

A Study on the Lagrangian Formulation of Dynamics with Applications to Control of Parallel Manipulators

A project report

submitted by

AKHIL SATHULURI

*in partial fulfilment of requirements
for the award of the dual degree of*

**BACHELOR OF TECHNOLOGY IN
ENGINEERING DESIGN
AND
MASTER OF TECHNOLOGY IN
AUTOMOTIVE ENGINEERING**



**DEPARTMENT OF ENGINEERING DESIGN
INDIAN INSTITUTE OF TECHNOLOGY MADRAS**

MAY 2019

CERTIFICATE

This is to certify that the project titled **A Study on the Lagrangian Formulation of Dynamics with Applications to Control of Parallel Manipulators**, submitted by **Mr Akhil Sathuluri**, to the Indian Institute of Technology Madras, for the award of the degrees of **Bachelor of Technology** and **Master of Technology**, is a *bona fide* record of the research work done by him under my supervision. The contents of this project, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr Sandipan Bandyopadhyay
Project Adviser
Associate Professor
Department of Engineering Design
Indian Institute of Technology Madras
Chennai 600 036

Place: Chennai

Date: May 7, 2019

ACKNOWLEDGEMENTS

I would like to thank my adviser for his guidance and all the lab members for their support, with a special thanks to Teja and Anirban who always made time to accommodate my doubts and Ambuj for his codes which have cut down my effort significantly. I also extend my thanks to Phanindra, Manikandan, Pawel, V Manoj for the discussions which helped me understand things with more clarity. Finally I would like to thank the institute and my family for their constant support.

ABSTRACT

KEYWORDS: Parallel manipulators, Semi-regular Stewart platform manipulator, 6-RSS manipulator, Lagrangian dynamics, feedback linearisation, extended-configuration-space, computational time, computed torque control, trajectory-tracking control

Parallel manipulators are a class of robots characterised by their closed-loop architecture which gives them the advantages of high precision and load carrying capacity over their serial counterparts. Due to the presence of closed kinematic loops, the dynamics model consists of both actuated and passive joints related by sizeable symbolic expressions, evaluation of which hinders the speed of computer simulation. As a result, in practice, the dynamics model is often simplified, and control algorithms are used to compensate the model inaccuracies. Further, the computation of a model-based control input in realistic time scales with a precise formulation of the system remains a challenge. Therefore, a dynamics formulation that would enable faster calculations without compromising on the fidelity of the model forms the central idea of the report.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF TABLES	vi
LIST OF FIGURES	viii
ABBREVIATIONS	ix
NOTATIONS	x
1 Introduction	1
1.1 Problem Statement	1
1.2 Motivation	1
1.3 Choice of system for simulation: SRSPM	2
1.4 Literature survey	3
1.4.1 Approaches for dynamic modelling of parallel manipulators	3
1.4.2 Methods in dynamic simulation and control	4
1.4.3 Computational efficiency and implementations of dynamics models	6
1.5 Overview of the report	7
1.6 Summary	7
2 Dynamics models of parallel manipulators using the Lagrangian formulation	9
2.1 Basic definitions and notations	9
2.2 Lagrangian formulation of the equation of motion	10
2.3 Configuration-space formulation	11
2.4 Actuator-space formulation	11
2.5 Task-space formulation	12

2.6	Forward Kinematics	12
2.7	Inverse Kinematics	12
3	Elements of kinematic modelling of parallel manipulators	13
3.1	Introduction	13
3.1.1	Convention for representing the moving platform	13
3.1.2	Root-tracking methods	14
3.2	Formulations of velocity Jacobian matrices	17
3.3	Summary	19
4	Formulation of the dynamics models	20
4.1	Introduction	20
4.2	Formulation the of the dynamics model in the configuration-space .	20
4.3	Formulation of the dynamics model in the actuator-space	25
4.4	Formulation of the dynamics model in the task-space	26
4.4.1	Implementation	26
4.5	Summary	27
5	Dynamics model in the extended-configuration-space	29
5.1	Formulation of the dynamics model	29
5.2	Extended configuration-space model mapped to task-space variables	32
5.3	Extended configuration-space model mapped to actuator-space variables	33
5.4	Summary	34
6	Simulation and comparison of various formulations	35
6.1	Introduction	35
6.1.1	Dynamic simulations and observations	35
6.1.2	Implementation of dynamic simulations in C++	37
6.2	Comparison of different methods	39
6.3	Example: Dynamics formulation and simulation of 6-RSS manipulator	40
6.3.1	Verification of the formulated model	41
6.3.2	Inverse dynamics for heave motion requirement	42
6.4	Summary	43
7	Trajectory-tracking control with the SRSPM	45

7.1	Introduction	45
7.2	Feedback linearisation	46
7.2.1	The SRSPM following a given circular path	46
7.2.2	Following a 3D-Lissajous curve with SRSPM	47
7.3	Following a rectangular path with SRSPM	50
7.4	Summary	51
8	Conclusion	53
8.1	Overview	53
8.2	Possible extensions	54
A	Manipulator parameters	56
B	Effect of compilers and flags on the execution time of dynamics models	57
C	Details of C++ implementation of the dynamics models	58

LIST OF TABLES

1.1	Comparison of the number of floating point operations of various methods, reproduced from [27]	7
4.1	Size of the coefficient matrices used in configuration-space dynamic model of the manipulator	25
4.2	Size of the coefficient matrices used in dynamics modelled in the task-space of the manipulator	27
5.1	Size of the coefficient matrices used in dynamics modelled in the extended-configuration-space mapped to task-space the manipulator	32
6.1	Computational time taken for the free-fall simulation in <i>Mathematica</i> ® 11.2	36
6.2	Computational time taken for the free-fall simulation in C++	38
6.3	Mechanical parameters of the 6-RSS manipulator used in the simulation of the dynamics model	44
A.1	Mechanical parameters of the SRSPM used in the simulation of the dynamics model	56
A.2	Details of the tracked 3D-Lissajous curve	56
B.1	Comparison of compilers with respect to compilation and execution times for a free-fall simulation time of 0.5 s	57
B.2	Comparison of flags with respect to compilation and execution times of the C matrix	57

LIST OF FIGURES

1.1	Semi-Regular Stewart Platform Manipulator	2
3.1	Base and axis representation followed in and reproduced from [6] . .	14
3.2	Relative orientations of the base and the moving platform with respect to the global frame of reference used in this report	15
4.1	Absolute maximum error in loop-closure equations with time as given in Eq. (4.2)	22
4.2	Absolute maximum of first order loop-closure equations with time as given in Eq.(4.3)	23
4.3	Absolute maximum of second order loop-closure equations with time as given in Eq.(4.4)	24
4.4	Percentage change in the total mechanical energy of the system with time as given in Eq. (4.5)	24
4.5	Validation of the actuator-space mathematical model	26
4.6	Validation of the task-space mathematical model	28
5.1	Validation of the extended-configuration-space mapped to task-space mathematical model	33
5.2	Validation of the extended-configuration-space mapped to actuator-space mathematical model	34
6.1	Motion of the manipulator when simulated in the actuator-space . .	36
6.2	Motion of the manipulator when simulated in the configuration-space	37
6.3	Motion of the manipulator when simulated in the extended-configuration-space	37
6.4	Visualising the sparsity of \mathbf{M} and \mathbf{C} matrices in the extended-configuration space vs. configuration-space formulations	40
6.5	The 6-RSS manipulator	41
6.6	Validation of the extended-configuration-space mapped to actuator-space mathematical model	42
6.7	Torque profile for the heave motion requirement of the moving platform	43
6.8	Configurations of the manipulator during heave motion	43

7.1	Following a circular path by SRSPM	47
7.2	The 3D Lissajous curve to be tracked by SRSPM	48
7.3	Norm of the tracking error	49
7.4	Norm of the tracking error	49
7.5	Followed vs desired path of the 3D-Lissajous curve by SRSPM . . .	50
7.6	Followed vs desired rectangular path by SRSPM	50
7.7	Error in the followed vs desired rectangular path by SRSPM	51
7.8	Followed vs desired rectangular path by SRSPM	51
C.1	Errors using the explicit Runge-Kutta solver for configuration-space dynamic model simulation	59

ABBREVIATIONS

SRSPM	Semi Regular Stewart Platform Manipulator
SWZ	Safe Working Zone
FK	Forward Kinematics
FKU	Forward Kinematic Univariate
IK	Inverse Kinematics
CTC	Computed Torque Control
NR	Newton-Raphson

NOTATIONS

θ	Set of variables representing the actuator joint variables
ϕ	Set of variables representing the passive joint variables
x	Set of variables representing the task-space coordinates
q	Set of variables representing the configuration-space coordinates, $\{\theta, \phi\}$
q_e	Set of variables representing the extended-configuration-space, $\{x, \theta, \phi\}$
DoF	Degrees of freedom
n	Number of DoF/ Number of actuator variables
m	Number of passive variables
$ \cdot $	Returns the absolute value of all the elements of an input vector
$(\cdot)^\vee$	The 3-vector form of a 3×3 skew-symmetric matrix
$\text{tr}(\cdot)$	Trace of a matrix
$\max(\cdot)$	Maximum value among all the elements of a given list
$\min(\cdot)$	Minimum value among all the elements of a given list

CHAPTER 1

Introduction

1.1 Problem Statement

This report deals with the problem of obtaining a dynamics model within the Lagrangian framework, that reduces the computational effort and enables the implementation of model-based control of parallel manipulators.

1.2 Motivation

Parallel manipulators are a class of robots characterised by their closed-loop architecture which gives them the advantages of high precision and load carrying capacity over their serial counterparts. Due to the presence of closed kinematic loops, the manipulator has both actuated and passive joints related by sizeable symbolic expressions, evaluation of which hinders the speed of computer simulation. As a result, in practical implementations, the dynamics model is often simplified, and control algorithms are used to compensate the model inaccuracies. Even if one has a precise formulation of a dynamics model, model-based control computation in realistic time scales remains a challenge. Therefore, a formulation that would enable faster calculations without compromising on the fidelity of the model is needed.

In general, recursive Newton-Euler dynamic models are fast as shown in [52]; though these are not in the scope of the report, for lower degree-of-freedom systems, Lagrangian formulation might still hold a chance to match and may even surpass Newton-Euler formulation in terms of computational efficiency. Moreover, the Lagrangian method brings out the differential-geometric structure which is useful in deriving insights in some cases. For an in-length discussion on this aspect, one may refer to [36]. Such a structure in the formulation further allows one to study aspects such as controllability and helps in the design of non-linear controllers which respect the properties of

the group in which the system evolves. Hence, despite having various formulations of dynamics, which would meet the requirements of fidelity and computational time, in this report the system is modelled only in the Lagrangian framework. Since the idea of this work is not to investigate singularities, it is assumed that the manipulator is moving within a Safe Working Zone (SWZ) as defined in [26], unless otherwise specified. The next section describes the Stewart Platform Manipulator (SPM), which is used throughout the report to benchmark various dynamics formulations followed by a brief overview on the literature on formulation, simulation and control.

1.3 Choice of system for simulation: SRSPM

For the verification of dynamic models to be formulated in the report, Semi-Regular Stewart Platform Manipulator (SRSPM) is used as a benchmarking example. The Gough-Stewart or the Stewart platform manipulator is a six degree-of-freedom platform type manipulator, as shown in Fig. 1.1. Its construction involves a ground or fixed platform, connected to six linear actuators (legs) through universal or spherical joints. A moving platform is attached to the other ends of the linear actuators via spherical joints as shown in Fig. 1.1. Since its introduction in [50], this manipulator has attracted a large amount of research on the topics of kinematics, dynamics and control.

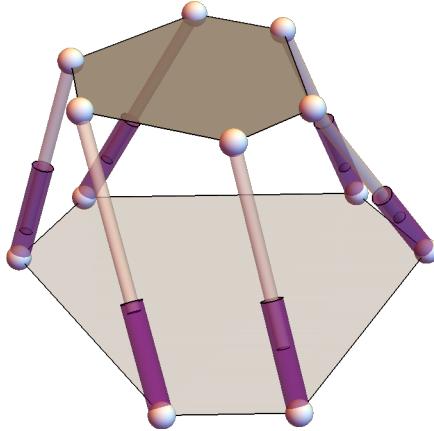


Figure 1.1: Semi-Regular Stewart Platform Manipulator

The interest in this particular manipulator comes from its wide range of applications including automotive simulators [16, 45], flight simulators [46], motion of machine tools [31] etc.

1.4 Literature survey

The current section presents a summary of the literature survey on the topics of simulation, control, computational efficiency and dynamics formulations concerning the Stewart Platform Manipulator (SPM).

1.4.1 Approaches for dynamic modelling of parallel manipulators

A dynamic model is a description of a system's evolution with time in terms of chosen generalised coordinates on the configuration-space, given an input. The kinematic constraints introduced by the closed-loop nature of parallel manipulators make the formulation of equations of motion complicated. This section presents a brief overview of the attempts made to formulate the dynamics of the Stewart platform using various techniques.

The earliest study of the dynamics of the Stewart Platform, as given in [15], discusses the statics problem of the manipulator by neglecting the leg masses. Later, the effect of leg inertia was studied through dynamic simulations in [25]. Since the effects of the inertia of the legs have a significant dependence on the motion specification for the required applications, no explicit rules for neglecting or considering the inertias was given. Further it was concluded that, the leg inertias are ignored when the velocity of the end-effector is slow, or the payload mass is comparably large in which cases the errors are still within the tolerable range.

Newton-Euler formulation

A Newton-Euler approach was adopted in [14] for the formulation of dynamics of the 6-SPS platform with the assumption of slender legs i.e., the legs are effectively treated as one-dimensional bodies, to study the inverse dynamics problem for a given path. The work carried in [11] deals with the modelling of the task-space dynamics using the Newton-Euler method and its simulation in MATLAB using `ode45` solver, which is based on the 4th and 5th order Runge-Kutta integrator with adaptive step size. The work claims that unlike in serial manipulators where the description is in the joint-space, for parallel manipulators, the natural description is in the task-space. Further, the

significance of the theoretical results in observability and the challenges in designing controllers for parallel manipulators due to the existence of multiple forward kinematic branches have also been illustrated. A general formulation of closed-form dynamics equations of parallel manipulators using Newton-Euler formulation is presented in [9].

Euler-Lagrange formulation

A Lagrangian dynamic model with the task-space variables as the generalised coordinates has been obtained in [17, 38]. A complete description of the dynamics in the task-space using the Lagrangian formulation was given in [31]. The closed-form expressions for the leg velocity Jacobian matrices have not been obtained in the work. Moreover, the work demonstrates solving the inverse kinematics (IK) problem to obtain the values of the configuration variables from the task-space variables. Further an algorithm to obtain the M , C and G , matrices of the legs was also discussed.

Other methods

Apart from the approaches mentioned above, for the formulation of the dynamics model, authors of [8] have proposed a Bond Graph-based approach. The procedure is based on a hierarchical multi-level approach. Further, a set of 57 equations can be formulated using Kane's method as shown in [39] which claims to have a benefit of simplicity and to reduce the number of symbolic computations required. Further, the formulated system was simulated using the `ode45` of MATLAB. An approach using virtual-work has been employed in [53]. A modification over virtual-work was proposed by [55] which was stated to be faster than Newton-Euler methods.

1.4.2 Methods in dynamic simulation and control

The work in [40] discusses general issues with dynamics and controls of parallel manipulators. Modelling and control of a hydraulically actuated SPM are presented in [37], with assumptions that the dynamic effect of the legs is negligible with respect to the payload and the effect of leg dynamics was evaluated empirically using experiments for a pressure feedback control algorithm. The work [39] suggested the use of the

Baumgarte stabilisation method for solving the DAEs and simulation in MATLAB using `ode45` was presented. A stiffness-based control scheme for application in milling was proposed and formulated in [31].

The time taken to compute FK is reported as 5-15 ms, in [13] and a dynamic model with negligible actuator masses was simulated with a settling time constraint of 0.2 s. For the sake of position control, the dynamic model was approximated by assuming the non-linear terms to be constant in [32]. These modelling idealisations were treated as disturbances and were expected to be compensated by the proposed H_∞ controller. It was assumed that the operating bandwidth of parallel manipulators is within 10 Hz and a 50 MHz DSP was used for the implementation of control. It was reported that the full inverse dynamics computation takes around 25 ms and the approximated inverse dynamics lesser than 1 ms. It was also observed that using the complete dynamic model produced larger errors while tracking corners or high-frequency signals because of the increase in the required sampling rate. A real-time adaptive controller was implemented on the SPM in [44] assuming slow motion of the manipulator, i.e., the time-varying coefficients are assumed as constants. A model-based high-speed tracking control strategy was implemented in [29]. Based on the small workspace used and the high-frequency application proposed, the Coriolis and centripetal terms were neglected and were compensated by a sliding-mode controller. Further, [24] proposed a sliding-mode tracking controller and [51] proposed a disturbance rejection high precision control augmenting a non-linear PD control of SPM.

It is shown in [23] that for speeds more than 0.1 m/s of the moving platform the tracking errors of PD joint controllers are generally huge and have shown that a manipulator with highly non-linear dynamics can attain high accuracy only when a non-linear controller is used. A much more recent work, [48], has suggested the use of Adams-MATLAB co-simulation for motion control of SPM. Co-simulation enables real-time control and aims to reduce the cost of programming complexity and the cost of physical prototyping.

1.4.3 Computational efficiency and implementations of dynamics models

The work [12] presented the computational complexity of a MATLAB implemented Newton-Euler formulation in terms of the floating point operations needed for inverse dynamics (ID) computations. The paper reports 694 and 273 flops to compute forces on each leg and for moving platform respectively and a total number of operations of 4889 for ID, whereas a 6 degree-of-freedom serial manipulator requires only 1454 flops. The results showed that leg inertia contributes 20%-50% of the total demanded force, which further increases with increased operational speeds and decreased payload mass. Moreover, an attempt to qualitatively compare the time complexities of actuator and configuration-space formulations was presented in [42] and the implementation was done in *Mathematica*. Expressions for theoretical, computational costs and the runtime to compute these expressions were also presented. It is reported that the configuration-space dynamic model is computationally less intensive than actuator-space formulation. A parallel processing approach to implement resolved Newton-Euler formulation for model-based control using a network of microprocessors is discussed in [22], which uses a 16 bit host PC with four neighbours connected. Sampling time of 0.66 ms was achieved with PUMA 560 serial manipulator. The inverse dynamics computations were reported to take 464.2 μ s.

Jordan's principle of virtual work was followed in [1]. The work formulates the dynamics model including all the friction effects. The proposed algorithm has in total of 1987 operations to generate the control action in 0.15 ms at a sampling rate of 0.5 ms. A circle tracker was implemented with an end-effector velocity of 1 m/s with real-time control enabled. It was shown that friction compensation could improve control accuracy significantly. As mentioned in [1], the other benchmarks in the literature, [20] and [27], have reported the 2150 and 2078 computations respectively. Comparison of various methods is as shown in Table 1.1 for the computation of the proposed kinematic and inverse dynamic models. Further, for their specific application of Hexaglide manipulator, 202 μ s for a sampling period of 900 μ s with 3651 computations for dynamics compensation in the control loop was reported in [23].

