I Contents

II Abstract

Industrial robots (especially 6-axis) are used for machining tasks in various applications nowadays. In contrast to conventional machine tools, they are less rigid, negatively impacting the workpiece quality. To improve their accuracy and precision, it is necessary to model their dynamic parameters and compensate for torque calibrations. This involves characterizing the robot's dynamic behavior, including all dynamic parameters, and using that information to adjust the control algorithms and improve the robot's performance. The goal is to achieve better control over the robot's movements, reducing errors and increasing the accuracy and repeatability of the machining process. For this purpose, identifying dynamic model parameters of the industrial robot, referring to the inertial parameters of the robot links, Coulomb-viscous friction parameters, and actuator(motor) inertia, becomes paramount. Assuming we have good friction models, the inertial parameters of the robot links and the motor inertia are termed as inertial parameters of the whole robot, which is the focus of this research thesis.

In most identification routines, there are three common similarities/features: modeling the dynamics, designing excitation trajectories, and using an estimation technique. In modeling, robot dynamics can be derived using either Lagrangian or Newton-Euler formulation. Excitation trajectories ensure persistence in the excitation of the dynamic parameters, resulting in a well-conditioned regressor matrix. Finally, estimation techniques help to build inferences on the estimated dynamic parameters. In this thesis, the focus is on the first two features. Since the approach is directed to find the inertial parameters, the robot dynamics are simplified to find a second-order equation in terms of inertial parameters and friction parameters under a set of assumptions made in choosing the excitation trajectories. The objectives of this thesis are: to obtain an analytical model of the load-side inertia at each axis, to find a collision-free set of configurations of the robot, and to find the optimal ones from that set to optimize the condition number in order to achieve well-conditioned regressor matrix. The theory is presented in the clearest yet rigorous fashion while providing enough advanced material and references so that the reader is prepared to contribute new material to the state-of-the-art.

In detail, the following sub-tasks were performed:

- Researched about analytical modeling of robot dynamics and found ways to model the load-side inertia.
- Familiarized with the initial parameter-identification toolbox built; the updated one is here (https://git-ce.rwth-aachen.de/robotmachining/robotmodel/parameter-identification).
- Developed functions based on algorithms 4.1- 4.8 that give a symbolic form of the load-side inertia in a minimal model form.
  - General-purpose Kinematic Transform (kinematicTransform) - an algorithm that helps to do homogeneous transformations.
  - Load-side Inertia (inertiaTransform) - initial load-side inertia at a particular axis input.
  - Axiswise Linear Systems (axisLinSys) - calculates an axiswise matrix equation of the load-side inertia from inertiaTransform.
  - Eliminate repeating Base Parameter (eliminateRepeatBaseParam) - eliminates re-

peating base parameters.

  - Pre-Minimal Model (preMinimalModel) - combines all axiswise matrix equations into a single one.
  - Find Linearly Dependent Base Parameters (findLinDep) - finds linearly dependent base parameters.
  - Coefficient multiplication and division (mulbf_divbp) - Normalizes the base parameters and base functions.
  - Calculate Minimal Model (calcMinimalModel) - calculates the final minimal model of the load-side inertia.

- Developed a live script named Base_Parameter_Calculations.mlx for ease of understanding of how to model the load-side inertia.
- Researched about condition number optimization pertaining to designing excitation trajectories and subsequent estimation theory.
- Developed functions based on algorithms 5.1- 5.3 that give a concise set of configurations that makes the regressor matrix from the minimal model well-conditioned and ready for designing excitation trajectories to find the inertia.
  - Generate Poses(generatePoses) - used for discretization or generating random configurations of the robot.
  - Generate Target Matrix (generateTargetMatrix) - generating an ideal target matrix for conditioning of regressor matrix.
  - Condition Number Optimization (condNumOpt) - gives the final set of well-conditioning configurations.
- Developed live scripts named Create_collision_free_poses.mlx and Condition_Number_Optimization_sir for ease of understanding of how to get the optimum set of configurations.
- Documentation of thesis.

