.OperatingSystem.SynchronizeRealtime. https://build.openmodelica.org/Documentation/ Modelica_DeviceDrivers.Blocks.OperatingSystem.SynchronizeRealtime.html.

[9] OpenModelica website - https://www.openmodelica.org/. https://openmodelica.org/.

[10] Solving Modelica Models — OpenModelica User's Guide v1.19.0-dev-284-gf58737cb1dd documentation. https://www.openmodelica.org/doc/ OpenModelicaUsersGuide/latest/solving.html.

[11] SpaceMouse Pro - Negozio ufficiale 3Dconnexion IT. https://3dconnexion.com/it/product/spacemouse-pro/.

[12] SPINEA Catalogue. 252.

[13] User Manual DUET_FL. 196.

[14] youBot Store. http://www.youbot-store.com/developers/modelica-model-of-youbot-arm.

[15] Wayback Machine. https://web.archive.org/web/20130506134518/http://www.wokinghamu3a.org.uk/Maths%20of%20the%20Stewart

[16] Introduction to the model translation and symbolic processing, Nov. 2017.

[17] Modelica_DeviceDrivers. Modelica 3rd-party libraries, Feb. 2022.

[18] Regular expression. *Wikipedia* (Feb. 2022).

[19] AIVALIOTIS, P., GEORGOULIAS, K., ARKOULI, Z., AND MAKRIS, S. Methodology for enabling Digital Twin using advanced physics-based modelling in predictive maintenance. *Procedia CIRP 81* (Jan. 2019), 417–422.

[20] AIVALIOTIS, P., GEORGOULIAS, K., AND CHRYSSOLOURIS, G. The use of Digital Twin for predictive maintenance in manufacturing. *International Journal of Computer Integrated Manufacturing 32*, 11 (Nov. 2019), 1067–1080.

[21] ASADA, H., AND SLOTINE, J.-J. E. *Robot Analysis and Control*. John Wiley & Sons, 1986.

[22] ÅSTRÖM, K. J., AND HÄGGLUND, T. *PID Controllers: Theory, Design, and Tuning*. ISA - The Instrumentation, Systems and Automation Society, Research Triangle Park, North Carolina, 1995.

[23] BENEDICT, M., MATTABONI, M., CHOPRA, I., AND MASARATI, P. Aeroelastic Analysis of a Micro-Air-Vehicle-Scale Cycloidal Rotor in Hover. *AIAA Journal 49*, 11 (Nov. 2011), 2430–2443.

[24] BIANCOLINI, M. E., CELLA, U., GROTH, C., AND PORZIANI, S. An overview of the latest advances of the use of CAE technologies in the medical field. *CASE STUDIES* (2020), 9.

[25] BRETT, J. G., BROWNING, B., AND VELOSO, M. Accurate and Flexible Simulation for Dynamic, Vision-Centric Robots. In *In: Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems* (2004).

[26] BRIOT, S., AND KHALIL, W. *Dynamics of Parallel Robots: From Rigid Bodies to Flexible Elements*, vol. 35 of *Mechanisms and Machine Science*. Springer International Publishing, Cham, 2015.

[27] CHRISTEN, E., BAKALAR, K., DEWEY, A. M., AND MOSER, E. Analog and Mixed-Signal Modeling Using the VHDL-AMS Language. 199.

[28] CRAIG, J. J. Mechanics and Control Third Edition. 408.

[29] DESAI, N., ANANYA, S. K., BAJAJ, L., PERIWAL, A., AND DESAI, S. R. Process Parameter Monitoring and Control Using Digital Twin. In *Cyber-Physical Systems and Digital Twins*, M. E. Auer and K. Ram B., Eds., vol. 80. Springer International Publishing, Cham, 2020, pp. 74–80.

[30] DYM, C. L., AND SHAMES, I. H. *Solid Mechanics*. Springer New York, New York, NY, 1973.

[31] GALLARDO-ALVARADO, J. An overview of parallel manipulators. In *Kinematic Analysis of Parallel Manipulators by Algebraic Screw Theory*. Springer International Publishing, Cham, 2016, pp. 19–28.

[32] GLATT, M., SINNWELL, C., YI, L., DONOHOE, S., RAVANI, B., AND AURICH, J. C. Modeling and implementation of a digital twin of material flows based on physics simulation. *Journal of Manufacturing Systems 58* (Jan. 2021), 231–245.