Table 1.1: Comparison of the number of floating point operations of various methods, reproduced from [27]

	Kinematic Modelling	Inverse Dynamic Model	Total
Method-1	659 ‘*’, 299 ‘+’	631 ‘*’, 489 ‘+’	2078
Method-2	659 ‘*’, 299 ‘+’	715 ‘*’, 609 ‘+’	2282
Gosselin	468 ‘*’, 282 ‘+’	834 ‘*’, 566 ‘+’	2150
Dasgupta	-	-	4489

1.5 Overview of the report

Chapter 2 covers the basic notations and formulations used in the report. Chapter 3 explores a few possible ways of computationally less intensive implementations to solve the root-tracking problem and different formulations of angular velocity Jacobian matrices. Chapter 4 deals with formulating and verifying dynamic models in three different sets of generalised coordinates. Useful aspects are extracted from these models, and an extended-configuration-space formulation and possible mapping to smaller dimensional spaces are discussed in detail in Chapter 5. Simulation, comparison and C++ implementation of these models constitute Chapter 6. An example problem, dynamic simulation of the 6-RSS manipulator is also presented. The best model is selected and is used for trajectory tracking control in Chapter 7. The report ends with possible future extensions and conclusion in Chapter 8.

1.6 Summary

An important point to note from the literature is that the requirements of accuracy of the model and real-time simulation are opposed in nature. The model is generally simplified to meet realistic time constraints, and the errors resulting from such simplifications are compensated using control algorithms. Therefore in this report, various formulations of dynamics model are explored to find the right balance between fidelity and computational time. Moreover, Lagrangian formulation and simulation of the complete system (SRSPM) followed by control seems to be missing in the literature which is addressed in the current report. The current chapter is a brief survey on literature present broadly on the topics of dynamics, simulation, control implementation in SRSPM and

an overview of the report structure.

CHAPTER 2

Dynamics models of parallel manipulators using the Lagrangian formulation

This chapter contains the basics definitions and notations used in the report followed by the formulation of three variants of dynamics using the Lagrangian equation of motion namely, *configuration-space*, *actuator-space* and *task-space*.

2.1 Basic definitions and notations

The architecture parallel manipulators consists of both *active*, i.e., active and *passive*, i.e. non-actuated joints. These variables are related by *constraint equations* resulting from the condition that all the closed loops in the manipulator should remain intact at all times. Such constraints are called *loop-closure* equations and are defined in terms of the *configuration variables*, \mathbf{q} , as:

$$\boldsymbol{\eta}(\mathbf{q}) = \mathbf{0}, \quad (2.1)$$

where $\mathbf{q} = [\boldsymbol{\theta}^\top, \boldsymbol{\phi}^\top]^\top$, and $\boldsymbol{\theta} \in \mathbb{R}^n$, $\boldsymbol{\phi} \in \mathbb{R}^m$ are the active and passive variables¹ of the manipulator respectively.

Since the loop-closure equations do not change with time:

$$\frac{d\boldsymbol{\eta}}{dt} = \mathbf{J}_{\boldsymbol{\eta}\dot{\mathbf{q}}} \dot{\mathbf{q}} = \mathbf{0}, \quad \text{where } \mathbf{J}_{\boldsymbol{\eta}\dot{\mathbf{q}}} = \frac{\partial \boldsymbol{\eta}}{\partial \dot{\mathbf{q}}}. \quad (2.2)$$

The above, Eq. (2.2) can further be written in terms of the active and passive variables as:

$$\mathbf{J}_{\boldsymbol{\eta}\boldsymbol{\theta}} \dot{\boldsymbol{\theta}} + \mathbf{J}_{\boldsymbol{\eta}\boldsymbol{\phi}} \dot{\boldsymbol{\phi}} = \mathbf{0}, \quad \text{where } \mathbf{J}_{\boldsymbol{\eta}\boldsymbol{\theta}} = \frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\theta}}, \quad \mathbf{J}_{\boldsymbol{\eta}\boldsymbol{\phi}} = \frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\phi}}. \quad (2.3)$$

¹In the following sections the *task-space* variables or the *end-effector* coordinates are also included in $\boldsymbol{\phi}$, as they are also unactuated as well as unknown.

The matrices $\mathbf{J}_{\eta q}$, $\mathbf{J}_{\eta \phi}$ are called as the *configuration Jacobian matrix* and the *constraint Jacobian matrix* and note that the numerical evaluation of these matrices is critical for the simulation of the dynamics model system.

2.2 Lagrangian formulation of the equation of motion

The following are the assumptions under which the formulations remain valid:

1. All the links are assumed to be rigid,
2. There is no friction, and all the joints are ideal,
3. The mass distribution of all the links is uniform.

Since parallel manipulators are *constrained mechanical systems*, the equations of motion are obtained using the *constrained Lagrangian* formulation, which is described widely in literature, such as, [18, 54]. The constraint Lagrangian is formulated as:

$$\mathcal{L}' = \mathcal{L} + \boldsymbol{\eta}^\top \boldsymbol{\lambda}.$$

The Euler-Lagrange equation applied on the formulated Lagrangian results in the constrained Lagrangian equation of motion which is of the form:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{G} = \mathbf{Q}^{\text{nc}} + \mathbf{J}_{\eta q}^\top \boldsymbol{\lambda}, \quad (2.4)$$

where, \mathbf{M} is the *generalised mass matrix*, \mathbf{C} is the combination of centripetal and Coriolis forces, \mathbf{G} is the vector of potential forces, \mathbf{Q}^{nc} is the vector of external forces, and $\mathbf{J}_{\eta q}^\top \boldsymbol{\lambda}$ is the vector of *constraint forces*, arising out of the loop-closure constraints. Eliminating the Lagrangian multipliers, $\boldsymbol{\lambda}$ and formulation of the task-space dynamics is discussed in the subsequent sections.

Note that since the formulation is done along with the constraints, *any* set of *generalised coordinates*, even if not completely independent can be used to formulate the systems dynamics.

2.3 Configuration-space formulation

The set of coordinates used to describe the system in this space is, $\mathbf{q} = [\boldsymbol{\theta}^\top, \boldsymbol{\phi}^\top]^\top$, i.e., it consists of both the active and passive joint variables. In this case, $\boldsymbol{\lambda}$ can be obtained by considering the time derivative of the Eq. 2.2:

$$\mathbf{J}_{\eta q}\ddot{\mathbf{q}} + \dot{\mathbf{J}}_{\eta q}\dot{\mathbf{q}} = \mathbf{0}. \quad (2.5)$$

Substituting for $\ddot{\mathbf{q}}$ from the Eq. 2.4, $\boldsymbol{\lambda}$ is obtained as:

$$\boldsymbol{\lambda} = -\mathbf{A}^{-1} \left(\dot{\mathbf{J}}_{\eta q}\dot{\mathbf{q}} + \mathbf{J}_{\eta q}\mathbf{M}^{-1}\mathbf{f} \right), \text{ where} \quad (2.6)$$

$$\mathbf{A} = \mathbf{J}_{\eta q}\mathbf{M}^{-1}\mathbf{J}_{\eta q}^\top, \quad (2.7)$$

$$\mathbf{f} = \mathbf{Q}^{\text{nc}} - \mathbf{C}\dot{\mathbf{q}} - \mathbf{G}. \quad (2.8)$$

Using the above equations, forward dynamics model is of the form:

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}\mathbf{f} - \mathbf{M}^{-1}\mathbf{J}_{\eta q}^\top\mathbf{A}^{-1} \left(\dot{\mathbf{J}}_{\eta q}\dot{\mathbf{q}} + \mathbf{J}_{\eta q}\mathbf{M}^{-1}\mathbf{f} \right). \quad (2.9)$$

Refer to [41] for an in-depth discussion on \mathbf{A} and Jacobian matrices.

2.4 Actuator-space formulation

The generalised coordinates used in this formulation are limited to the actuated variables. In this case $\boldsymbol{\lambda}$ is eliminated using the fact that the constraint forces represented by $\mathbf{J}_{\eta q}^\top\boldsymbol{\lambda}$, do not do any work and hence:

$$(\mathbf{J}_{\eta q}^\top\boldsymbol{\lambda})^\top\dot{\mathbf{q}} = \boldsymbol{\lambda}^\top\mathbf{J}_{\eta q}\dot{\mathbf{q}} = 0.$$

Further using, $\dot{\mathbf{q}} = \mathbf{J}_{q\theta}\dot{\boldsymbol{\theta}}$, the equation of motion can be reduced into:

$$\mathbf{M}_\theta\ddot{\boldsymbol{\theta}} + \mathbf{C}_\theta\dot{\boldsymbol{\theta}} + \mathbf{G}_\theta = \mathbf{Q}_a^{\text{nc}}, \quad (2.10)$$

\mathbf{M}_θ , \mathbf{C}_θ and \mathbf{G}_θ are analogues of the *coefficient matrices* derived in Eq. 2.4 of configuration-space dynamic model. It should be noted that the process has achieved a change of co-

ordinates i.e., *mapping* the dynamic model from one set of variable space to the other, which is a recurring theme in the *extended-configuration-space* formulation. Note that such a mapping is made possible in the actuator space by the $J_{\eta\phi}$ matrix and hence is valid only on the existence of its inverse. The singularity caused by the constraint Jacobian matrix is termed as the gain-type singularity.

2.5 Task-space formulation

The set of variables representing the output space of a manipulator is called the task-space, represented by \mathbf{x} . The task-space generally consists of, the Cartesian coordinates, $\{x, y, z\}$ of the geometric centre and the Rodrigues' parameters, $\{c_1, c_2, c_3\}$, related to the orientation of the moving platform. The form of equation of motion is similar to actuator-space model:

$$\mathbf{M}_x \ddot{\mathbf{x}} + \mathbf{C}_x \dot{\mathbf{x}} + \mathbf{G}_x = \mathbf{Q}_a^{nc}, \quad (2.11)$$

\mathbf{M}_x , \mathbf{C}_x , \mathbf{G}_x are the coefficient matrices of the task-space dynamics, which are functions of the task-space variables only.

2.6 Forward Kinematics

The relationship between the passive variables and the actuated variables is described by the forward kinematic (FK) map. For a parallel manipulator, this map might not be unique, i.e., for a given set of actuator variables there might be more than one feasible configuration of the manipulator.

2.7 Inverse Kinematics

The inverse kinematics (IK) deals with the problem of finding the actuator and passive variables given the task-space variables. In parallel manipulators, each of the legs are independent and generally have simple serial linkage connections, which simplifies the IK problem.

CHAPTER 3

Elements of kinematic modelling of parallel manipulators

3.1 Introduction

There are several formulations to arrive at the FK solutions for the SRSPM, as given by [33], [34], [10] etc. However, as mentioned in [42], solving the FK problem involves arriving at the forward kinematic univariate (FKU), which requires solving a 20 degree polynomial for an SPM, which might be computationally expensive. As discussed in Section 2.4, the actuator-space dynamic model is obtained from the configuration-space coefficients and hence is a function of q . So, this requires solving the FK problem at every instant of dynamics simulation to find the passive angle values. Moreover, each FK solutions are not unique, and hence the required solution needs to be tracked. In this section, the implementation of root-tracking and the forward kinematics problem in the context of the SRSPM are discussed. The inverse kinematic (IK) relations used in the current work are derived as described in [4].

3.1.1 Convention for representing the moving platform

In an SRSPM, the length of the i th prismatic joint is represented as l_i , and (x, y, z) represent the Cartesian coordinates (position) of the top plate. Rodrigues' parameterisation, (c_1, c_2, c_3) is used for describing the orientation of the moving platform. The angles at the i th universal joint is represented by (ϕ_i, ψ_i) , $i = 1, 2 \dots, 6$ which are rotations about the y and the x axes, respectively. Therefore, the total number of configuration-space and task-space variables are 18 and 6, respectively.

The base and the moving platforms of the manipulator are represented by hexagons circumscribed by a circle of radius r_b and r_t respectively the angle between alternate consecutive vertices being γ_b and γ_t . The convention followed in [6] is as shown in Fig. 3.1.

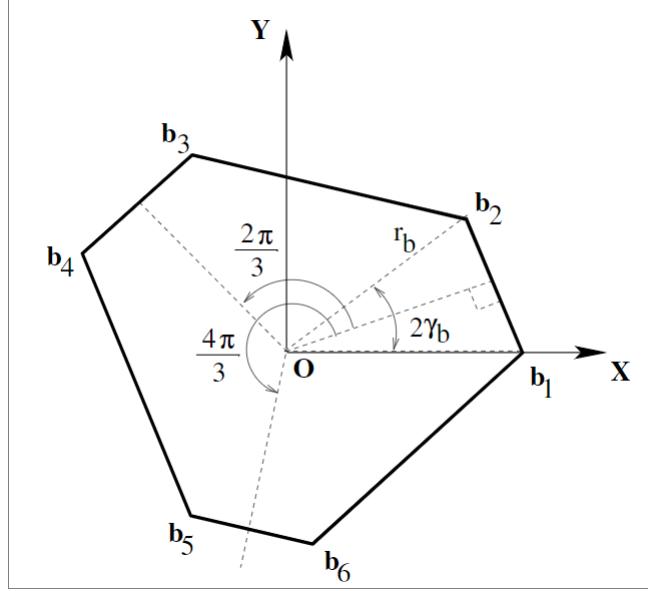


Figure 3.1: Base and axis representation followed in and reproduced from [6]

In the current work, the relative orientation of the base and moving platforms at $t = 0$ are such that their sides are parallel to each other as shown in Fig. 3.2. This orientation can be achieved by rotating the right-handed reference frames attached to the top and fixed platforms in Fig. 3.1 by $\frac{2\pi}{3} - 2\gamma_t$ and $\frac{2\pi}{3} - 2\gamma_b$ ¹ respectively.

3.1.2 Root-tracking methods

At every time step of the dynamics simulation the actuator variables corresponding to the current instant are obtained by integrating the equation of motion, Eq. (2.4) up to the current time step. For a given set of actuator variables, there might be several possible branches of solutions for the FK problem. Hence, in reality, the current branch needs to be tracked among all the solutions obtained.

Nearest neighbour method

In this method, the FK problem is solved and the obtained solutions are compared with solution of the previous time step to select the appropriate branch.

In method 1, Φ^{i-1} is the selected branch at previous instant and Φ_b^i represents the b^{th} branch of the current time step i . The operations max and min returns the maximum, and minimum value of a given list and $|\cdot|$ returns the absolute value of each element in

¹Units for all the angles are in radians

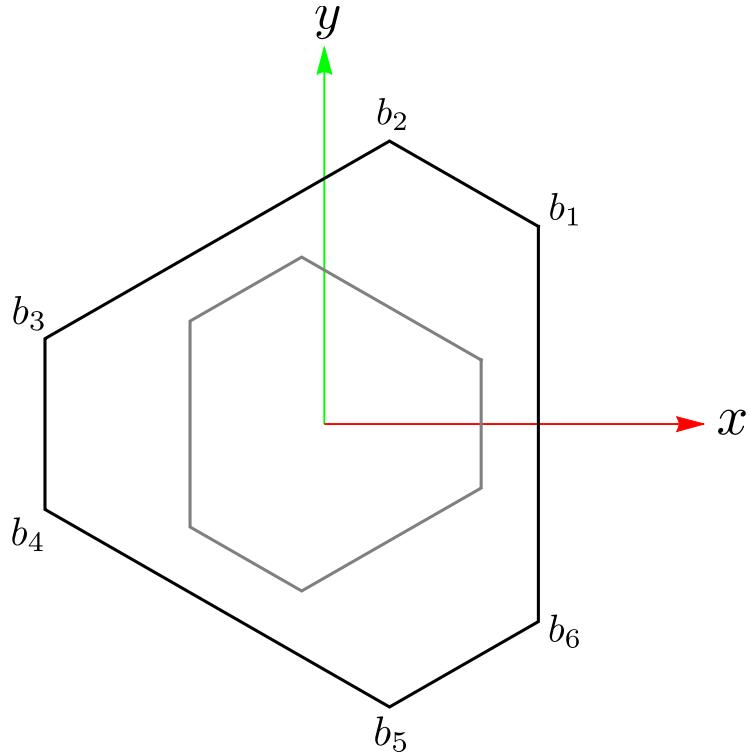


Figure 3.2: Relative orientations of the base and the moving platform with respect to the global frame of reference used in this report

Method 1 Root-tracking method using the nearest neighbour method

- 1: Obtain θ^i integrating the equation of motion, Eq. (2.4), up to the i^{th} time step
 - 2: $\phi^i \leftarrow$ All the solutions of the FK problem
 - 3: **if** $\phi_k^i \in \mathcal{S}^1$, where $k = 1, 2 \dots, m$ **then**
 - 4: $\Phi \leftarrow$ Map each ϕ_k^i into the range of $[0, 2\pi]$
 - 5: **end if**
 - 6: **for** $b \leq \text{number_of_branches}$ **do**
 - 7: $\Psi_b \leftarrow \max(|\Phi_b^i - \Phi^{i-1}|)$
 - 8: **end for**
 - 9: Select the branch b corresponding to $\min(\Psi_b)$,
-

a given list.

In the context of the FK problem, this method is computationally intensive as it is required to find all the roots at each time step, and the solutions corresponding to each branch compared. However, in reality, for dynamic simulation, only one set of results corresponding to the initial conditions of the physical system is required. Therefore a method that computes only the necessary set of solutions at every instant is needed.

Integration of first-order form of the constraint equations

Consider the loop-closure equations given in Eq. (2.1), consider its first order derivative given in Eq. (2.3). Writing the equation in terms of $\dot{\phi}$ gives:

$$\dot{\phi} = \mathbf{J}_{\phi\theta}\dot{\theta}, \quad \text{where } \mathbf{J}_{\phi\theta} = -\mathbf{J}_{\eta\phi}^{-1}\mathbf{J}_{\eta\theta}, \det(\mathbf{J}_{\eta\phi}) \neq 0. \quad (3.1)$$

Substituting the values of actuator-space variables obtained by the integration of the equation of motion, Eq. (2.4), until the previous instant in the Eq. (3.1) leads to an initial value problem which, when solved from the previous time step to the current, gives the required passive joint angle values.

Method 2 Root-tracking by integration of first order constraint

- 1: Obtain θ^i integrating the equation of motion, Eq. (2.4), up to the i^{th} time step
 - 2: Substitute θ^i into Eq. (3.1) and obtain a differential equation only in variables, ϕ
 - 3: With ϕ^{i-1} as initial condition, integrate equation obtained in step-2 until the next time step to obtain ϕ^i
-

As opposed to the nearest neighbour method, only the required branch is tracked in this algorithm. Since this technique uses integration at each time step, there is a trade-off between accuracy and the time taken to compute the solutions. With time, the calculated values diverge, violating the loop-closure constraint if not enforced explicitly.