III Acknowlegdements

I highly and truly acknowledge the stage provided by Mr. Lukas Gründel, and Ms. Minh Trinh to do my research. I feel honored to be part of Werkzeugmaschinenlabor (WZL) for the duration of my research. Also, many thanks to Mr. Carsten Reiners for all the help during this research.

IV Symbols and Abbreviations

Symbol	Unit	Description
n		Number of links
q	rad	Angular position
$\dot{q}$	$\frac{rad}{s}$	Angular velocity
$\ddot{q}$	$\frac{rad}{s^2}$	Angular acceleration
$\mathbf{q}$	rad	Angular position trajectory
$\dot{\mathbf{q}}$	$\frac{rad}{s}$	Angular velocity trajectory
$\ddot{\mathbf{q}}$	$\frac{rad}{s^2}$	Angular acceleration trajectory
q_{lb}	rad	Lower bound of angular position
$\dot{q}_{lb}$	$\frac{rad}{s}$	Lower bound of angular velocity
$\ddot{q}_{lb}$	$\frac{rad}{s^2}$	Lower bound of angular acceleration
q_{ub}	rad	Upper bound of angular position
$\dot{q}_{ub}$	$\frac{rad}{s}$	Upper bound of angular velocity
$\ddot{q}_{ub}$	$\frac{rad}{s^2}$	Upper bound of angular acceleration
J_{AX_k}	kgm^2	Total inertia at axis k
τ_{ck}	Nm	Torque due to Coriolis and centripetal forces
τ_{gk}	Nm	Torque due to gravity forces
τ_{fk}	Nm	Torque due to Coulomb-viscous friction forces
τ_k	Nm	Torque applied
m_i	kg	Mass of link i
m_iX_i, m_iY_i, m_iZ_i	kgm	First moments of inertia of link i
ϕ_i		Inertial paramters of link i
Γ	Nm	Torque at all joint
Φ_{min}		Base parameters
$Y(\Gamma)$	Nm	Torque measured along the trajectories
$W(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$		Regressor matrix
$W(\mathbf{q})$		New regressor matrix
U		Orthonormal basis for the column space of $W(\mathbf{q})$
Σ		Square diagonal matrix of singular values
V		Orthonormal basis for the row space of $W(\mathbf{q})$

Symbol	Unit	Description
$Y(J_{AX})$	kgm^2	Inertia measured at optimal configurations **q**
ρ		zero-mean additive noise
r		number of rows in the regressor matrix
n_c		number of parameters to estimate
$\hat{\Phi}_{min}$		Estimated parameters vector
$\hat{\Phi}_{min,j}$		j^{th} element of the estimated parameters vector
$\mathbf{C}_\rho$		variance-covariance matrix
σ_ρ^2		error variance in the estimated parameters
$\sigma_{\hat{\Phi}_{min,j}}$		standard deviation of the estimation error of j^{th} parameter
$\sigma_{\hat{\Phi}_{min,jr}}$		relative standard deviation of the estimation error of j^{th} parameter
p, q		Two points on the rigid body
$\tilde{p}^i$		Point p in homogeneous coordinates in frame i
$\tilde{r}_{j,i}^i$		vector from origin p to point q
${}^iT_j$		Homogeneous transformation matrix
${}^iR_j$		Rotation matrix
$r_{j,i}^i$		Translation vector
$J_{AX_k}(q_{k+1}, \ldots, q_n)$	kgm^2	Inertia at joint k
$J_{I,i}^k$	kgm^2	Link inertia tensor transformed into joint k coordinate system
J_{mls}^k	kgm^2	Inertia of the motor of the driving joint k
u_k		gear-ratio
z_k		vector of the rotational axis
m_i	kg	mass of the link i
$\tilde{v}_i^k$		Skew symmetric matrix
I_i^i		Link inertia in coordinate frame i
$bfVals$		Array of base functions evaluations
$identMod$		Modified identity matrix
$penalty$		penalizing parameter