[33] HAHN, H. *Rigid Body Dynamics of Mechanisms: 1 Theoretical Basis*. Springer Science & Business Media, Mar. 2002.

[34] HAMILTON, S. W. R. On a general method in dynamics. 65.

[35] HAMMARSTRÖM, J. Validation of a Digital Twin for a Ship Engine Cooling System. 73.

[36] KAI LIU, AND FITZGERALD, M. Modeling and control of a stewart platform manipulator, 1991.

[37] KANE, T. R., LIKINS, P. W., AND LEVINSON, D. A. Spacecraft dynamics. *New York* (Jan. 1983).

[38] KEPPLER, D. V. Creating an accurate digital twin of a human user for realistic modeling and simulation. *CASE STUDIES*, 4 (2019), 4.

[39] KHATIB, O., BROCK, O., CHANG, K.-S., RUSPINI, D., SENTIS, L., AND VIJI, S. Human-Centered Robotics and Interactive Haptic Simulation. *The International Journal of Robotics Research 23*, 2 (Feb. 2004), 167–178.

[40] KUMAR, S. DESIGN, SIMULATION AND TESTING OF SHRIMP ROVER USING RECURDYN. 7.

[41] LANCZOS, C. *The Variational Principles of Mechanics*. Courier Corporation, 1970.

[42] LIAU, Y., LEE, H., AND RYU, K. Digital Twin concept for smart injection molding. *IOP Conference Series: Materials Science and Engineering 324* (Mar. 2018), 012077.

[43] MERLET, J. P. *Parallel Robots*. Springer Science & Business Media, Dec. 2005.

[44] MILLER, A., AND CHRISTENSEN, H. Implementation of multi-rigid-body dynamics within a robotic grasping simulator. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)* (Sept. 2003), vol. 2, pp. 2262–2268 vol.2.

[45] MURASHOV, I., ZVEREV, S., SMORODINOV, V., AND GRACHEV, S. Development of digital twin of high frequency generator with self-excitation in Simulink. *IOP Conference Series: Materials Science and Engineering 643* (Nov. 2019), 012078.

[46] NEGRI, E., FUMAGALLI, L., CIMINO, C., AND MACCHI, M. FMU-supported simulation for CPS Digital Twin. *Procedia Manufacturing 28* (Jan. 2019), 201–206.

[47] OÑEDERRA, O., ASENSIO, F. J., EGUIA, P., PEREA, E., PUJANA, A., AND MARTINEZ, L. MV Cable Modeling for Application in the Digital Twin of a Windfarm. In *2019 International Conference on Clean Electrical Power (ICCEP)* (July 2019), pp. 617–622.

[48] PIMENTA, F., PACHECO, J., BRANCO, C. M., TEIXEIRA, C. M., AND MAGALHÃES, F. Development of a digital twin of an onshore wind turbine using monitoring data. *Journal of Physics: Conference Series 1618* (Sept. 2020), 022065.

[49] POLLARD JR., W. L. G. Spray painting machine, Aug. 1940.

[50] QI, Q., TAO, F., HU, T., ANWER, N., LIU, A., WEI, Y., WANG, L., AND NEE, A. Y. C. Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems 58* (Jan. 2021), 3–21.

[51] RAINERI, I., LA MURA, F., AND GIBERTI, H. Digital Twin Development of HexaFloat, a 6DoF PKM for HIL Tests. In *Advances in Italian Mechanism Science* (Cham, 2019), G. Carbone and A. Gasparetto, Eds., Mechanisms and Machine Science, Springer International Publishing, pp. 258–266.

[52] RASSÕLKIN, A., RJABTŠIKOV, V., VAIMANN, T., KALLASTE, A., KUTS, V., AND PARTYSHEV, A. Digital Twin of an Electrical Motor Based on Empirical Performance Model. In *2020 XI International Conference on Electrical Power Drive Systems (ICEPDS)* (Oct. 2020), pp. 1–4.

[53] SCHLUSE, M., AND ROSSMANN, J. From simulation to experimentable digital twins: Simulation-based development and operation of complex technical systems. In *2016 IEEE International Symposium on Systems Engineering (ISSE)* (Oct. 2016), pp. 1–6.