Newton-Raphson based root-tracking

In this method, the Newton-Raphson (NR) method is employed at each time step to ensure that the constraint equations are not violated. In method 3, $\boldsymbol{\eta}$ and \mathbf{J} represent

Method 3 Newton-Raphson based root-tracking

- 1: Obtain θ^i integrating the equation of motion, Eq. (2.4), up to the i^{th} time step
 - 2: $\boldsymbol{\eta}(\phi)$ and $\mathbf{J}(\phi) \leftarrow \theta^i$ substituted in $\boldsymbol{\eta}(\theta, \phi)$ and $\mathbf{J}(\theta, \phi)$
 - 3: $\boldsymbol{\eta} \leftarrow$ Substitute ϕ^{i-1} in $\boldsymbol{\eta}(\phi)$
 - 4: **while** $\max(|\boldsymbol{\eta}|) \geq \epsilon$ **and** $\text{loopcount} \leq \text{max_iterations}$ **do**
 - 5: $\text{loopcount}++$
 - 6: $\mathbf{J} \leftarrow$ Substituting ϕ^{i-1} in $\mathbf{J}(\phi)$
 - 7: $\delta\phi \leftarrow$ Solution of $\mathbf{J} \delta\phi = \boldsymbol{\eta}$
 - 8: $\phi^i \leftarrow \phi^{i-1} - \delta\phi$
 - 9: $\boldsymbol{\eta} \leftarrow$ Substituting ϕ^i in $\boldsymbol{\eta}(\phi)$,
 - 10: **end while**
-

the constraints and its Jacobian matrix respectively, and ϵ is the desired numerical zero

i.e., any value a is considered to be zero if, $|a| \leq \epsilon \in \mathbb{R}^+$.

The advantages of this method are:

1. Ability to track only the required branch among all the solutions
2. Ability to tune the accuracy of the solutions
3. Ability to obtain solutions that strictly satisfy the loop-closure constraints

A strategy used by [5] is to use the analytical FK formulation whenever the root tracking fails and continue tracking using one of the above methods from the next time step. It is to be noted that all of the above methods fail when a manipulator passes through a singularity, where any two or more of the forward kinematic branches merge. For the current scope of simulations, it is assumed that the manipulator is working within the Safe Working Zone (SWZ), as defined in [26]. Given the advantages of the NR method based root-tracking, this method is used for all the subsequent simulations.

3.2 Formulations of velocity Jacobian matrices

In this section, two ways to arrive at expressions for the velocity Jacobian matrices is discussed.

Given the velocity of three points of the moving platform

This method is reproduced from the appendix A of the book by J. Angeles, [2], to compute the angular velocity Jacobian matrices given velocities of three non-collinear points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ on a rigid body. Let the centroid of these three points be:

$$\mathbf{p} = \frac{1}{3} \sum_{i=1}^3 \mathbf{p}_i.$$

Then the velocity of the centroid would be,

$$\dot{\mathbf{p}} = \frac{1}{3} \sum_{i=1}^3 \dot{\mathbf{p}}_i.$$

Given velocity of the centroid, position of the points and angular velocity of the rigid body, the velocity of any given three points can be found as,

$$\dot{\mathbf{p}}_i = \dot{\mathbf{p}} + \boldsymbol{\Omega}(\mathbf{p}_i - \mathbf{p}) \quad i = 1, 2, 3, \quad (3.2)$$

where $\boldsymbol{\Omega}$ is the skew-symmetric matrix corresponding to the angular velocity vector $\boldsymbol{\omega}$.

Define a matrix \mathbf{P} as,

$$\mathbf{P} = [\mathbf{p}_1 - \mathbf{p} \mid \mathbf{p}_2 - \mathbf{p} \mid \mathbf{p}_3 - \mathbf{p}],$$

which on differentiating with respect to time gives,

$$\dot{\mathbf{P}} = [\dot{\mathbf{p}}_1 - \dot{\mathbf{p}} \mid \dot{\mathbf{p}}_2 - \dot{\mathbf{p}} \mid \dot{\mathbf{p}}_3 - \dot{\mathbf{p}}], \text{ then}$$

$$\dot{\mathbf{P}} = \boldsymbol{\Omega}\mathbf{P}, \quad \text{from Eq. (3.2).}$$

Multiply the Eq. (3.2) by $\boldsymbol{\omega}$ on both sides, this gives,

$$(\dot{\mathbf{p}}_i - \dot{\mathbf{p}})^\top \boldsymbol{\omega} = \mathbf{0}, \quad i = 1, 2, 3, \quad (3.3)$$

$$\dot{\mathbf{P}}^\top \boldsymbol{\omega} = \mathbf{0}. \quad (3.4)$$

From the theorem given in Appendix A of the book [2], given the above condition in Eq. (3.4), the angular velocity can be found as,

$$\begin{aligned} \mathbf{D}\boldsymbol{\omega} &= \dot{\mathbf{P}}^\vee, \\ \mathbf{D} &= \frac{1}{2}[\text{tr}(\mathbf{P})\mathbf{I} - \mathbf{P}], \\ \dot{\mathbf{P}}^\vee &= \frac{(\dot{\mathbf{P}} - \dot{\mathbf{P}}^\top)^\vee}{2}. \end{aligned}$$

Therefore $\boldsymbol{\omega}$ of the rigid body can be found by inverting the \mathbf{D} matrix,

$$\boldsymbol{\omega} = \mathbf{D}^{-1}\dot{\mathbf{P}}^\vee,$$

Given the positions of three non-collinear points of the moving platform

This method of computing the linear and angular velocity Jacobian matrices for SRSPM can be found in [4]. The idea is to first find the orientation, $\mathbf{R} \in \mathbb{SO}(3)$ of the rigid body, given three points on the body then finding the angular velocity as,

$$\boldsymbol{\Omega} = \mathbf{R}^\top \dot{\mathbf{R}}.$$

Since the coordinates of the moving platform of the SRSPM are a function of the configuration-space variables, this method is used to compute the angular velocity of the moving platform in this report.

3.3 Summary

Three methods for root-tracking are illustrated, and their advantages are discussed. The nearest-neighbour method requires comparison of all the branches whereas integration and NR method tracks the roots corresponding to the required branch only. Subsequently NR method is chosen for root-tracking as gives direct control on the precision of the solution. Two alternatives for the computation of angular velocity are shown.

CHAPTER 4

Formulation of the dynamics models

4.1 Introduction

The current chapter briefly summarises the implementation of three popular formulations of dynamics models followed by validation of the numerical results obtained. Common implementation issues in modelling that are not identified by these verifications checks are illustrated. Let the SRSPM fall freely from a pose only under the influence of gravity starting from rest, i.e., $\dot{\mathbf{q}}(0) = \mathbf{0}$ with the following initial conditions in the task-space variables,

$$\mathbf{q}(0) = [0, 0, 1.1, 0, 0, 0]^\top.$$

All the dynamics models formulated are simulated for these conditions for numerical validation of the models. For a complete derivation of the equations of motion for a given system using constrained Lagrangian formulation can be found in standard texts like [19] or [18], and the details are skipped here.

4.2 Formulation the of the dynamics model in the configuration-space

For manipulators such as SRSPM or the 6-RSS one cannot obtain a closed form solution for the FK problem and hence one cannot reduce all the passive joint angles in terms of the actuator coordinates. In parallel manipulators, the set of active and passive joint coordinates together are called as the configuration-space of the manipulator and are represented by,

$$\mathbf{q} = [\boldsymbol{\phi}^\top, \boldsymbol{\theta}^\top]^\top,$$

where ϕ is the vector of passive variables and θ is the vector of active variables as discussed in Section 2.3 and the equation of motion is of the form, Eq. (2.4). Let, \mathbf{b}_i and \mathbf{a}_i be the vectors corresponding to the i^{th} vertex of the base and moving platforms, respectively, in the global frame. Then,

$$\mathbf{a}_i = \mathbf{b}_i + \mathbf{R}_y(\phi_i) \mathbf{R}_x(\psi_i) \mathbf{l}_i, \quad \text{where } i = 1, 2 \dots 6, \quad (4.1)$$

where \mathbf{l}_i represents a vector along the i^{th} leg with a magnitude equal to that of its length, and \mathbf{R}_x and \mathbf{R}_y are the rotation matrices about x and y axes, respectively. A set of constraint equations are obtained by imposing the conditions of the rigidity of the moving platform as given in [6], as explained below,

1. Relative distances between the vertices are fixed, giving rise to a set of 6 equations;
2. Forming rigid triangles by placing a constraint on the distance between three consecutive points gives 3 more equations;
3. Ensuring that the axis of all the planes of the above formed triangles parallel, i.e., all the triangles are in the same plane results in 3 additional equations.

These together form 12 constraints which are functions of 18 variables, i.e., 6 actuator-space variables, θ and 12 passive angle variables, ϕ . For the complete formulation of configuration-space dynamic model, one may refer to [4].

Verification of the model for mathematical consistency

The symbolic expressions for the coefficients of the equation of motion of SRSPM are derived in *Mathematica*[®] 11.2. The inbuilt `Compile` function was used to optimise each of the expressions for real-valued input, which improves the computation time significantly. These expressions are then converted into a C++ compatible code. Refer to Appendix C for greater details.

Numerical error measures to validate mathematical consistency of the dynamic model of a parallel manipulator was given in [41] by considering the norm of zeroth, first and second order forms of the constraint equations as given in Eq. (2.1). In the current work instead of norm of the errors, the corresponding maximum absolute error is considered as the measure. The error plots corresponding to the data generated from *Mathematica*[®] 11.2 for the free-fall simulation are presented below.

1. Zeroth-order check on the loop-closure constraint (e_1) :

This measure quantifies the deviation from the loop-closure equations in Eq. (2.1) vs time,

$$e_1 = \max(|\boldsymbol{\eta}(\mathbf{q})|). \quad (4.2)$$

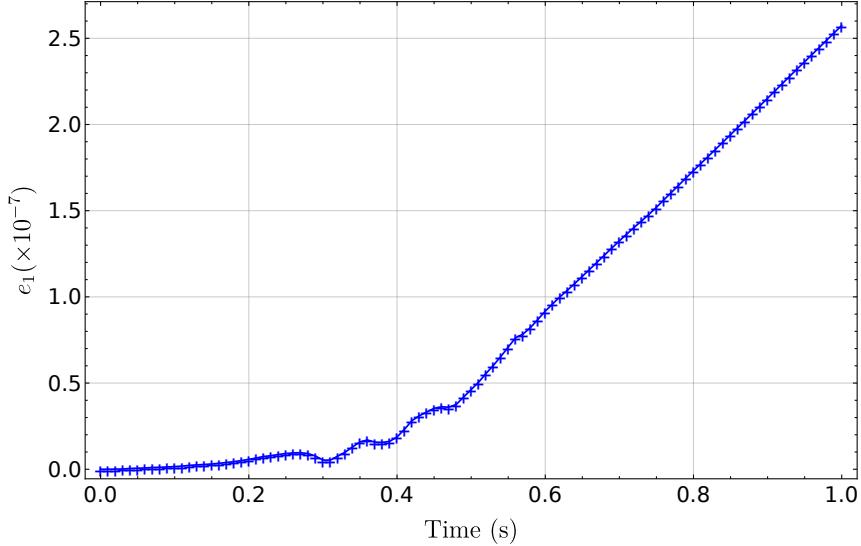


Figure 4.1: Absolute maximum error in loop-closure equations with time as given in Eq. (4.2)

2. First-order check on loop-closure constraint (e_2) :

Consider the deviation of first order derivative of the loop-closure constraints with respect to time,

$$e_2 = \max(|\mathbf{J}_{\eta q} \dot{\mathbf{q}}|). \quad (4.3)$$

This verifies the consistency of the joint velocities with the loop-closure constraint in the *Pfaffian form*.

3. Second-order check on the loop-closure constraint (e_3) : The consistency of the generalised accelerations with the loop-closure constraint is verified by considering its second order derivative with respect to time.

$$e_3 = \max(|\mathbf{J}_{\eta q} \ddot{\mathbf{q}} + \dot{\mathbf{J}}_{\eta q} \dot{\mathbf{q}}|). \quad (4.4)$$

4. Conservation of Energy (e_4) :

The total mechanical energy of the manipulator, $E(t)$ is given as,

$$E(t) = T(t) + V(t), \quad (4.5)$$

where $T(t)$ and $V(t)$ represent the kinetic and potential energy, respectively, of the manipulator, since the current simulation is a free-fall case, there is no actuation and hence the total energy should remain constant. We check the percentage

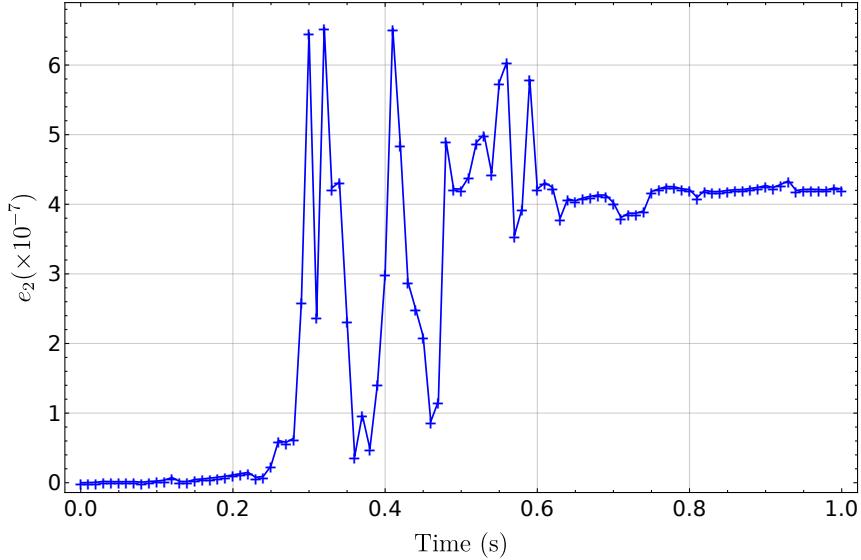


Figure 4.2: Absolute maximum of first order loop-closure equations with time as given in Eq.(4.3)

change in energy of the system for the entire time given as:

$$\delta E = \frac{E(t) - E(0)}{E(0)} \times 100.$$

Where $E(0)$ is the energy of the system at time $t = 0$.

A note on the mathematical consistency checks

The checks mentioned in Section 4.2 do not necessarily mean that the system modelled is the *actual* system; it only ensures that the formulation is mathematically consistent. Common sources of inaccuracies involve mismatch of inertia or mass of the elements in the model vs the physical system. These checks also fail to catch any representation errors. Say, if the local frame of the moving platform of the SRSPM is defined with a constant offset in its orientation from the global frame, in such cases, even if the simulation data is not consistent with this representation, the model remains mathematically consistent. In other words, modelling errors done before the formulation of mass matrix remain mathematically consistent but might not represent the real intention of the simulation.

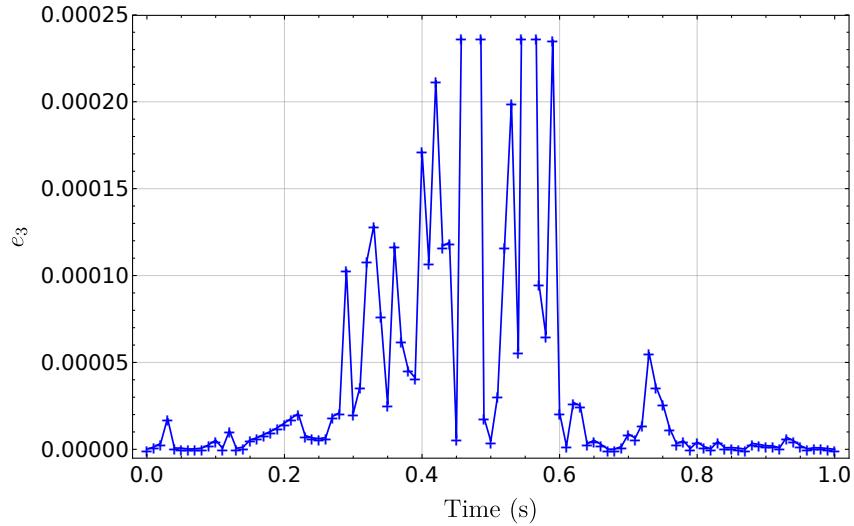


Figure 4.3: Absolute maximum of second order loop-closure equations with time as given in Eq.(4.4)

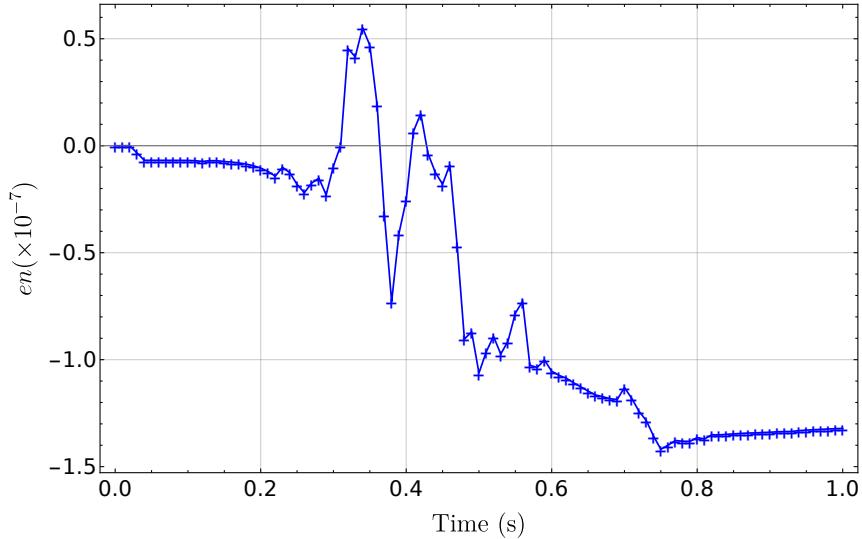


Figure 4.4: Percentage change in the total mechanical energy of the system with time as given in Eq. (4.5)

Size¹ of relevant matrices

The sizes of the coefficient matrices used in the dynamic simulation in C++ and Mathematica are as shown in Table 4.1.

¹Corresponds to the size of the expression computed using ByteCount function in *Mathematica*® 11.2 or the size of the file containing the expression in C++

Table 4.1: Size of the coefficient matrices used in configuration-space dynamic model of the manipulator

Expression	Size (MB)	
	C++	Mathematica
M	0.900	13.369
C	103.4	748.207

4.3 Formulation of the dynamics model in the actuator-space

Deriving the equation of motion in the actuator space is as defined in Section 2.4. Since there are no closed-form solutions of the FK problem, the dynamic model is formulated in a higher dimensional space, i.e., the configuration-space and is mapped to the actuator-space, resulting in an equivalent unconstrained dynamic system. The derivation of the same can be found in [43, 41].

Verification of the model for mathematical consistency

Since the M , C and G matrices consist of both passive and active variables; it is necessary to solve the FK problem at each time step and hence NR method based root tracking described in 3 is used for the same. It is ensured that each root is found with enough accuracy to keep the error in the constraint equation within the order of 10^{-15} . Since the manipulator attains a singular pose at time $t = 0.48$ s, where the inversion of $J_{\eta\phi}$ fails as discussed in Section 2.4, the simulation does not proceed after this time. Hence the numerical checks are also performed up to $t = 0.48$ s only. Fig. 4.5 shows the checks described in Section 4.2.

It is clearly seen that enforcing the loop-closure constraint explicitly has brought down the order of errors as compared to the configuration-space model which is shown in Section 4.2. However, the expressions of M_θ and C_θ matrices could not be obtained in closed form as they involve expensive symbolic inverse computations and hence are calculated numerically from the M and C matrices at each time step. An in-depth comparison between the formulations and their relative time complexities can be found in [42].