Symbol	Unit	Description
$condArr$		Array of condition number corresponding to the penalty
$obsMat$		Regressor Matrix
$configMat$		Optimal Configurations Matrix

Abbreviations	Descriptions
IR	*Industrial Robot*
ISO	*International Organisation for Standardization*
RNE	*Recursive Newton-Euler*
SVD	*Singular value decomposition*
TCP	Tool center point
COG	Center of gravity

V List of Figures

VI List of Tables

VII List of Algorithms

1 Introduction

1.1 Motivation

Computing the dynamic model of a robot manipulator is of interest both for simulation and control. For the former problem, the direct dynamics of the manipulator is of significance, i.e., from a given set of joint torques, compute the resulting joint positions, velocities, and accelerations. For the latter problem, the inverse dynamics of the manipulator is of significance, i.e., from a given set of joint positions, velocities, and accelerations, compute the resulting joint torques. The computational load of direct dynamics is expected to be larger than that of inverse dynamics because the dynamic model naturally gives the mapping from the joint positions, velocities, and accelerations to the joint torques.

Simulations have become fundamental in robotics; they are almost always cheaper and faster than actual experiments, and simulated experiments can run without human input. Moreover, the variables in a simulated environment are more controllable than in the real world, and accurate measurements of any variable can be obtained without spending resources on expensive setups and sensors. In hindsight, accurate predictions of given system behavior in simulations can facilitate robust controllers for mapping real-time system behavior. Therefore, simulations and control complement each other, meaning simulations facilitate control strategies, and better controllers facilitate better simulation modeling strategies.

Hence, the simulation and control of robots require the development of different mathematical models. Several levels of modeling: geometric, kinematic, and dynamic, are needed depending on the objectives, the constraints, and the desired performance of the task. Obtaining these models is a challenging task. The difficulty varies according to the complexity of the kinematics of the mechanical structure and its degrees of freedom. In order to use these models in simulation and control, efficient and easy-to-use algorithms are required to estimate the values of geometric and dynamic parameters that help us capture the complexity of the robot.

Thus, the need for better methods of dynamic parameter estimation has become paramount because robots are becoming more complex, and their interactions with the unmodelled world are becoming more frequent. In machine tooling, industrial robots(IR) are seen as promising alternatives, such as for robot machining and human-robot collaboration, to name a few. Hence, these methods should be fast, reliable, and easily adaptable to various classes of robots. The dynamic model parameters of the IR refer to the inertial parameters of the robot links, Coulomb-viscous friction parameters, and motor inertia. The inertial parameters of the robot links and the motor inertia are termed as inertial parameters of the whole robot, which will be the focus of this research project thesis.

1.2 Objectives

In this thesis, the main objective is to present the mathematical and algorithmic approaches, to the best possible, of finding the optimal set of configurations of the IR for the eventual estimation of the inertial parameters, based on the known description of mechanisms used (i.e., geometric parameters), to allow a uniform approach for serial IRs. The approaches that are presented are

all given with the goal of eventual implementation in software of any choice.

In order to achieve the main objective, there are three significant sub-objectives. The first sub-objective is to obtain an analytical model of the load-side inertia at each axis of the IR. Further, adopting a new strategy to excite the dynamics of the IR using frequency-based excitation trajectories leads us to find optimum configurations of the IR. In this regard, the second sub-objective is to find a collision-free set of configurations used for the third sub-objective, that is, to find the optimal ones from that set to optimize the condition number at each axis, thus achieving our primary objective. These optimal configurations are used further to set up the measurement and subsequent estimation sub-routines of the identification routine of the inertial parameters.

1.3 Structure

Chapter 2 introduces the industrial robot used for this research and their existing approaches to identify its inertial parameters. The first part in the identification routine, commonly coined as *modeling of dynamics*, is explained in detail along with the existing modeling approaches and which one this research focuses on. The second part, called the *strategy to excite*, is further discussed in brief, explaining which strategies to opt for to ensure better excitation of the parameters needed to be estimated. The last part is the *adoption of an estimation technique* where the most well-known estimation technique of singular value decomposition along with an evaluation metric is introduced.