[54] SINHA, R., PAREDIS, C. J. J., LIANG, V.-C., AND KHOSLA, P. K. Modeling and Simulation Methods for Design of Engineering Systems. *Journal of Computing and Information Science in Engineering 1*, 1 (Nov. 2000), 84–91.

[55] STEWART, D. A Platform with Six Degrees of Freedom. *Proceedings of the Institution of Mechanical Engineers 180*, 1 (June 1965), 371–386.

[56] TALKHESTANI, B. A., JUNG, T., LINDEMANN, B., SAHLAB, N., JAZDI, N., SCHLOEGL, W., AND WEYRICH, M. An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System. *at - Automatisierungstechnik 67*, 9 (Sept. 2019), 762–782.

[57] VASILEIOU, C., SMYRLI, A., DROGOSIS, A., AND PAPADOPOULOS, E. Development of a passive biped robot digital twin using analysis, experiments, and a multibody simulation environment. *Mechanism and Machine Theory 163* (Sept. 2021), 104346.

[58] VATHOOPAN, M., JOHNY, M., ZOITL, A., AND KNOLL, A. Modular Fault Ascription and Corrective Maintenance Using a Digital Twin. *IFAC-PapersOnLine 51*, 11 (Jan. 2018), 1041–1046.

[59] VLADAREANU, L., VLADAREANU, V., GAL, A. I., MELINTE, O. D., PANDELEA, M., RADULESCU, M., AND CIOCÎRLAN, A.-C. Digital Twin in 5G Digital era developed through Cyber Physical Systems. *IFAC-PapersOnLine 53*, 2 (Jan. 2020), 10885–10890.

[60] YANG, L., AND ZHANG, X. Dynamics Modeling and Simulation of Robot Manipulator. In *Intelligent Robotics and Applications* (Cham, 2015), H. Liu, N. Kubota, X. Zhu, and R. Dillmann, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 525–535.

Appendix A

Step by Step Experimental Procedure

SP7 Motion Platform

1. Switch on:
    a. Power on the platform
        i. Turn on the blue switch in the wall
        ii. Turn on the white switch on the floor
    b. Power on the network switch
        i. Turn on the network switch and the raspberry
    c. Set up the platform (with the light blue control station)
        i. Check that the yellow stripe in the emergency button in visible (pull up the button)
        ii. Press the white button (Reset)
        iii. Press the green button (Start)
2. Switch off:
    a. Power off the platform
        i. Check that the yellow stripe in the emergency button in visible (pull up the button)
        ii. Press the red button (Stop)
        iii. Turn off the white switch on the floor
        iv. Turn off the blue switch in the wall
    b. Power on the network switch
        i. Turn off the network switch and the raspberry

SP7 Control Application

1. Run the application
2. Connect to socket
    a. Insert the IP address
    b. Press Connect
3. Possible controls
    a. Load trajectory from file (after connection)
        i. Press “Open”
        ii. Select txt file containing the trajectory
    b. Controller
        i. Plug the Space Mouse in
4. Start the simulation
    a. Press Start Feeder
5. Stop the simulation
    a. From light blue controller: Press the Emergency Button (hard stop)
    b. From light blue controller: Press the Stop Button (idle state)
    c. From SP7App: press Stop Feeder
6. Stop the application
    a. Disconnect from socket
    b. Close the application

Steam VR

1. Turn on Steam
2. Set up VR device
    a. Plug display port, power, usb cable and the wire coming from VR in the vive hub
    b. Press the blue button to turn on the hub
    c. Plug in the power supply for the base stations
3. Launch SteamVR
4. Set up Play Area (not necessary to do every time)
    a. Within SteamVR window: press Room Setup

b. Follow the steps shown

5. Setup Vive tracker

   a. Allocate the tracker in the right position (never move them after that)
   b. Plug in the three receivers to your pc
   c. Connect the vive tracker devices (press and hold x2) in the following order:
      i. LHR – 09DD940F
      ii. LHR – 09DE45F2
      iii. LHR – 0FC0A69A
   d. Inside STEAMVR -> three line menu -> Devices -> Pair Controller

6. Disconnect SteamVR

   a. Vive tracker will automatically switch off
   b. Turn off the vive hub (press blue button) and unplug the power supply and the VR cable
   c. Unplug the power supply from the base stations

7. Turn off steam
8. Recharge vive tracker