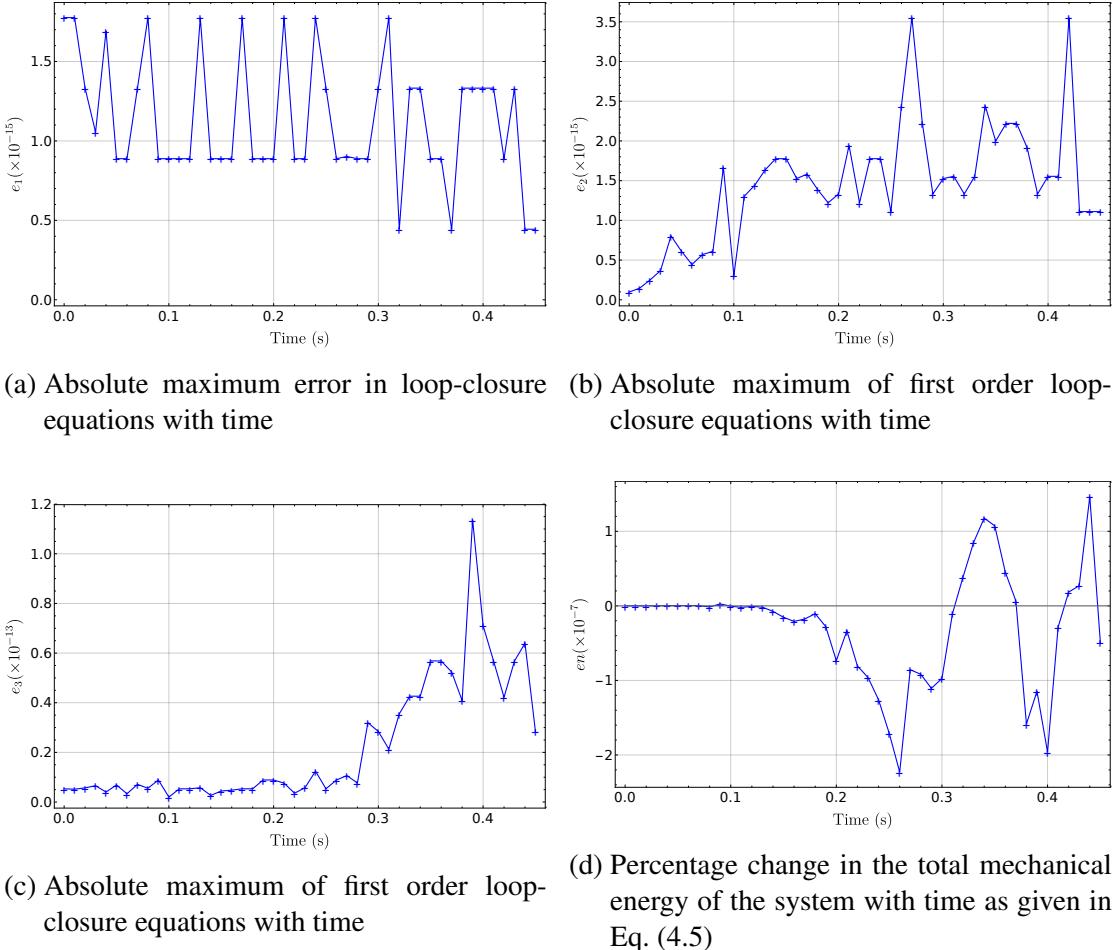


Figure 4.5: Validation of the actuator-space mathematical model

4.4 Formulation of the dynamics model in the task-space

The task-space dynamics model is derived as discussed in Section 2.5. In such a description of the system, expressions for angular and linear velocity Jacobian matrices of the moving platform will be more straightforward as compared to that of the legs. To formulate the dynamics completely in the task-space the IK solutions are used to relate the configuration-space variables \mathbf{q} and the task-space variables \mathbf{x} .

4.4.1 Implementation

It should be noted that it was not possible to obtain the closed-form expressions of the symbolic \mathbf{C} matrix as it is computationally intensive as already mentioned in [31] and hence is obtained numerically from the partial derivatives of the mass matrix as given

in Eq. 4.6, at each instant, increasing the execution time significantly.

$$C_{ij} = \frac{1}{2} \sum_{k=1}^{18} \left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{kj}}{\partial q_i} \right) \dot{q}_k \quad (4.6)$$

Size and computation time

The sizes of the coefficient matrices used in the dynamic simulation, in *Mathematica* are as shown in Table 4.2. A much wiser way to formulate the task-space dynamics

Table 4.2: Size of the coefficient matrices used in dynamics modelled in the task-space of the manipulator

Expression	Size (MB)
M	369.044
All the partial derivatives of M	18531.400

would be to solve the IK problem at each instant keeping the coefficients matrices as functions of the configuration-space variables. The error plots corresponding to the data generated from *Mathematica*[®] 11.2 for the free-fall simulation are presented in Fig. 4.6.

4.5 Summary

Dynamic models are formulated with three different sets of generalised coordinates. The relative sizes are given for comparison. A free-fall simulation of the SRSPM is done to check for numerical errors by measures defined by [41]. A few implementation details of the simulation in C++ is also presented. A note summarising the scope of the numerical checks is also given.

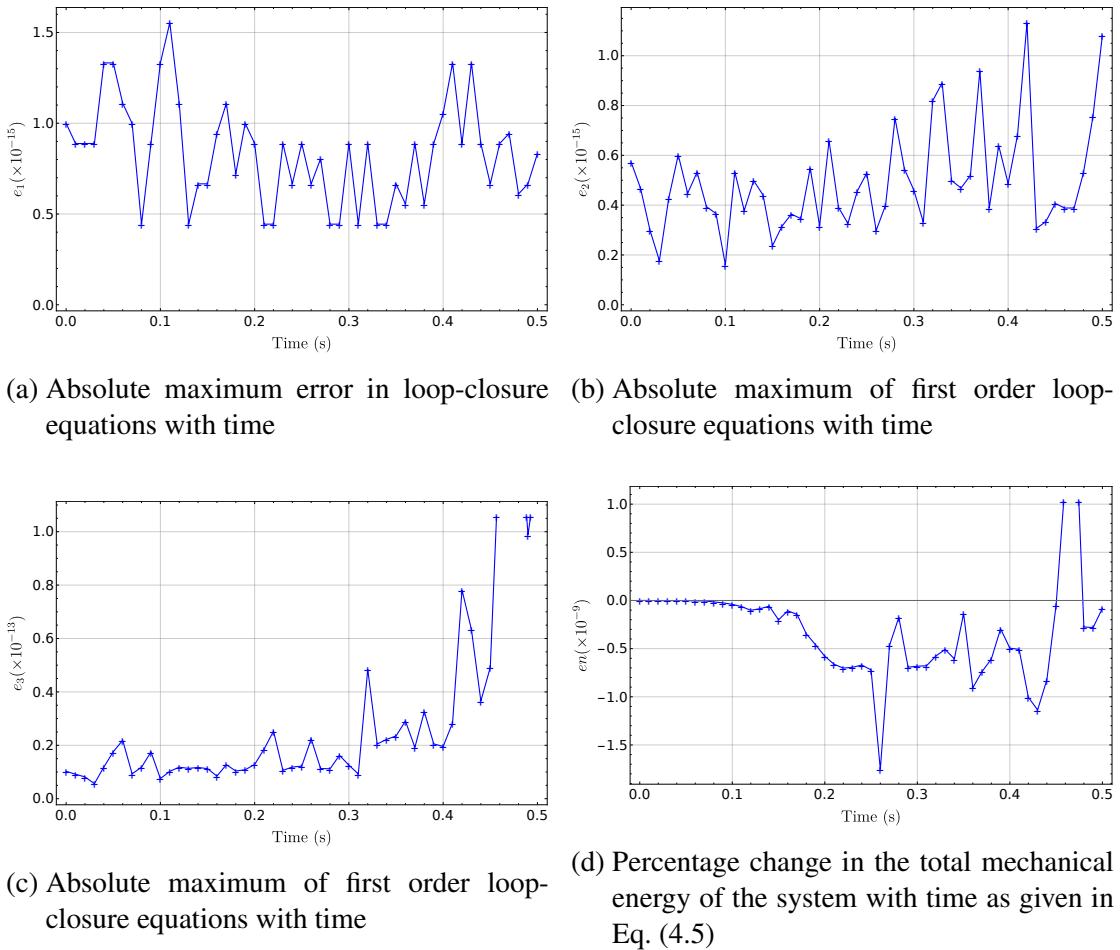


Figure 4.6: Validation of the task-space mathematical model

CHAPTER 5

Dynamics model in the extended-configuration-space

Observing the previous formulations, it is apparent that each method has relative advantages and disadvantages. The primary computational burden in the configuration-space formulation is due to the computation of the velocity Jacobian matrices of the moving platform, whereas in the task-space it is due to the Jacobian matrices of the prismatic links. In task-space, representing the centre of mass and orientation of the legs in terms of task-space variables; and in configuration-space, representation of the position and orientation of the moving platform in terms of configuration-space variables, make the expressions complicated.

5.1 Formulation of the dynamics model

The idea of extended-configuration-space formulation is to represent each body of the system in its *natural* coordinates, i.e., to describe all the elements with their associated variables. Such a description would avoid posing one set of variables in terms of others and thereby reducing the complexity of the entire system's description. In the case of SRSPM, all the variables together represent the extended-configuration-space.

$$\mathbf{q}_e = [\mathbf{x}^\top, \boldsymbol{\theta}^\top, \boldsymbol{\phi}^\top, \boldsymbol{\psi}^\top]^\top.$$

Another advantage is a significant reduction of complexity in the formulation of the constraint equations. For an SRSPM the constraints may be formulated by equating the positions of the vertices of the moving platform as reached from the centre of the moving platform and along the legs,

$$\mathbf{p} + \mathbf{R}_{tp}\mathbf{a}_{li} - \mathbf{a}_i = \mathbf{0}, \quad \text{where } i = 1, 2, \dots, 6, \quad (5.1)$$

and \mathbf{p} is the position of the geometric centre of the moving platform, $\{x, y, z\}$, \mathbf{R}_{tp} represents the rotation matrix of the moving platform, \mathbf{a}_{li} are the vertices of the moving

platform described in the local frame, and \boldsymbol{a}_i is as defined in Eq. (4.1).

Such a formulation of constraint equations drastically simplifies the expressions corresponding to the angular and linear velocity Jacobian matrices of the manipulator and is reflected in the execution time required. Since not all the equations contain all the variables, Jacobian matrices remain sparse reducing the computational time. Since there is no need to find the relations between different sets of variables (using FK or IK), it is easy to model the dynamics of a system in this method. The position and orientation of the moving platform are given in terms of the task-space variables, \boldsymbol{x} , as \boldsymbol{p} and \boldsymbol{R}_t and the rest in configuration-space, \boldsymbol{q} . Hence, the linear and angular velocity Jacobian matrices of the moving platform are given as,

$$\begin{aligned}\boldsymbol{J}_{\boldsymbol{v}_t} &= \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{q}_e}, \\ \boldsymbol{J}_{\boldsymbol{\omega}_t} &= \left(\boldsymbol{R}_t^\top \frac{\partial \boldsymbol{R}_t}{\partial \boldsymbol{q}_e} \right)^\vee,\end{aligned}$$

where $(\cdot)^\vee$ returns the corresponding vector form of a skew-symmetric matrix. The derivation of expressions for the centre of mass of the links can be found in [4] and is reproduced below for the sake of completeness.

$$\boldsymbol{p}_{bi} = \boldsymbol{b}_i + \boldsymbol{R}_y(\phi_i) \boldsymbol{R}_x(\psi_i) \boldsymbol{l}_{bi}, \quad \text{where } i = 1, 2 \dots 6, \quad (5.2)$$

$$\boldsymbol{p}_{ai} = \boldsymbol{b}_i + \boldsymbol{R}_y(\phi_i) \boldsymbol{R}_x(\psi_i) \boldsymbol{l}_{ai}, \quad \text{where } i = 1, 2 \dots 6, \quad (5.3)$$

where \boldsymbol{p}_{ai} and \boldsymbol{p}_{bi} are the positions and \boldsymbol{R}_l for orientation of the first and second link of i th prismatic joint computed as given in Section 3.2. The angular and linear velocity Jacobian matrices are given as,

$$\begin{aligned}\boldsymbol{J}_{\boldsymbol{v}_{ai}} &= \frac{\partial \boldsymbol{p}_{ai}}{\partial \boldsymbol{q}_e}, \\ \boldsymbol{J}_{\boldsymbol{v}_{bi}} &= \frac{\partial \boldsymbol{p}_{bi}}{\partial \boldsymbol{q}_e}, \\ \boldsymbol{J}_{\boldsymbol{\omega}_l} &= \left(\boldsymbol{R}_l^\top \frac{\partial \boldsymbol{R}_l}{\partial \boldsymbol{q}_e} \right)^\vee.\end{aligned}$$

Equation of motion

The total kinetic energy of the manipulator T is given as,

$$\begin{aligned} T_a &= \frac{1}{2} \sum_{i=1}^6 (\mathbf{J}_{v_{ai}}^\top m_{ai} \mathbf{J}_{v_{ai}} + \mathbf{J}_{\omega_l}^\top \mathbf{I}_{ai} \mathbf{J}_{\omega_l}); \\ T_b &= \frac{1}{2} \sum_{i=1}^6 (\mathbf{J}_{v_{bi}}^\top m_{bi} \mathbf{J}_{v_{bi}} + \mathbf{J}_{\omega_l}^\top \mathbf{I}_{bi} \mathbf{J}_{\omega_l}); \\ T_t &= \frac{1}{2} (\mathbf{J}_{v_t}^\top m_t \mathbf{J}_{v_t} + \mathbf{J}_{\omega_t}^\top \mathbf{I}_t \mathbf{J}_{\omega_t}); \\ T &= T_a + T_b + T_t. \end{aligned}$$

In the above T_a , T_b , T_t are the kinetic energies and m_{ai} , m_{bi} , m_t are masses of the legs and moving platform respectively. Assuming the plane containing the fixed platform as the reference, the total potential energy of the manipulator, denoted by V is computed as,

$$\begin{aligned} V_l &= \sum_{i=1}^6 (m_{ai} g z_{ai} + m_{bi} g z_{bi}) \\ V_t &= m_t g z_t \\ V &= V_l + V_t, \end{aligned}$$

where z_{ai} , z_{bi} , z_t are the z -coordinates of the legs and the top plate. Once the Lagrangian is attained, the equations of motion can be derived by using the Euler-Lagrange equations which would be of the form,

$$\mathbf{M}(\mathbf{q}_e) \ddot{\mathbf{q}}_e + \mathbf{C}(\mathbf{q}_e, \dot{\mathbf{q}}_e) \dot{\mathbf{q}}_e + \mathbf{G}(\mathbf{q}_e) = \mathbf{Q}_e^{\text{nc}} + \mathbf{J}_{\eta \mathbf{q}_e}^\top \boldsymbol{\lambda}, \quad (5.4)$$

The equations of motion are in the form of 24 coupled non-linear and differential equations, with the coefficient matrices being functions of the extended-configuration-space variables and their first-order time derivatives. This space alone does not convey much physical meaning as the prismatic joints are the only controllable variables. Since the formulation is done in a higher i.e., 24 dimensional space, there are multiple options to map i.e., either to actuator-space or to the task-space, both of which are 6 dimensional spaces. Note that this mapping is similar to the mapping from 18 dimensional configuration-space to the actuator-space. Further, such a mapping is not possible to

the configuration-space, as the mapping matrix, $J_{\eta x}$ is rectangular and hence non-invertible.

The sizes of the coefficient matrices used in the dynamic simulation, in C++ and *Mathematica*[®] 11.2, are as shown in Table 5.1. It is clearly seen that the order of

Table 5.1: Size of the coefficient matrices used in dynamics modelled in the extended-configuration-space mapped to task-space the manipulator

Expression	Size (KB)	
	C++	Mathematica
M	4.100	35.515
C	15.500	100.984

magnitude of the expression sizes is configuration was in MB whereas here it is in KB.

5.2 Extended configuration-space model mapped to task-space variables

Here the reduction of dynamics model from extended-configuration-space to the task-space unconstrained dynamics is illustrated.

$$\{x, \theta, \phi\} \mapsto \{x\}. \quad (5.5)$$

Consider the derivative of the constraint equations, Eq. (5.1) with respect to time,

$$\dot{q} = -J_{\eta q}^{-1} J_{\eta x} \dot{x}, \quad (5.6)$$

$$\text{where } J_{\eta x} = \frac{\partial \eta}{\partial x} \text{ and } J_{\eta q} = \frac{\partial \eta}{\partial q}. \quad (5.7)$$

From Eq.(5.7), it is apparent that this map fails when $J_{\eta q}$ is non-invertible, which leads to a configuration-space singularity as explained in [41]. The numerical checks described in Section 4.2 are used here and the results are shown in Fig. 5.1.

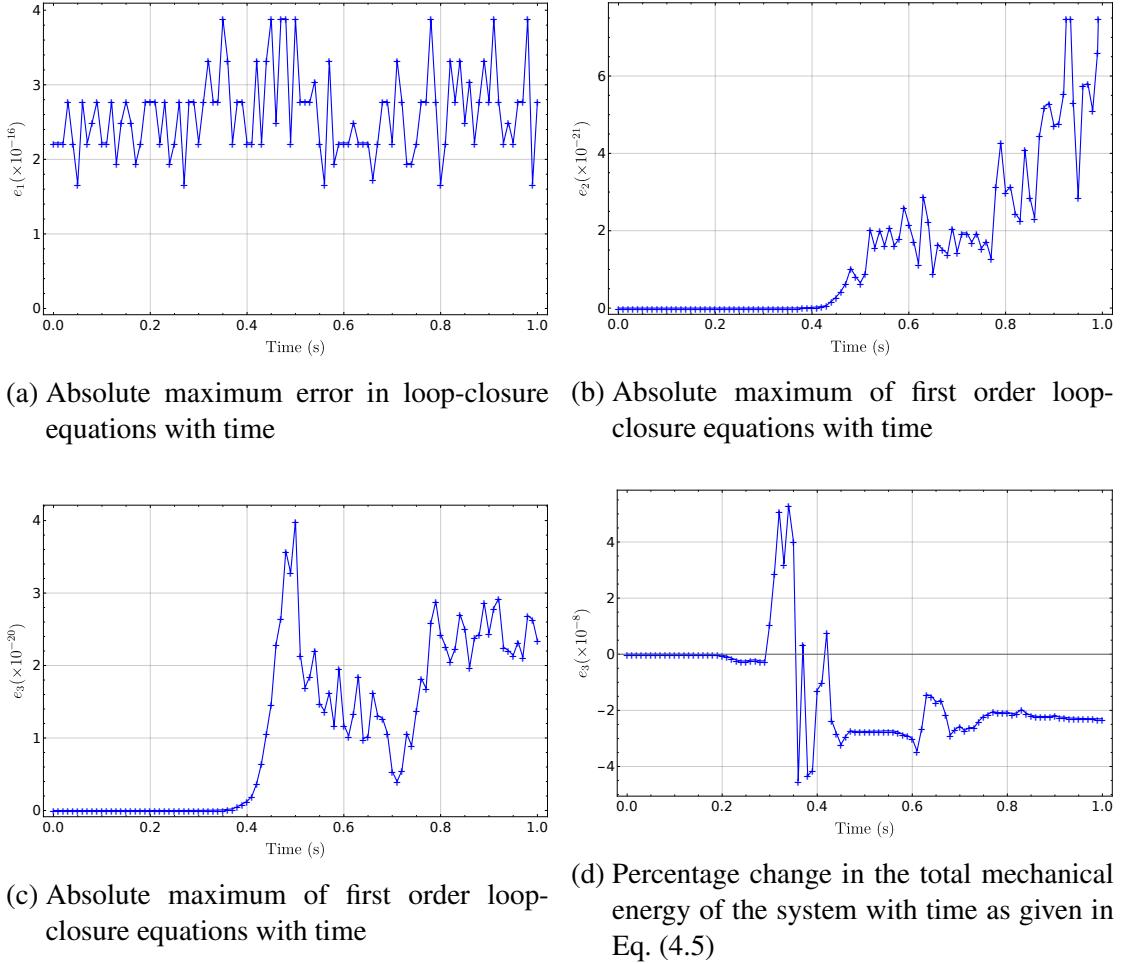


Figure 5.1: Validation of the extended-configuration-space mapped to task-space mathematical model

5.3 Extended configuration-space model mapped to actuator-space variables

In this case, the aim is to reduce the dynamic model from the extended-configuration-space to the actuator-space, to form a set of unconstrained equations of motion that would enable control of the manipulator.

$$\{x, \theta, \phi\} \mapsto \{\theta\}. \quad (5.8)$$

Let the set of task-space and passive variables be denoted as,

$$q_x = [x^\top, \phi^\top]^\top.$$

Consider the derivative of the constraint equations, Eq.(5.1), with respect to time,

$$\dot{q} = -J_{\eta q_x}^{-1} J_{\eta \theta} \dot{\theta}, \quad \text{where } J_{\eta \theta} = \frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\theta}} \text{ and } J_{\eta q_x} = \frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{q}_x}$$

From the above expression, it is apparent that this map fails when $J_{\eta q_x}$ is non-invertible which needs to be studied. The checks as given in Subsection 4.2 are performed and are in Fig. 5.2.