Based on the discussion of the state-of-the-art approaches, chapter 3 derives the solution approach for this research work. A simple flowchart in this chapter summarizes the whole research approach adopted to achieve the final objective. Chapter 4 discusses in brief how to model the load-side inertia first and then later, with the help of introduced algorithms, how to boil down the load-side inertia into a linear model. Chapter 5 uses the regressor matrix of the linear model derived in the previous chapter, and derives and explains the procedure on how to solve the constrained optimization problem to find the optimal configuration to minimize the condition number of the regressor matrix.

Chapter 6 gives a detailed overview of the algorithms implemented in MATLAB. The code has to ensure further complexities that the skeletal algorithms introduced in chapters 4 and 5 do not capture that well, but it served to be the basis for further synthesis. Hence, in the end, chapter 7 ends with critical remarks on the overall approach and gives directions for further research in this identification field.

2 State-of-the-art

This chapter provides the theoretical background for the development of the inertial parameter identification routine for the serial IR. At first, the robotic manipulator used for the purpose of the research is introduced. Further, the three common features of most identification routines are discussed. Next, from the perspective of classical robotics, how the dynamic modeling of the serial IR is expressed as a linear combination of its dynamic parameters is presented. It is followed by describing how a new strategy for designing the exciting trajectories helps in obtaining a regressor matrix dependent only on configurations of the IR, thus minimizing the effect of noisy measurements as well as errors in analytical modeling caused due to approximating velocities and accelerations if the IR is equipped with only position sensors. Lastly, a most widely used estimation technique for finding a physically plausible set of base parameter solution for the overdetermined linear system is discussed.

2.1 Industrial Robots

According to *ISO 8373:2021*, “an IR is an automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or fixed to a mobile platform for use in automation applications in an industrial environment." [ISO21] The IR includes: manipulator mechanism (including robot actuators), controlled by the robot controller (the means by which to teach and/or program the robot), and any communications interface (hardware and software). IRs can be viewed as a multibody systems having a fixed number of degrees of freedom, a set of minimal coordinates required to represent its configuration, usually corresponds to the number of non-redundant joints calculated using Chebychev–Grübler–Kutzbach criterion. Structure of most IRs can be distinguished into open-chain mechanisms (serial manipulators) or closed-chain mechanisms (parallel manipulators). A serial manipulator can be defined as consisting of a single chain of links connected by joints from the base to the end-effector, whereas parallel manipulators have one or more closed-loops connecting the base to the end-effector. The focus is on IRs designed using serial mechanisms, i.e., serial IRs. The implementation of the research is performed on the robot MABI

Figure 2.1: MABI MAX 100

Max 100 of the Swiss company MABI. It consists of six links connected by six revolute joints forming an open-chain. It has six degrees of freedom which can be confirmed using the Chebychev–Grübler–Kutzbach criterion.

2.2 Common features in Identification Routines

Dynamic modeling of an n-link serial rigid-body chain, representing a robotic manipulator or other mechanical system, is an important subject which has received thorough treatment in literature, particularly in the 1980s and 1990s [KHAL86; KHAL11; KHAL04c; KHAL04b; NAKA89; HOLL80; FEAT83; BOYE98; PARK95]. Dynamic models are an important tool for many control algorithms and have applications for feed-forward dynamic compensation, adaptive control [CRAI87], state estimation algorithms, motion planning, manipulator design and much more. These methods highly depend on the accuracy of the dynamic model, which in turn, depends on accurately identifying the dynamic parameters. Though system identification itself is widely studied in many other fields, in robotics, it utilizes the linearity of inertial parameters in the equations of motion of a robot [VAND95], a useful form of dynamics called as *regressor* form (equation (3)). Most identification routines share these three common features [GRÜN21]:

1. Use of a **model**, e.g., inverse dynamics, etc.
2. A **strategy to excite** the dynamics ,e.g., position, velocity, and acceleration trajectories.
3. Adaptation of an **estimation technique**, e.g., least square estimate, etc.

2.3 Modeling of the Industrial Robot Dynamics

The first common feature from section (2.2), in most literature, is coined as *modeling*. Models are among the most essential tools in robotics, such as kinematic and dynamic models of the robot's own body and controllable external objects. It is widely believed that intelligent mammals also rely on internal models in order to generate their actions. Likewise, accurate models of the system and its environment are crucial for planning, control, and many other applications in robotics. Classical robotics relies on manually generated models that are based on human insights into physics. However, with the available plethora of machine learning techniques, nowa-days modeling approaches can be classified in three distinct groups [GEIS20]:

1. **Analytical models** (white-box models), which consist of functional relationships motivated by first-principles, also called physics-related models.
2. **Data-driven models** (black-box models), which solely models using available data.
3. **Analytical structured models** (gray-box models), which combines analytical and data-driven models, also called hybrid models sometimes.

Since 1980s, most widely adopted is the analytical model; in the following part of the thesis we are going to adopt this modeling approach. Robot dynamics can be derived using either Lagrangian formulation or Newton-Euler formulation (refer table 2.1). Since, the interest is in modeling the non-conservative forces as well, it is well known that taking the Newton-Euler approach, the dynamic model of a serial IR with n moving links, under the assumption of no external forces acting on the end-effector boils down to the following equations of motion [GRÜN21]:

$$J_{AX_k}(q_{k+1}, \ldots, q_n)\ddot{q}_k + \tau_{ck}(q_k, \dot{q}_k) + \tau_{gk}(q_k) + \tau_{fk}(q_k, \dot{q}_k) = \tau_k \tag{1}$$

which is a second-order vector differential equation for the motion, where $k = \{1, ..., n\}$, q_k, $\dot{q}_k$ and $\ddot{q}_k$ are the angular position, velocity and acceleration respectively of joint k, $J_{AX_k}(q_{k+1}, ..., q_n)$ is the total inertia effecting axis k, $\tau_{ck}(q_k, \dot{q}_k)$ is the torque due to Coriolis and centripetal forces, $\tau_{gk}(q_k, \dot{q}_k)$ is the gravity torques and $\tau_{fk}(q_k, \dot{q}_k)$ is the friction-related torques (which comprises of the Coulomb-viscous friction forces). All of the terms on the left hand side of equation (1)

Lagrangian Formulation	**Newton-Euler Formulation**
Motion is described by energies	Motion is described by forces
Based on the principle of least action	Based on Newton's laws of motion
Generalized coordinates are used instead of constraint forces	Involves constraint forces
Conservation laws can be derived easily (Noether's theorem)	Does not have a systematic method for deriving conservation laws
Does not use vectors (energy being a scalar)	Uses vectors
Not ideal for non-conservative forces (such as friction)	Can handle non-conservative forces quite well

Table 2.1: Comparison between Lagrangian and Newton-Euler Formulations

together add up to the applied torque τ_k. If we arrange equation (1) by assuming that we have good friction models, the terms on the left hand side of the following

$$J_{AX_k}(q_{k+1}, \ldots, q_n)\ddot{q}_k + \tau_{ck}(q_k, \dot{q}_k) + \tau_{gk}(q_k) = \tau_k - \tau_{fk}(q_k, \dot{q}_k) \tag{2}$$

can be expressed linearly with respect to $(n_s \times 1)$ vector of standard inertial parameters Φ using $D(q, \dot{q}, \ddot{q})$. The right hand side can be summarized as a $(n \times 1)$ vector of non-conservative generalized forces acting on the system denoted by Γ,

$$\begin{gathered} D(q, \dot{q}, \ddot{q})\Phi = \Gamma \\ D(q, \dot{q}, \ddot{q}) \begin{bmatrix} \phi_1 & \ldots & \phi_n \end{bmatrix}^T = \begin{bmatrix} \tau_1 & \ldots & \tau_n \end{bmatrix}^T \end{gathered} \tag{3}$$