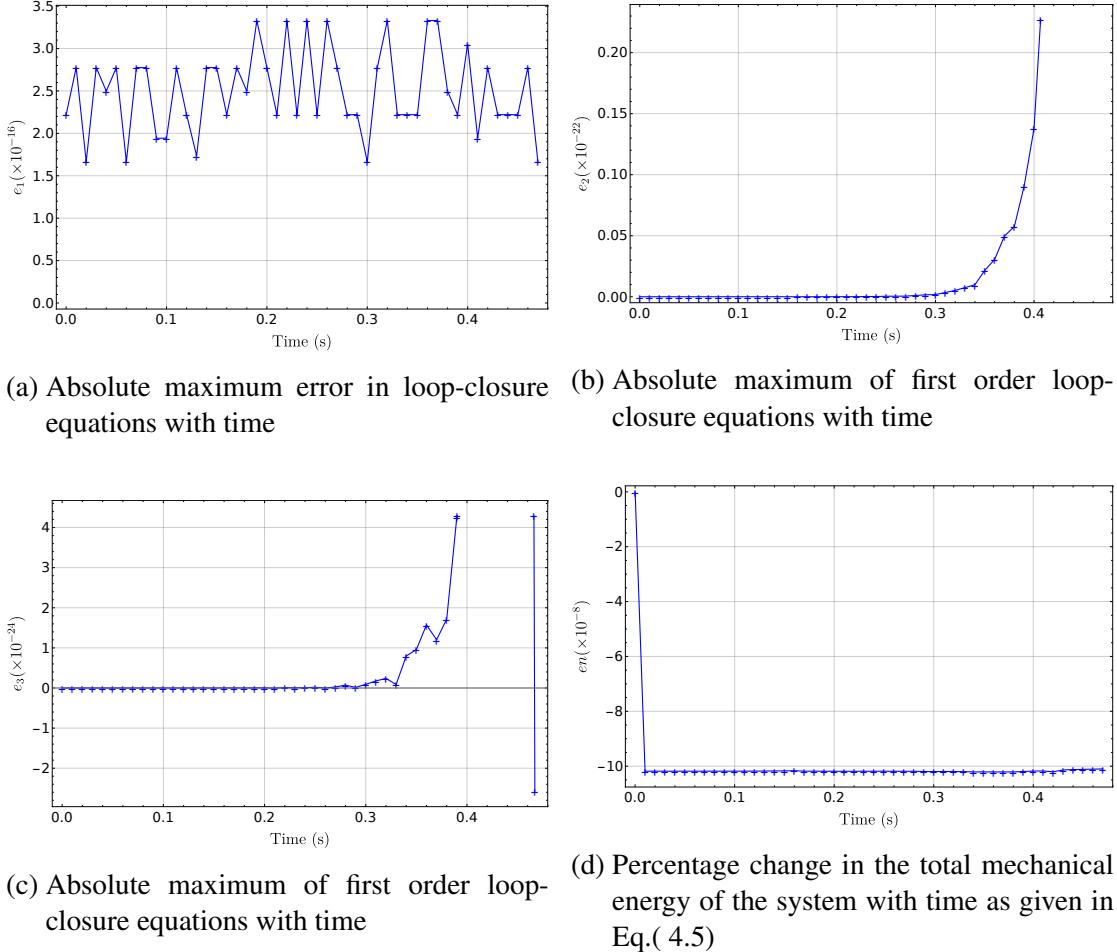


Figure 5.2: Validation of the extended-configuration-space mapped to actuator-space mathematical model

5.4 Summary

The extended-configuration-space dynamic model is formulated by taking in both the task-space and configuration-space variables together. The equation of motion is then mapped to task and actuator-spaces to obtain respective unconstrained dynamic models.

CHAPTER 6

Simulation and comparison of various formulations

6.1 Introduction

The dynamic models formulated in the previous chapters are compared with respect to their simulation times and sizes of the coefficient matrices in the current section. Consider a symmetric pose of the SRSPM, with no relative rotation of the local axis of the moving platform concerning the global frame. The simulation setting is as explained earlier, let the manipulator fall freely from such a pose only under the influence of gravity starting from rest, i.e., $\dot{\mathbf{q}}(0) = \mathbf{0}$ with the following initial conditions,

$$\mathbf{q}_e(0) = [0, 0, 1.1, 0, 0, 0]^\top.$$

All the formulations are compared against each other and are validated for this specific simulation data. The formulated dynamic models are simulated in both *Mathematica*® 11.2 and C++ for a simulation time of 1 s on an AMD Ryzen 7 1800X, 8 core CPU with 16 GB RAM and a clock speed of 3.6 GHz installed with Ubuntu 18.04.1 LTS. Please note that only one core is used for simulating the free-fall.

6.1.1 Dynamic simulations and observations

The formulated models are rewritten in state-space representation:

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

The inbuilt `NDSolve` function is used for integrating the equations of motion, with the parameter of `AccuracyGoal` set to 10^{-13} and `Adams` is used for numerical integration. The times taken for free-fall simulation in each case are as shown in Table 6.1. The time taken to solve the differential equations in *Mathematica*® 11.2 is recorded by using the `RepeatedTiming` function, with evaluation time set to a minimum of 60 s,

which returns the average time taken to run the code by running it at least for 60 s in a loop. All of the coefficient matrices are converted into compiled functions using the `Compile` command which optimises all the expressions for real-valued input, cutting down evaluation time significantly, from the order of days to seconds.

Table 6.1: Computational time taken for the free-fall simulation in *Mathematica*[®] 11.2

Formulation	Real-time (s)	Simulation time (s)
Configuration space	1.000	0.236
Actuator space	0.480	0.195
Extended mapped to task-space	1.000	0.241

It is to be noted that for the given initial conditions if one wishes to increase the Accuracy Goal to 10^{-15} , the configuration-space dynamics takes around 120 s simulation time for 1 second of real-time. This shows that not only the solver but also the initial conditions and accuracy requirements have a significant influence on the simulation time.

During the simulation of the actuator-space dynamics, the root-tracker encounters ill-conditioned matrices at $t = 0.48$ s, where the determinant of $J_{\eta\phi}$ matrix is 3.44×10^{-9} , i.e., the manipulator is close to a gain-singular pose. Hence the simulation terminates at $t = 0.48$ s, which corresponds to the pose shown in Fig. 6.1

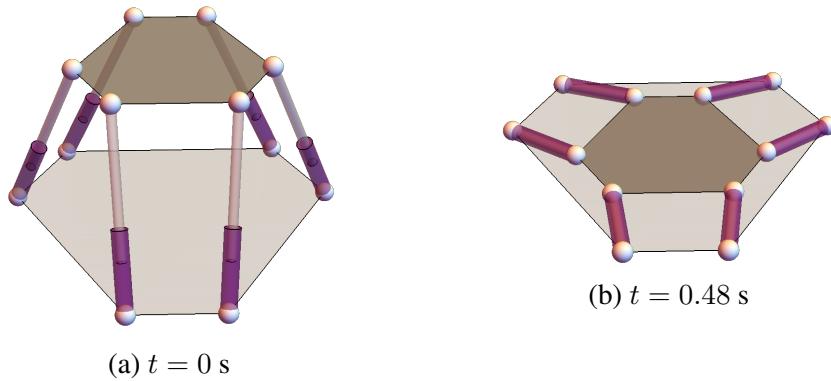


Figure 6.1: Motion of the manipulator when simulated in the actuator-space

When the manipulator reaches the singular pose in the configuration-space model, further motion is solely dependent on the solver, and one cannot precisely predict the direction of movement. As shown in Fig. 6.2, the moving platform starts moving upward after reaching the singular pose which occurs at time $t = 0.48$ s.

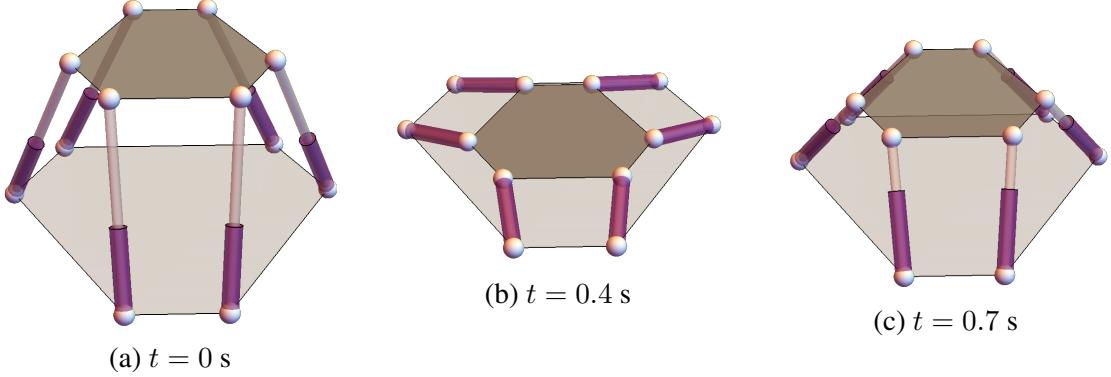


Figure 6.2: Motion of the manipulator when simulated in the configuration-space

In the extended-configuration-space mapped to task-space case, the system does not encounter any singularity at time $t = 0.48 \text{ s}$ and proceeds to fall, moving below the fixed platform as expected according to [41], which is shown in the Fig. 6.3.

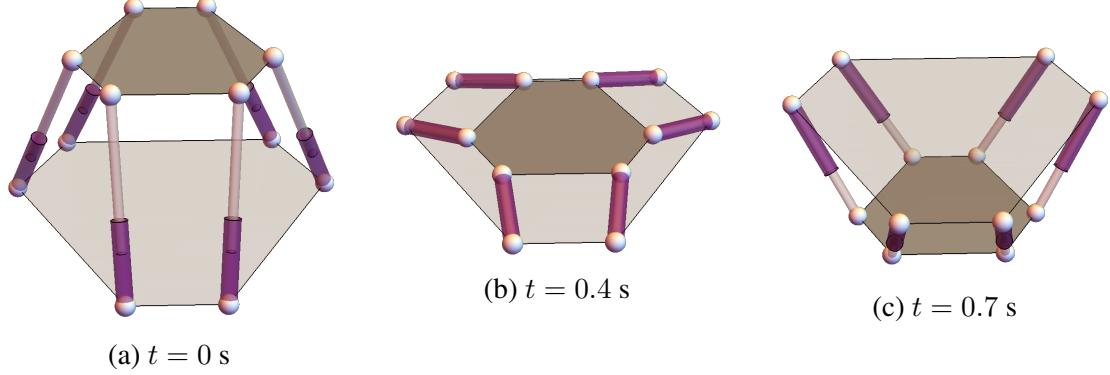


Figure 6.3: Motion of the manipulator when simulated in the extended-configuration-space

6.1.2 Implementation of dynamic simulations in C++

All the models are ported to C++ using `FileTemplateApply` and `CForm` functions for comparing their run times. Further details about the same can be found in Appendix C. These coefficient matrices in C++ are then used for simulation and comparison between the methods. Linear algebra package Eigen [?], is used for matrix manipulations. Numerical solvers of `GSL`, GNU Scientific Library [21], are used for numerical integration and solving linear equations.

After exploring a few compilers, it is found that the `clang++` [30] compiler has an advantage in compilation time, thereby cutting the testing and debugging time, with almost the same execution time as the `g++` compiler [49]. Further information can be

found in Appendix B. Execution time is further improved by using code optimisation flags during compilation. Comparison of various compiler flags and their corresponding performance can be found in the Appendix B. As expected, the maximum reduction in execution times is obtained with `-Ofast` optimisation flag. The equations of motion rewritten in the state-space form are integrated using `gsl_odeiv2_step_rkf45`, which is an explicit Runge-Kutta-Fehlberg method, a good general purpose integrator. Further information is given in Appendix C. The simulation is carried from $t = 0$ s to $t = 1$ s, with 100 equally spaced steps. For the actuator-space simulation, a Newton-Raphson method based root-tracking algorithm is written in C++. The GNU solver, `gsl_linalg_complex_LU_solve` is used for solving linear equations. The details of the code and implementation can be found in the GitLab repository https://gitlab.iitm.ac.in/ed14b037/Dynamics_CPP.git. The average simulation time for 1000 runs is recorded for each case and is as shown in Table 6.2.

Table 6.2: Computational time taken for the free-fall simulation in C++

Formulation	Real time (s)	Simulation time (s)	Compilation time (min)
Configuration space	1.000	0.223	3.259
Actuator space	0.480	0.193	3.229
Extended mapped to task-space	1.000	0.036	0.255

A few things to note during the implementation of dynamic simulation in C++ are:

1. Noted that compiling the complete \mathbf{C} matrix at once is not possible (with the computer used) and hence each element of the matrix is extracted as an individual .cpp file and is compiled independently after which they are all used to reconstruct the \mathbf{C} matrix.
2. The coefficient matrices are functions of repetitive sine and cosine of variables, and hence the evaluation of these expressions would consume a fair amount of computational effort if not computed once and reused.
3. Instead of calculating all the configuration-space variables in terms of the task-space variables alone, a cascaded substitution can be done to obtain the passive angles; such a substitution avoids recomputing repetitive expressions, saving execution time. In the case of SRSPM, the following is done,
4. The matrix inversion operation is computationally more expensive than solving a linear equation for computing the second-order derivatives of the generalised coordinates, $\ddot{\mathbf{q}}_e$, in the Eq. (2.4). Please note that using the `LinearSolve` function

- 1: First find the link lengths, \mathbf{l} given the task-space variable data, \mathbf{x}
 - 2: Compute the values of ψ given the link lengths, \mathbf{l} , and the task space data, \mathbf{x}
 - 3: Finally, find ϕ which is a function of \mathbf{x} and ψ alone.
-

in *Mathematica*® 11.2 has not shown a significant improvement in performance. However, it might still be useful in reducing the execution time to compute when implemented in C++.

6.2 Comparison of different methods

From Table 6.2 it is clear that the extended-configuration-space mapped to task-space takes the least amount of computation time. This reinforces the observations of the authors in [35] that dynamic models with lesser number of variables would not necessarily lead to faster simulation. In other words, even though the coefficient matrices in configuration-space are of size (18×18) as compared to extended-configuration-space, (24×24) size matrices, the later has a significantly lower computation time. This can be attributed to the fact that in the latter case though the matrices have a higher dimensions, they are sparse as mentioned in Section 5.1. The sparsity of \mathbf{M} and \mathbf{C} matrices are visualised as shown in Fig. 6.4. In the matrix sparsity plot, Fig. 6.4, cells with “■” denote the presence of symbols, “■■” denote the presence of constants and the white spaces denote zero valued elements.

Comparison of errors in simulation of the dynamics models

As discussed in Section 4.2, validating the dynamics model formulated is essential to check its mathematical consistency. Dynamical simulations are prone to deviate from the true evolution of the system due to the incorrect modelling, truncation errors, approximations in the numerical method used etc. It can be seen from Fig. 4.2 that the magnitude of the absolute maximum error from the loop-closure equations of the configuration-space model is much higher than it is in the actuator-space modelling which is shown in Fig. 4.5.

The NR method based root-tracking imposes a strict condition on the loop-closure equations explicitly which is reflected in a lower magnitude of errors of the actuator-

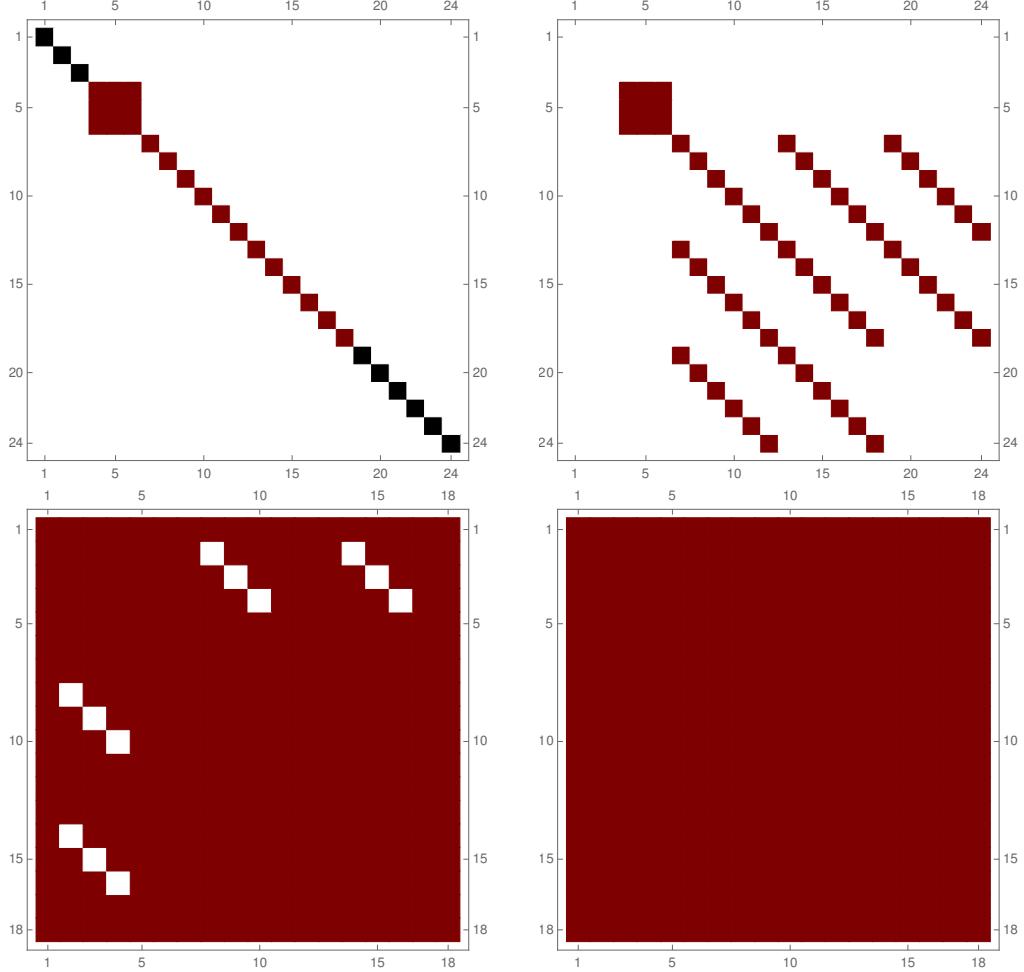


Figure 6.4: Visualising the sparsity of M and C matrices in the extended-configuration space vs. configuration-space formulations

space dynamic model as discussed in Section 4.3 as compared to the configuration-space formulation. On the other hand, the extended-configuration mapped to task-space formulation involves the computation of IK solutions at each instant which ensures the loop-closure to be satisfied and hence results in lower errors as shown in Section 5.1. The error e_1 is of the order 10^{-18} in the extended-configuration mapped to task-space case as compared to 10^{-15} in the actuator-space.

6.3 Example: Dynamics formulation and simulation of 6-RSS manipulator

Following the discussion from previous sections, extended-configuration-space is chosen to model the dynamics of the 6-RSS manipulator as an example. The architecture of

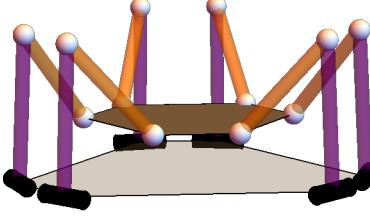


Figure 6.5: The 6-RSS manipulator

this manipulator is shown in Fig. 6.5. Since the moving and the fixed platforms of the 6-RSS manipulator are same as the SRSPM all the axes conventions and vertex numbering followed is the same. Further the same assumptions given in Section 2.2 still hold here.

The centre of the rotary joint axes are at each of its fixed platform vertices with their central axis tangential to the circle circumscribing the fixed platform. The x -axis is always aligned with the rotary joint axis and z -axis along with the length of the links by convention. Variables associated with each element of the manipulator together form the extended-configuration-space, which in this case are,

$$\mathbf{q}_e = [\mathbf{x}^\top, \boldsymbol{\phi}^\top, \boldsymbol{\theta}^\top]^\top,$$

where $\boldsymbol{\theta}$ are actuator joints, $\boldsymbol{\phi}$ are values at the passive spherical joints and \mathbf{x} denote variables representing the geometric centre and the orientation of the moving platform. This model is used to obtain an estimate of the torque requirements for a given motion of the moving platform. A mathematical model is developed following the process as provided in Chapter 5.

6.3.1 Verification of the formulated model

The numerical checks described in Section 4.2 are used here and the results are shown in Fig. 5.1.

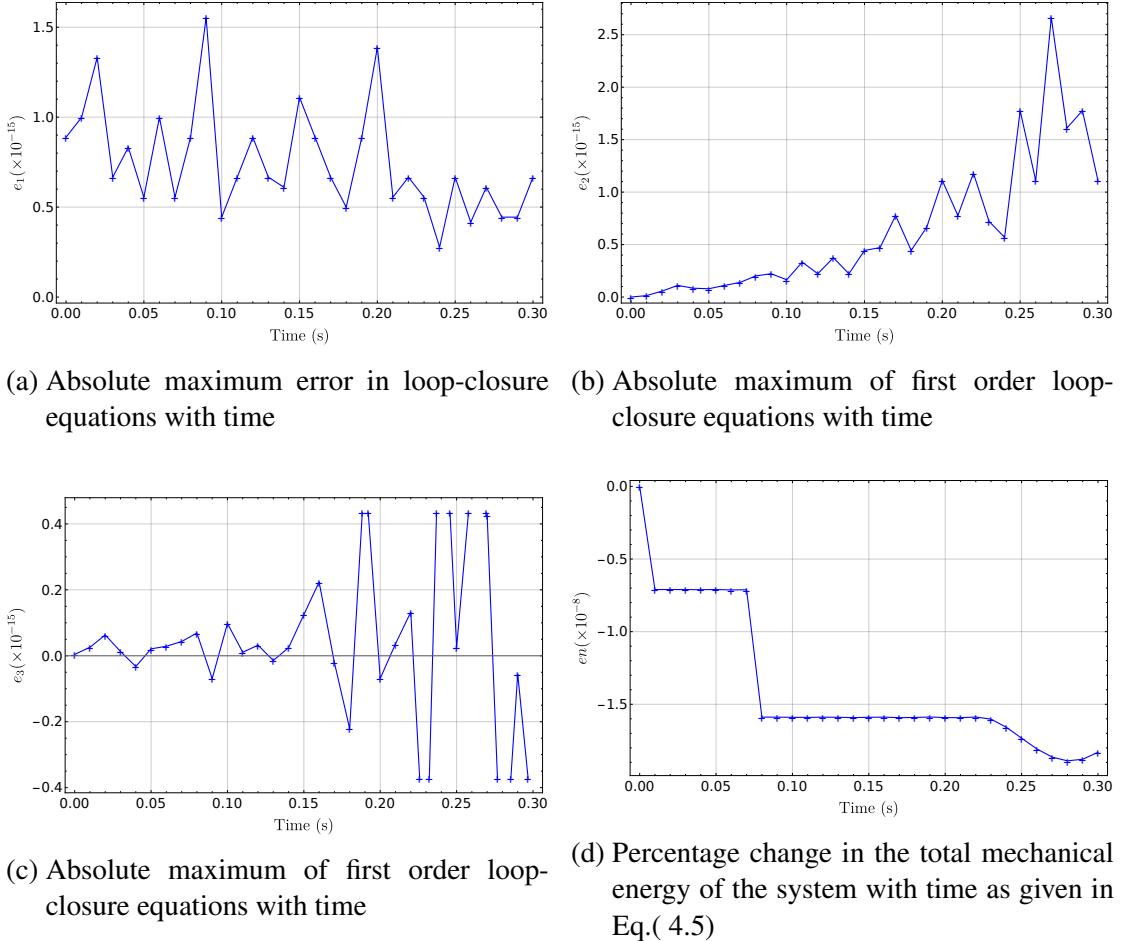


Figure 6.6: Validation of the extended-configuration-space mapped to actuator-space mathematical model

6.3.2 Inverse dynamics for heave motion requirement

The moving platform is required to produce a heave motion with constraints on its peak acceleration of $0.8g \text{ m/s}^2$ in the z direction, with the following terminal conditions,

$$\begin{aligned}\boldsymbol{x}_i &= [0, 0, 0.3, 0, 0, 0]^\top & \boldsymbol{x}'_i &= [0, 0, 0, 0, 0, 0]^\top \\ \boldsymbol{x}_f &= [0, 0, 0.4, 0, 0, 0]^\top & \boldsymbol{x}'_f &= [0, 0, 0, 0, 0, 0]^\top,\end{aligned}$$

where $\boldsymbol{x}_i, \boldsymbol{x}_f$ are the initial and final poses and $\dot{\boldsymbol{x}}_i, \dot{\boldsymbol{x}}_f$ are the initial and final velocities of the moving platform given in task-space coordinates.

The extended-configuration-space model is mapped to actuator-space and is used in the inverse dynamics (ID) simulations, to compute the torques. Newton-Raphson method based root-tracking is used for the FK computation. The model takes around 0.168 s to complete the ID simulation for a real-time of 0.3 s. The obtained torque profile is shown in Fig. 6.7. The manipulator parameters used for the simulation are given in Table ??.

The torque profile shows that for the achieved configurations during the motion the motor always supplies the positive torque. Since the initial and final poses have a zero velocity constraint, the manipulator decelerates as it reaches these positions and hence we see a peak in the torque profile. A small segment of the manipulator moving up is shown in the motion profile given in Fig. 6.8.

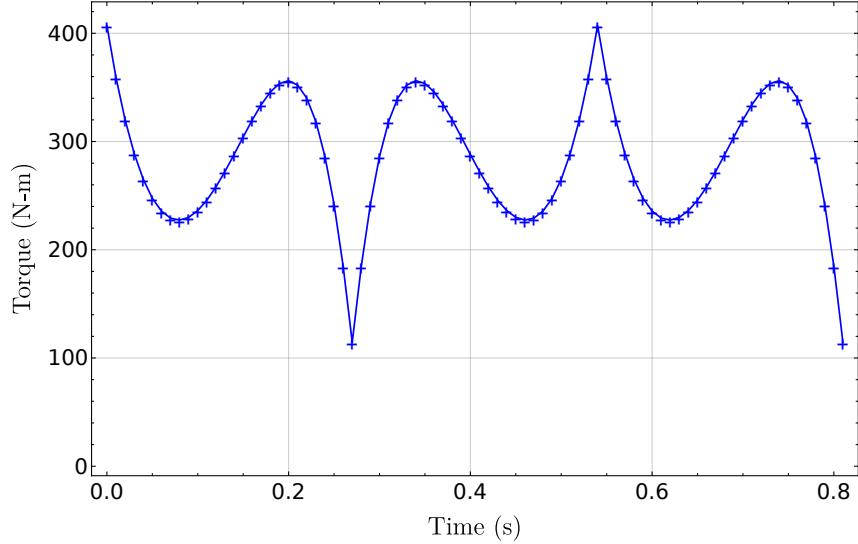


Figure 6.7: Torque profile for the heave motion requirement of the moving platform

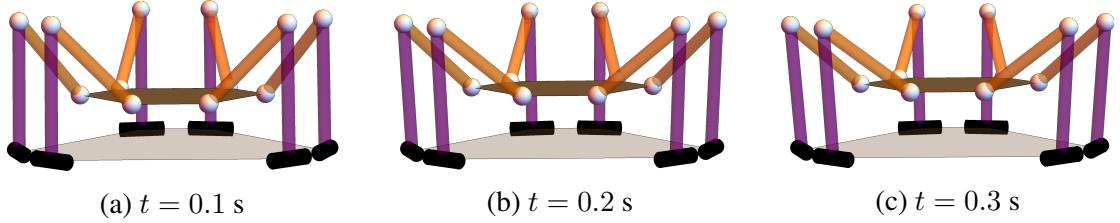


Figure 6.8: Configurations of the manipulator during heave motion

6.4 Summary

This chapter draws a comparison between different dynamics models for a given simulation. It is concluded that the extended-configuration-space formulation is the best in terms of the computational speed and the ease of coding. Modelling a system in the selected formulation is illustrated with the 6-RSS manipulator for obtaining the torque requirements for a given heave motion.

Parameter	Value
r_b	1 (m)
r_t	0.6 (m)
γ_b	0.8080 (rad)
γ_t	0.4040 (rad)
I_{tpxx}	1.1717 (kgm^2)
I_{tpyy}	1.1717 (kgm^2)
I_{tpzz}	2.3431 (kgm^2)
I_{lxx}	0.0283 (kgm^2)
I_{lyy}	0.0283 (kgm^2)
I_{lzz}	6.325×10^{-6} (kgm^2)
I_{rxx}	0.1764 (kgm^2)
I_{ryy}	0.1764 (kgm^2)
I_{rzz}	5.400×10^{-6} (kgm^2)
m_p	200.2033 (kg)
$m_{li}, i = (1, \dots, 6)$	0.5060 (kg)
$m_{ri}, i = (1, \dots, 6)$	0.4320 (kg)
$l_{li}, i = (1, \dots, 6)$	0.8200 (m)
$l_{ri}, i = (1, \dots, 6)$	0.7000 (m)

Table 6.3: Mechanical parameters of the 6-RSS manipulator used in the simulation of the dynamics model

CHAPTER 7

Trajectory-tracking control with the SRSPM

7.1 Introduction

Trajectory-tracking control deals with the problem of moving the end-effector along a given path. Given a trajectory it is easy to solve the inverse dynamics problem to obtain the required torques. However, since this computation is done using a model which is different from the actual system, the same path might not be followed if the computed torques are given to the system. One way to design a controller is to do a Jacobian linearisation about an operational point and use the linearised dynamic model to generate a control law. This works on the assumption that the M and C matrices are constant throughout the operational range, which is not valid in reality. Such controllers also do not ensure uniform performance throughout the working range as shown in [18].

Given a reliable dynamics model of the system, one can use non-linear controllers like feedback linearisation followed by a proportional-derivative (PD) controller. This technique is also called as computed torque control (CTC) or non-linear dynamics decoupling approach. The idea is to transform the model into a linear system by *cancelling* the non-linear terms. This approach is similar to coordinate transformation in mechanical systems to make their description simpler. Linear control techniques can be used once the model is feedback-linearised. Following [47], note that this method has the following limitations.

1. Not all models can be feedback linearised (if not fully actuated e.g. cart-pole system);
2. The coefficient matrices are functions of all the; extended-configuration-space variables, and hence full-state feedback is needed and is assumed that it can be measured at all times;
3. Robustness cannot be guaranteed under parameter uncertainty.

7.2 Feedback linearisation

In this section a PD augmented feedback linearisation is used to track a given trajectory. It is shown by [7] that for a fully actuated parallel manipulator, loss in the rank of the force transformation matrix leads to uncontrollable states. In other words, the manipulator becomes uncontrollable at a gain-type singularity. To avoid such configurations, it should be ensured that the followed path does not intersect the singularity manifold. A more strict condition for a parallel manipulator would be to limit its working within the SWZ, as defined in [26]. For dealing with a trajectory-tracking problem in the configuration-space, one may refer to [3].

7.2.1 The SRSPM following a given circular path

In this case, a circle defined in Eq.(7.1) of radius 0.4 m is to be tracked in 1.57 s with an initial error from the reference trajectory. The corresponding gains are $K_p = 2000$ and $K_d = 2\sqrt{2000}$, which makes the tracking slightly over-damped and satisfying a settling time constraint of $t = 0.2$ s are used.

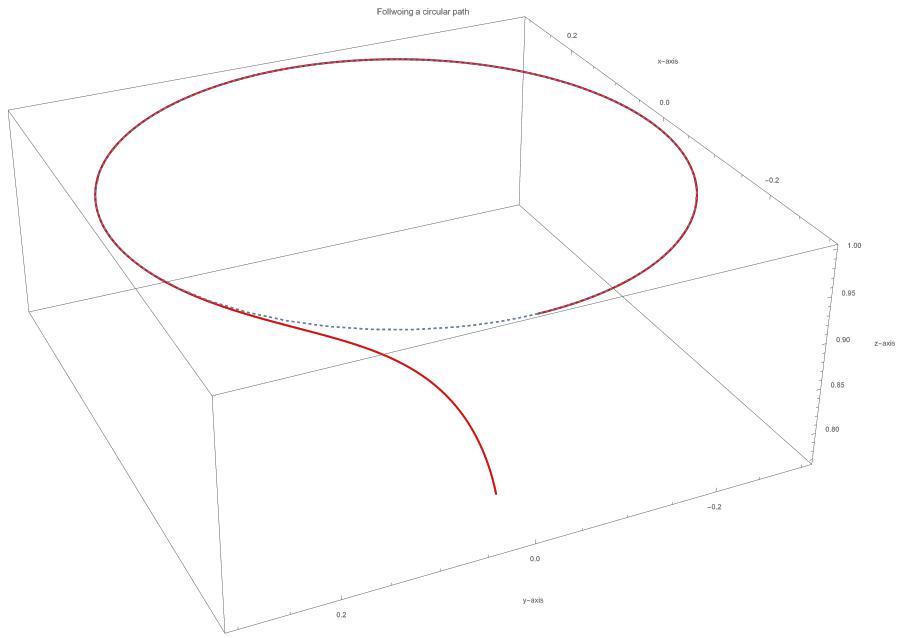
$$x^2 + y^2 - 0.4^2 = 0. \quad (7.1)$$

Figure 7.1 shows the error defined at a given time instant as:

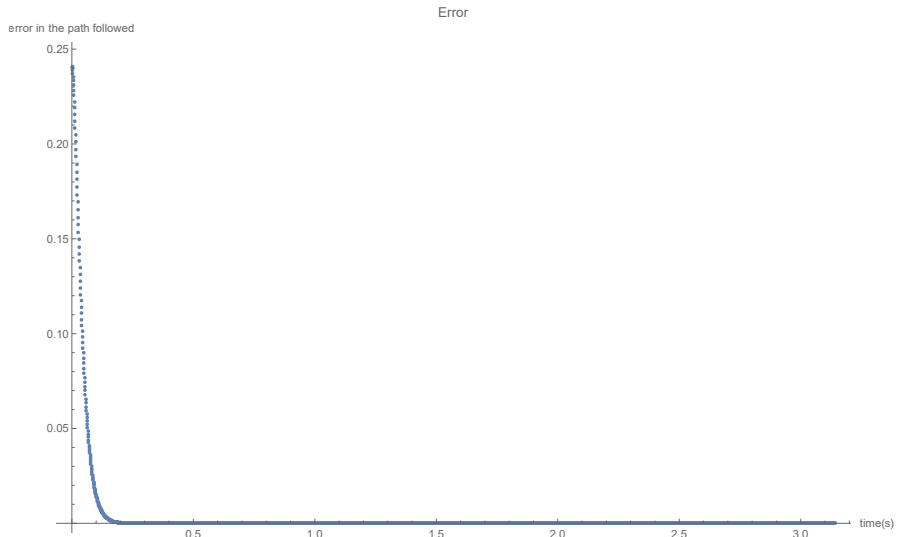
$$\text{tracking error} = \| \text{desired position} - \text{actual position} \|,$$

where the blue dotted line represents the desired path and the red solid line represents the tracked path. The tracking controller has taken only 0.552 s for the simulation. It is observed, as expected, that for slower end-effector speeds the magnitude of error is smaller. It would be interesting to study the behaviour of the controller for a high-frequency input, i.e., while tracking a square, near its vertices. The current dynamic model enables faster computation of control input thereby reducing the tracking errors. As expected from the error dynamics, since the eigenvalues are all on the left half plane, i.e., negative, the errors asymptotically go to zero as seen in Fig. 7.1. Now it is desired to test the model in two scenarios:

1. Following a path for longer time; to verify that the errors do not accumulate.



(a) Tracked vs the reference path



(b) Error in tracking

Figure 7.1: Following a circular path by SRSPM

2. Following a path with high frequency changes, i.e., like a vertex of a square.

7.2.2 Following a 3D-Lissajous curve with SRSPM

To check the first condition a 3D-Lissajous curve is taken to follow from a given offset from the desired position at time $t = 0$. An advantage of the 3D-Lissajous curve is that

it is time parametrised curve:

$$\begin{aligned}x(t) &= x_p \sin(\omega_x t - \delta_x) \\y(t) &= y_p \sin(\omega_y t - \delta_y) \\z(t) &= 1 + z_p \sin(\omega_z t - \delta_z)\end{aligned}$$

where x_p, y_p, z_p represent magnitude, $\omega_x, \omega_y, \omega_z$ are the frequencies, $\delta_x, \delta_y, \delta_z$ is the phase difference of parametrised form of the curve, and the details of which are given in Appendix A. One could also vary the individual frequencies in each of the Cartesian coordinate can be used to gauge the error response of along that coordinate. The chosen path is as shown in Fig. 7.2.

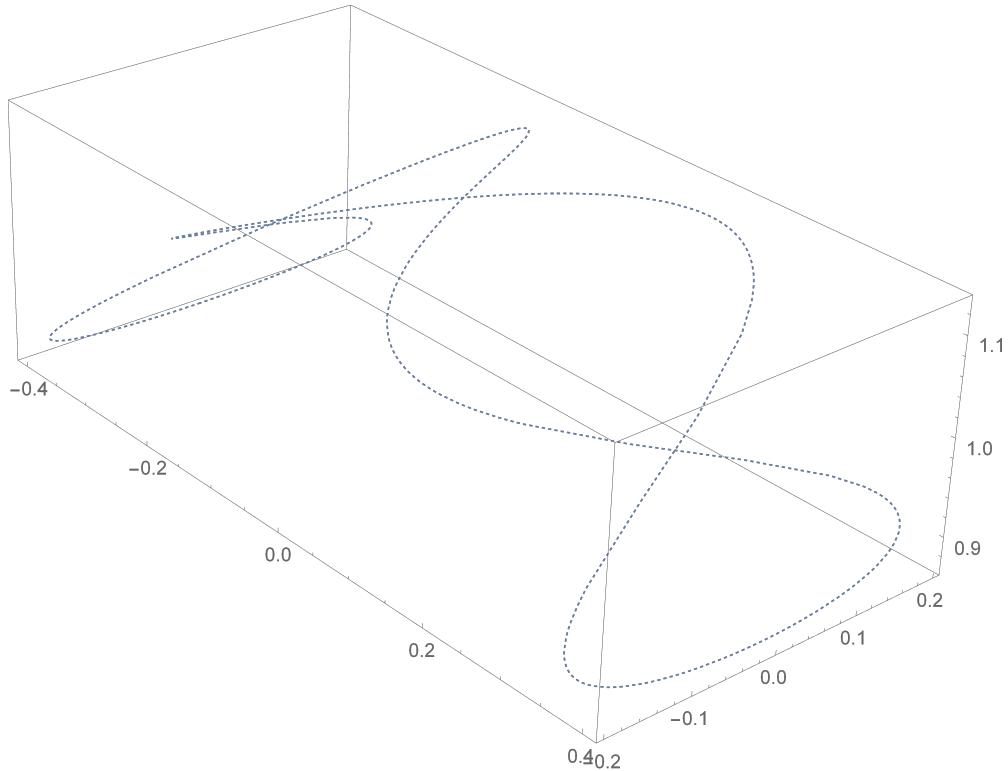


Figure 7.2: The 3D Lissajous curve to be tracked by SRSPM

The moving platform is assumed to be parallel to the base for the full motion, i.e., $\{c_1, c_2, c_3\}$ are zero for all time. Extended configuration-space mapped to the actuator-space model is used for control of the manipulator.

The derivation of the complete control law can be found in [18]. The choice of $K_p = 2000$ and $K_d = 2\sqrt{2000}$ ensure slight over damped nature and to achieve a settling time of lesser than 0.2 s. Starting with an initial error, the path is required to be tracked

in 3.14 s which is simulated in 1.24 s. In case of a physical manipulator, it must be ensured that the selected gains do not cause saturation in the actuator causing the control input to be clipped to the maximum. The error in the tracked path after the settling

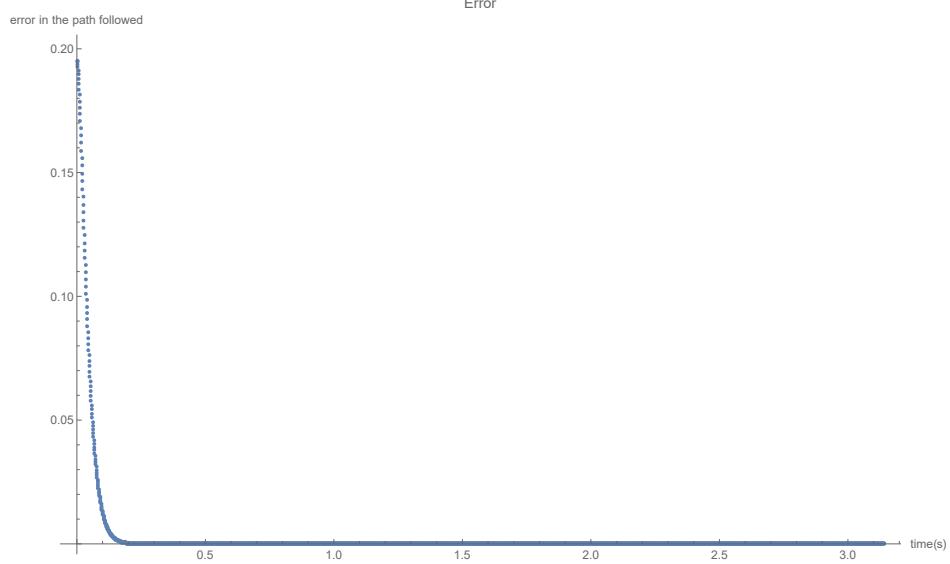


Figure 7.3: Norm of the tracking error

time constraint is shown in Fig. 7.4 The tracked vs the desired path are as shown in

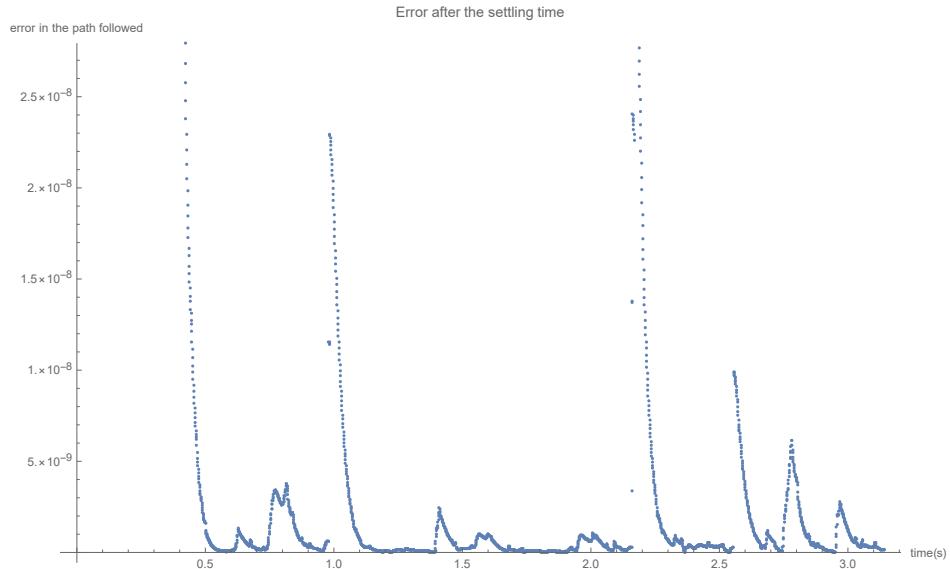


Figure 7.4: Norm of the tracking error

Fig. 7.5, where the red line corresponds to the tracked path from an initial error and the blue dotted line is the desired motion. It is observed that if more time is given for the manipulator to follow the path the errors further decrease, but this decrease is smaller and has a significant influence by the controller gains.

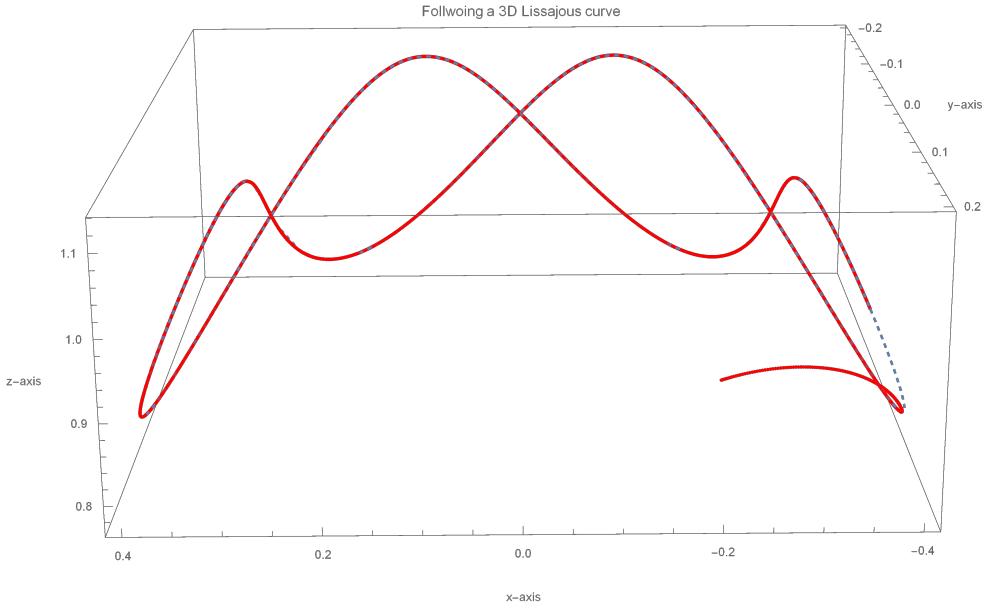


Figure 7.5: Followed vs desired path of the 3D-Lissajous curve by SRSPM

7.3 Following a rectangular path with SRSPM

To check the second hypothesis it is necessary to quantify the manipulators behaviour at the vertices of a rectangle. The manipulator is required to follow a rectangle of dimensions, $0.5 \text{ (m)} \times 1 \text{ (m)}$, with a time constraint of 4 s. For the above scenario, the following is the tracked vs desired path shown in Fig. 7.6 Observe the deviation of

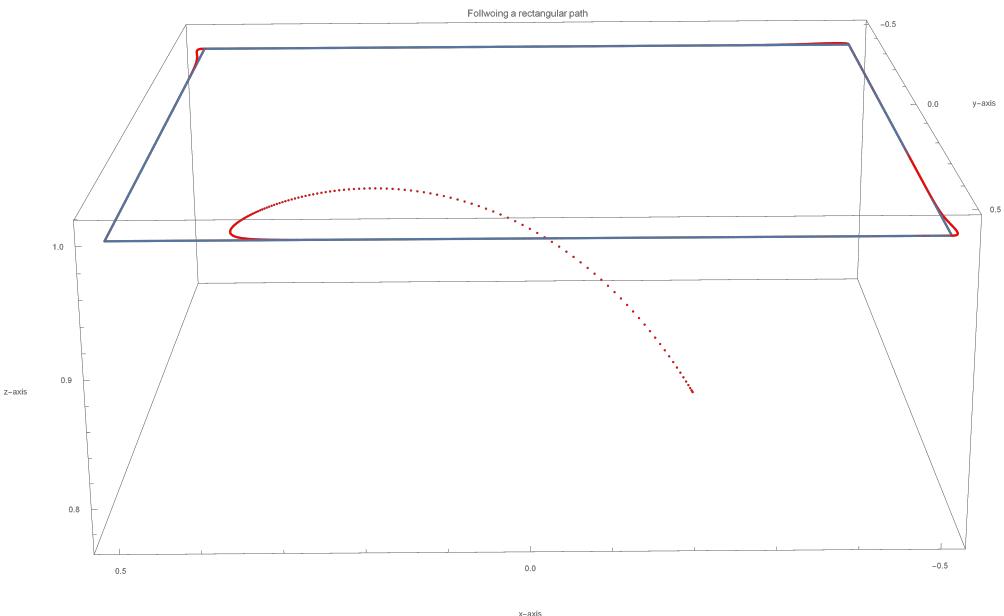


Figure 7.6: Followed vs desired rectangular path by SRSPM

the manipulator from its desired path at the vertices. The error in following this path is given in Fig. 7.7. When the time constraint to followed is halved, i.e., if the manipulator

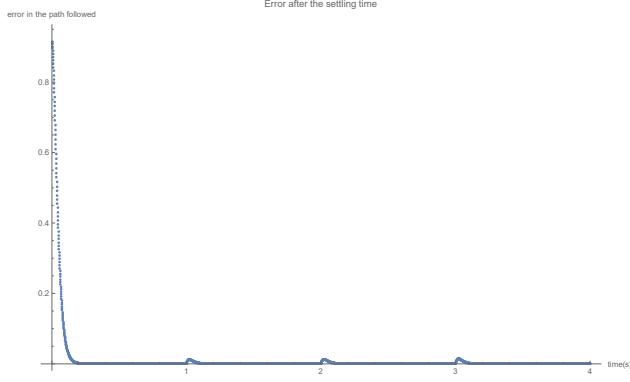


Figure 7.7: Error in the followed vs desired rectangular path by SRSPM

is required to follow the square in 2 s, the response is shown in Fig. 7.8. Note that it

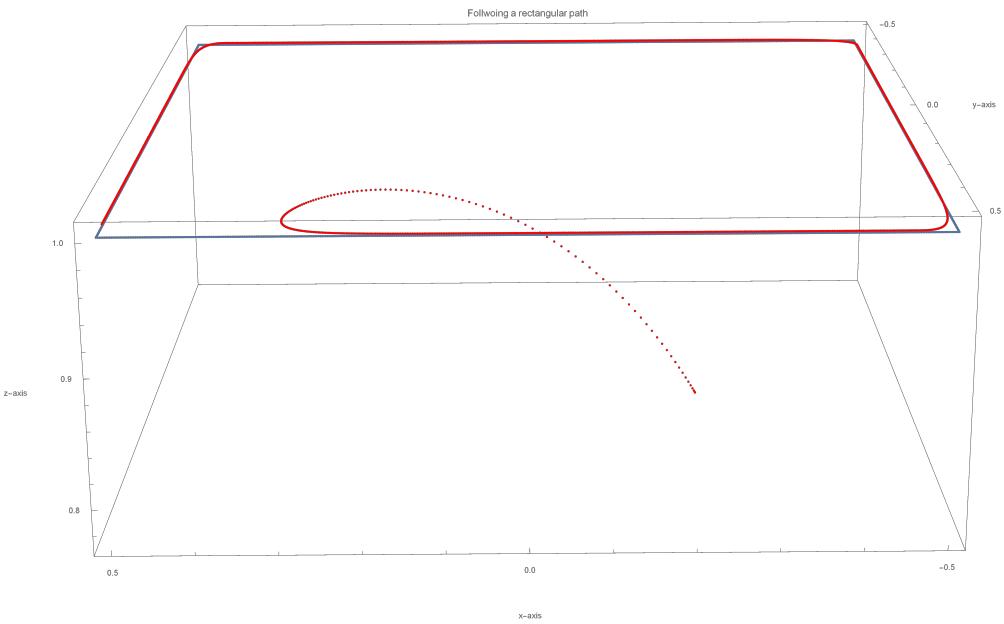


Figure 7.8: Followed vs desired rectangular path by SRSPM

this scenario, the manipulator is desired to move fast but due to the finite sampling time and computation of control input, the error increases from the previous case. However, the errors can be further reduced by setting the desired time and the gains used. Since there is no literature to compare the tracking of a square, no comments are given about the magnitude of the error.

7.4 Summary

A planar circle and 3D-Lissajous curves are followed with the extended-configuration-space dynamic model, which enables faster computation of control input. The deviation

of the end-effector from the desired or reference path is shown. The simulation times and errors are significantly smaller than the ones reported in [3]. Further a square path is also followed to demonstrate the computational capability of given model to follow the vertex points effectively. It is to be noted that the time taken for simulation in [3] is only for a small arc of motion and hence not compared with paths tracked in this report. But for the time of simulation reported i.e., 2 s the execution time taken was 20 m, whereas in the current report the order of magnitude of time taken for execution is seconds (less than the time requirement given for following in most cases).

CHAPTER 8

Conclusion

8.1 Overview

The report begins with a brief survey on modelling, simulation and control of SRSPM is presented in Chapter 1, from where it is understood that the model is often idealised to meet real-time execution goals and the errors are compensated by complex control algorithms which bring down the working bandwidth of the manipulator.

Chapter 2 discusses ways to track the FK solutions faster using root-tracking methods, followed by two ways in which the velocity Jacobian matrices can be derived. Newton-Raphson method based root-tracking is used for all the simulations as it has direct control of the precision of solutions obtained.

The standard formulations in dynamic modelling are illustrated. These models are then studied and verified for mathematical consistency. Implementation issues, model breakdowns and common inaccuracies in the validation process are discussed.

Taking useful elements from each of the models an extended-configuration-space formulation is developed. Since this is a higher dimensional space, it is demonstrated that it can be mapped to both the task-space and the actuator-space in Chapter 4. This results in an unconstrained actuator-space dynamic model which can be used for controller design.

All the dynamic models are simulated for a free-fall under the effect of gravity scenario, and the observations are discussed in Chapter 5. Comparisons of the corresponding implementations and run-times in C++ are given. Further, a few practical considerations are also discussed to improve the run-time. The relative sparsity of the coefficient matrices is illustrated, and an example of a dynamic simulation of the 6-RSS manipulator is also included.

Chapter 6 deals with the trajectory-tracking problem. Feedback linearisation followed by a PD control scheme is used for following a circle and a 3D-Lissajous curve.

It is seen that the simulation time and the deviation from the desired path are significantly smaller than the values reported in [3].

8.2 Possible extensions

For a robot, generally, the goal description is given in its task-space coordinates. Hence, as shown in [28], an operational-space approach would allow the system to take high-level commands in the task-space and execute them in the actuator-space. This is done by mapping the task-space wrench into the actuator-space forces by a Jacobian matrix:

$$\mathbf{T} = \mathbf{J}_{\eta q}^\top \mathbf{F},$$

where \mathbf{F} is the task-space wrench, $\mathbf{J}_{\eta q}$ is the Jacobian matrix and \mathbf{T} are the set of actuator-space forces. As seen in Section 5.2, task-space mapped extended-configuration-space dynamics is faster to compute and deviates lesser than the model mapped to actuator-space. Hence combining the idea of operational-space control of the manipulator with extended-configuration-space dynamic model would further reduce the execution time significantly.

The unconstrained dynamics of the actuator and task-space models are obtained from the extended space by performing mapping given by,

$$\begin{aligned}\boldsymbol{\tau}_x &= \mathbf{J}_{q_e x}^\top \boldsymbol{\tau}_e, \\ \boldsymbol{\tau}_\theta &= \mathbf{J}_{q_e \theta}^\top \boldsymbol{\tau}_e.\end{aligned}$$

Using the above information another way to relate the task and actuator-space forces is by defining a dynamic wrench-transformation matrix \mathbf{H} , such that:

$$\begin{aligned}\mathbf{J}_{q_e x} \mathbf{H}^\top &= \mathbf{J}_{q_e \theta} \\ \mathbf{H} &= (\mathbf{J}_{q_e x}^\# \mathbf{J}_{q_e \theta})^\top,\end{aligned}$$

where $(\cdot)^\#$ returns the pseudoinverse of a matrix. Since a pseudoinverse is used, the elements of \mathbf{H} matrix that might have been lost in the nullspace during the transformation should be extracted by posing additional constraints such as row-reducing the

Jacobian matrices on both sides and inverting the obtained 6×6 matrix from the 24×6 matrix on one side to obtain the \mathbf{H} matrix.

Moreover, the properties of the $\mathbf{J}_{\eta q_x}$ matrix should be studied to understand the possible degeneracies of the mapping between the extended-space and task-space.

Further a model-predictive controller needs to be implemented using the derived extended-configuration-space dynamics to illustrate the potential of the model in achieving high controller bandwidth.

APPENDIX A

Manipulator parameters

Parameter	Value
r_b	1 (m)
r_t	0.5803 (m)
γ_b	0.2985 (rad)
γ_t	0.6573 (rad)
I_{tpxx}	1.17172 (kg m^2)
I_{tpyy}	1.17172 (kg m^2)
I_{tpzz}	2.34310 (kg m^2)
I_{laxx}	2.1272 (kg m^2)
I_{layy}	2.1272 (kg m^2)
I_{lazz}	0.00173 (kg m^2)
I_{lbxx}	0.02431 (kg m^2)
I_{lbyy}	0.02431 (kg m^2)
I_{lbzz}	0.00040 (kg m^2)
m_p	0.20339 (kg)
$m_{bi}, i = (1, \dots, 6)$	11.3404 (kg)
$m_{ai}, i = (1, \dots, 6)$	1.15719 (kg)
$l_{bi}, i = (1, \dots, 6)$	0.5 (m)
$l_{ai}, i = (1, \dots, 6)$	1.5 (m)

Table A.1: Mechanical parameters of the SRSPM used in the simulation of the dynamics model

Details of the Lissajous curve are as given in the Table A.2.

Parameter	Value
x_p	0.400 (m)
y_p	0.200(m)
z_p	0.133 (m)
ω_x	2 (rad ¹ /s)
ω_y	8 (rad/s)
ω_z	4 (rad/s)
δ_x	$\pi/2$ (rad)
δ_y	0 (rad)
δ_z	$\pi/2$ (rad)

Table A.2: Details of the tracked 3D-Lissajous curve

APPENDIX B

Effect of compilers and flags on the execution time of dynamics models

A few compilers are tested to understand each of their computation and compilation time advantages. The comparison of their performance in simulating the extended-configuration-space dynamic model is presented in Table B.1, keeping the optimisation flags fixed. The documentation of each of the compilers is hyperlinked in the same.

Table B.1: Comparison of compilers with respect to compilation and execution times for a free-fall simulation time of 0.5 s

Compiler	Flag	Compilation time (s)	Execution time (s)
gcc	-Ofast	28	0.045
clang++	-Ofast	17	0.042
pgc++	-O4	1275	0.611
g++	-Ofast	28	0.042

For a given compiler the effects of various optimisation flags is listed in Table B.2. Configuration-space dynamic model is compiled with different flags, and the time taken for compilation and execution are noted in Table B.2.

Table B.2: Comparison of flags with respect to compilation and execution times of the C matrix

Compiler	Flag	Compilation time (min)	Execution time (s)
clang++	-O1	4.490	1.750
clang++	-O2	5.042	1.450
clang++	-O3	5.459	1.435
clang++	-Ofast	4.425	0.942

`Perf` profiler is useful tool in investigating the time taken by each function. This gives an idea on which kind of operation takes the most amount of time, i.e., a computation or a substitution and can be optimise accordingly.

APPENDIX C

Details of C++ implementation of the dynamics models

Please note that while benchmarking the time for running any piece of code, other programs running on the PC at the same time have a significant impact on the code execution timing and might take up to 2-3 times more time. Applications running in the background can be monitored using `htop` in Linux. An alternative might be to use the `google_benchmark` library, which is a micro-benchmark support library, i.e., it can be used to benchmark code snippets.

All the support and simulation files are uploaded to the repository `Dynamics_CPP` in [GitLab](https://gitlab.iitm.ac.in/ed14b037/Dynamics_CPP.git) (https://gitlab.iitm.ac.in/ed14b037/Dynamics_CPP.git). Every formulation has a `support` and `sim` branch directories. The later is ready to use folder which has a `Makefile` to build the dynamics model and output an executable binary file. All the initial, final conditions, number of steps, solver etc. should be mentioned in the `dynamics_main.cpp` file. The `support` branch can be used for testing and debugging individual expressions. It can also be used for changes in formulation, parameters, expression optimisation etc., by editing the main `.nb` file used for formulating the expressions. Template `.cpp` files present in this branch are used to automatically convert the expressions into C++ executable versions. Therefore, for generating an executable dynamics model, change the `.nb` file in the `support` branch and execute it to generate the model coefficients in C++ and then use the `make` command to build the model in the `sim` branch.

Runge-Kutta-Fehlberg (4, 5) method is used for simulating all the dynamics models, and its documentation can be found here. It is observed that when Adams solver is used for simulations in *Mathematica*® 11.2, there is a chance that it keeps refining its steps smaller and smaller eventually taking a lot of time during simulation. Whereas explicit Runge-Kutta method does not have such a problems, but the accuracy of the solutions is compromised as shown in Fig. C.1. This method is also quite prevalent in the literature as a general purpose integrator. This method when used for simulation, does not require the computation of Jacobian matrices as opposed to Adams solver.

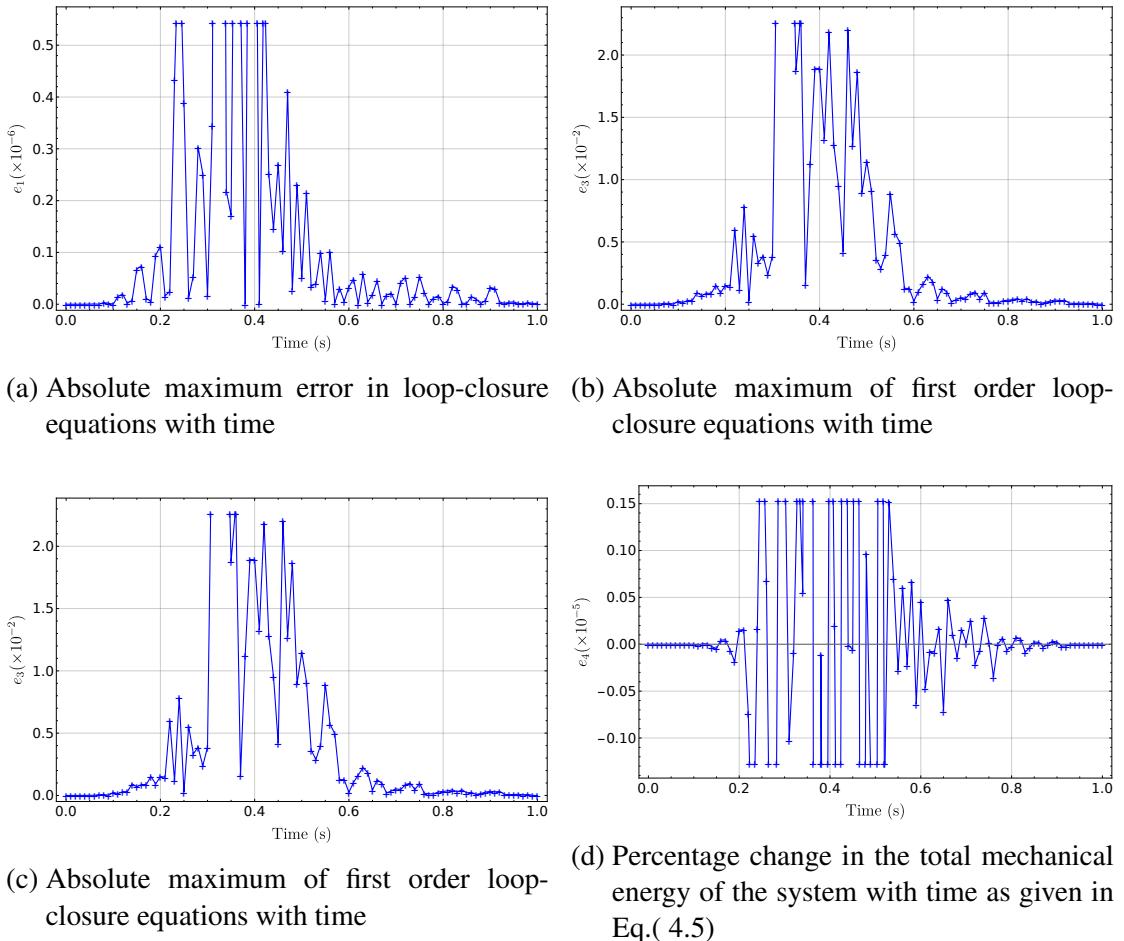


Figure C.1: Errors using the explicit Runge-Kutta solver for configuration-space dynamic model simulation

REFERENCES

- [1] H. Abdellatif, M. Grotjahn, and B. Heimann, “High efficient dynamics calculation approach for computed-force control of robots with parallel structures,” in *44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 2024–2029.
- [2] J. Angeles and J. Angeles, *Fundamentals of Robotic Mechanical Systems*. Springer, 2002, vol. 2.
- [3] S. M. Anwar and S. Bandyopadhyay, “Trajectory tracking control of semi-regular Stewart platform manipulator,” Master’s thesis, Indian Institute of Technology Madras, 2009.
- [4] ——, “Trajectory-tracking control of semi-regular Stewart platform manipulator,” in *Proceedings of the 15th National Conference on Machines and Mechanisms,(NaCoMM 2011)*, 2012, pp. 446–454.
- [5] V. Ashtekar and S. Bandyopadhyay, “Forward dynamics of the double-wishbone suspension mechanism using the embedded Lagrangian formulation,” *Asian MMS 2018 Conference*, vol. 064, 2018.
- [6] S. Bandyopadhyay and A. Ghosal, “Geometric characterization and parametric representation of the singularity manifold of a 6–6 Stewart platform manipulator,” *Mechanism and Machine Theory*, vol. 41, no. 11, pp. 1377–1400, 2006.
- [7] P. Choudhury and A. Ghosal, “Singularity and controllability analysis of parallel manipulators and closed-loop mechanisms,” *Mechanism and Machine Theory*, vol. 35, no. 10, pp. 1455–1479, 2000.
- [8] V. Damic and M. Cohodar, “Dynamic analysis of Stewart platform by bond graphs,” *Procedia Engineering*, vol. 100, pp. 226–233, 2015.
- [9] B. Dasgupta and P. Choudhury, “A general strategy based on the Newton-Euler approach for the dynamic formulation of parallel manipulators,” *Mechanism and machine theory*, vol. 34, no. 6, pp. 801–824, 1999.
- [10] B. Dasgupta and T. Mruthyunjaya, “A canonical formulation of the direct position kinematics problem for a general 6-6 stewart platform,” *Mechanism and Machine Theory*, vol. 29, no. 6, pp. 819–827, 1994.
- [11] ——, “Closed-form dynamic equations of the general Stewart platform through the Newton–Euler approach,” *Mechanism and machine theory*, vol. 33, no. 7, pp. 993–1012, 1998.
- [12] ——, “A Newton-Euler formulation for the inverse dynamics of the Stewart platform manipulator,” *Mechanism and machine theory*, vol. 33, no. 8, pp. 1135–1152, 1998.

- [13] I. Davliakos and E. Papadopoulos, “Model-based control of a 6-dof electrohydraulic Stewart-Gough platform,” *Mechanism and machine theory*, vol. 43, no. 11, pp. 1385–1400, 2008.
- [14] W. Do and D. Yang, “Inverse dynamic analysis and simulation of a platform type of robot,” *Journal of Robotic Systems*, vol. 5, no. 3, pp. 209–227, 1988.
- [15] E. F. Fichter, “A Stewart platform-based manipulator: general theory and practical construction,” *The International Journal of Robotics Research*, vol. 5, no. 2, pp. 157–182, 1986.
- [16] J. Freeman, G. Watson, Y. Papelis, T. Lin, A. Tayyab, R. Romano, and J. Kuhl, “The iowa driving simulator: An implementation and application overview,” SAE Technical Paper, Tech. Rep., 1995.
- [17] Z. Geng, L. S. Haynes, J. D. Lee, and R. L. Carroll, “On the dynamic model and kinematic analysis of a class of Stewart platforms,” *Robotics and autonomous systems*, vol. 9, no. 4, pp. 237–254, 1992.
- [18] A. Ghosal, *Robotics: Fundamental Concepts and Analysis*. New Delhi: Oxford University Press, 2006.
- [19] H. Goldstein, C. P. Poole, and J. L. Safko, *Classical Mechanics*. Addison Wesley, 2001.
- [20] C. Gosselin, “Parallel computational algorithms for the kinematics and dynamics of planar and spatial parallel manipulators,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 118, no. 1, pp. 22–28, 1996.
- [21] B. Gough, *GNU Scientific Library Reference Manual - Third Edition*, 3rd ed. Network Theory Ltd., 2009.
- [22] K. Hashimoto, K. Ohashi, and H. Kimura, “An implementation of a parallel algorithm for real-time model-based control on a network of microprocessors,” *The International Journal of Robotics Research*, vol. 9, no. 6, pp. 37–47, 1990.
- [23] M. Honegger, R. Brega, and G. Schweiter, “Application of a nonlinear adaptive controller to a 6 dof parallel manipulator,” in *IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 1930–1935.
- [24] C.-I. Huang, C.-F. Chang, M.-Y. Yu, and L.-C. Fu, “Sliding-mode tracking control of the Stewart platform,” in *5th Asian Control Conference*, vol. 1. IEEE, 2004, pp. 562–569.
- [25] Z. Ji, “Study of the effect of leg inertia in Stewart platforms,” in *IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 121–126.
- [26] M. K. Karnam, A. Basker, R. A. Srivatsan, and S. Bandyopadhyay, “Computation of the safe working zones of parallel manipulators,” accepted in Robotica 2018.
- [27] W. Khalil and S. Guegan, “Inverse and direct dynamic modeling of Gough-Stewart robots,” *IEEE Transactions on Robotics*, vol. 20, no. 4, pp. 754–761, 2004.
- [28] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation.”

- [29] N.-I. Kim and C.-W. Lee, “High speed tracking control of Stewart platform manipulator via enhanced sliding mode control,” in *IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, 1998, pp. 2716–2721.
- [30] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis and transformation,” San Jose, CA, USA, Mar 2004, pp. 75–88.
- [31] G. Lebret, K. Liu, and F. L. Lewis, “Dynamic analysis and control of a Stewart platform manipulator,” *Journal of Robotic Systems*, vol. 10, no. 5, pp. 629–655, 1993.
- [32] S.-H. Lee, J.-B. Song, W.-C. Choi, and D. Hong, “Position control of a Stewart platform using inverse dynamics control with approximate dynamics,” *Mechatronics*, vol. 13, no. 6, pp. 605–619, 2003.
- [33] T.-Y. Lee and J.-K. Shim, “Forward kinematics of the general 6–6 stewart platform using algebraic elimination,” *Mechanism and Machine Theory*, vol. 36, no. 9, pp. 1073–1085, 2001.
- [34] ——, “Improved dialytic elimination algorithm for the forward kinematics of the general Stewart–Gough platform,” *Mechanism and Machine Theory*, vol. 38, no. 6, pp. 563–577, 2003.
- [35] M. Léger and J. McPhee, “Selection of modeling coordinates for forward dynamic multibody simulations,” *Multibody System Dynamics*, vol. 18, no. 2, pp. 277–297, 2007.
- [36] A. D. Lewis, “Is it worth learning differential geometric methods for modeling and control of mechanical systems?” *Robotica*, vol. 25, no. 6, pp. 765–777, 2007.
- [37] D. Li and S. Salcudean, “Modeling, simulation, and control of a hydraulic Stewart platform,” in *IEEE International Conference on Robotics and Automation*, vol. 4. IEEE, 1997, pp. 3360–3366.
- [38] K. Liu, F. Lewis, G. Lebret, and D. Taylor, “The singularities and dynamics of a Stewart platform manipulator,” *Journal of Intelligent and Robotic Systems*, vol. 8, no. 3, pp. 287–308, 1993.
- [39] M.-J. Liu, C.-X. Li, and C.-N. Li, “Dynamics analysis of the Gough-Stewart platform manipulator,” *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 94–98, 2000.
- [40] O. Ma and J. Angeles, “Architecture singularities of platform manipulators,” in *IEEE International Conference on Robotics and Automation*. IEEE, 1991, pp. 1542–1547.
- [41] V. Muralidharan, T. K. Mamidi, S. Guptasarma, A. Nag, and S. Bandyopadhyay, “A comparative study of the configuration-space and actuator-space formulations of the lagrangian dynamics of parallel manipulators and the effects of kinematic singularities on these,” *Mechanism and Machine Theory*, vol. 130, pp. 403 – 434, 2018.

- [42] A. Nag and S. Bandyopadhyay, “A comparative study of the Configuration-Space and Actuator-Space forward dynamics of closed-loop mechanisms using the Lagrangian formalism,” in *Machines, Mechanism and Robotics*. Springer, 2019, pp. 95–106.
- [43] C. Nasa and S. Bandyopadhyay, “Trajectory-tracking control of a planar 3-RRR parallel manipulator with singularity avoidance,” in *13th World Congress in Mechanism and Machine Science, Guanajuato, Mexico*, 2011, pp. 19–25.
- [44] C. C. Nguyen, S. S. Antrazi, Z.-L. Zhou, and C. E. Campbell Jr, “Adaptive control of a Stewart platform-based manipulator,” *Journal of Robotic systems*, vol. 10, no. 5, pp. 657–687, 1993.
- [45] M. K. Park, M. C. Lee, K. S. Yoo, K. Son, W. S. Yoo, and M. C. Han, “Development of the pnu vehicle driving simulator and its performance evaluation,” in *IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, 2001, pp. 2325–2330.
- [46] J. Pradipta, M. Klünder, M. Weickgenannt, and O. Sawodny, “Development of a pneumatically driven flight simulator Stewart platform using motion and force control,” in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. IEEE, 2013, pp. 158–163.
- [47] J.-J. E. Slotine, W. Li *et al.*, *Applied nonlinear control*. Prentice Hall Englewood Cliffs, NJ, 1991, vol. 199, no. 1.
- [48] D. Sosa-Méndez, E. Lugo-González, M. Arias-Montiel, and R. A. García-García, “ADAMS-MATLAB co-simulation for kinematics, dynamics, and control of the stewart–gough platform,” *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, p. 1729881417719824, 2017.
- [49] R. M. Stallman and G. DeveloperCommunity, “Using the gnu compiler collection: A gnu manual for gcc version 4.3. 3,” *CreateSpace, Paramount, CA*, 2009.
- [50] D. Stewart, “A platform with six degrees of freedom,” *Proceedings of the Institution of Mechanical Engineers*, vol. 180, no. 1, pp. 371–386, 1965.
- [51] Y. X. Su, B. Y. Duan, C. H. Zheng, Y. Zhang, G. Chen, and J. Mi, “Disturbance-rejection high-precision motion control of a Stewart platform,” *IEEE Transactions on Control Systems Technology*, vol. 12, no. 3, pp. 364–374, 2004.
- [52] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
- [53] L.-W. Tsai, “Solving the inverse dynamics of a Stewart-Gough manipulator by the principle of virtual work,” *Journal of Mechanical Design*, vol. 122, no. 1, pp. 3–9, 2000.
- [54] F. E. Udwadia and R. E. Kalaba, *Analytical dynamics: a new approach*. Cambridge University Press, 2007.
- [55] J. Wang and C. M. Gosselin, “A new approach for the dynamic analysis of parallel manipulators,” *Multibody System Dynamics*, vol. 2, no. 3, pp. 317–334, 1998.