Considering a serial IR with n moving links, we get $n_s = 11n$ standard inertial parameters. For illustration, each link i has mass m_i, 3 components of the first moment (m_iX_i, m_iY_i, m_iZ_i), 6 components of the inertia tensor $(XX_i, XY_i, XZ_i, YY_i, YZ_i, ZZ_i)$, and the inertia of the motor I_{mot_i} as shown below in equation (4).

$$\phi_i = [m_i, m_iX_i, m_iY_i, m_iZ_i, XX_i, XY_i, XZ_i, YY_i, YZ_i, ZZ_i, I_{mot_i}]^T \tag{4}$$

Obtaining such linear models have been highly revered in nonlinear system identification research; chiefly because they are easy to solve. Further, equation (3) is reduced to a global

form, expressed in terms of its base parameters, denoted by Φ_{min}:

$$\Gamma = D(q, \dot{q}, \ddot{q})\Phi_{min} \tag{5}$$

These base parameters Φ_{min} are a set of minimum number of linearly combined inertial parameters from Φ. Much research has been performed to determine the best way to derive Φ_{min} in a computationally efficient manner. Earlier, the symbolic derivation of dynamic models was done through the symbolic Lagrange formulation [UICK69; KAHN71]. However, since then, significant work has reduced the computational requirements. In 1980, Kane presented an alternative symbolic method, which came to be known as Kane's method [KANE80], and which allowed for easier derivation and fewer computations compared to the Lagrange method. Also in 1980, the recursive Newton-Euler (RNE) method, proposed by Luh et al. [LUH80], was introduced. The RNE method is a recursive algorithm with linear complexity $O(n)$. This complexity is a significant improvement over the recursive form of the Lagrange formulation, which had $O(n^4)$ complexity, proposed by Hollerbach [HOLL80].

However, none of these formulations were expressed in regressor form, making dynamic modeling difficult. Variants have since been proposed to the RNE method, which do return a regressor matrix, such as in the work of Atkeson et al. [ATKE86] or Khalil and Kleinfinger [KHAL87]. Additionally, rules for identifying the base inertial parameters were developed by Mayeda et al. [MAYE90], Gautier [GAUT90], and others, and integrated into symbolic formulations [KANA84].

The existing work can be categorized into two main subsets: numerical and symbolic methods. Numerical methods, such as those based on the RNE algorithm, are fully procedural algorithms taking inputs as kinematic parameters (such as a Denavit-Hartenberg table) and inertial parameters. These methods are simple to use since they need only be programmed once and can subsequently be used with any manipulator, given only some set of kinematic and inertial parameters. However, they may perform redundant computations since they do not take advantage of any possible simplifications of the dynamic model, which arise due to alignment in the mechanical system structure or due to zero-valued kinematic and inertial parameters.

Symbolic algorithms, conversely, leverage symbolic simplification routines to eliminate redundant terms and simplify the calculations, resulting in a lower computational complexity [KHAL04a]. These methods are more difficult to use than the numerical methods since a derivation step is required before the model can be used. As the dynamic models of a manipulator are exceedingly complex, this derivation can be quite lengthy. In this thesis, symbolic simplification methods are adopted, well-described in chapter 4, to obtain a reduced global form of equation (5), that can be further leveraged to design a condition number optimization routine.

2.4 Excitation Trajectories

By sampling equation (5) at e different sample times from an applied trajectory to the robot and assuming there are neither modeling errors nor perturbations, one obtains a compatible overdetermined linear system of r equations and n_c (number of elements of Φ_{min}) unknowns,

$$Y(\Gamma) = W(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\Phi_{min} \tag{6}$$

$$\text{where, } \quad Y = \begin{bmatrix} \Gamma_1(1:e) & \dots & \Gamma_n(1:e) \end{bmatrix}^T, \quad W(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \begin{bmatrix} D_1(1:e) & \dots & D_n(1:e) \end{bmatrix}^T$$

where, r is the number of equations (5) such that $r = n \cdot e \geq n_c$, $\Gamma_k(1:e)$ is the $e \times 1$ torque measurements of joint k, $D_k(1:e)$ is $e \times n_c$ matrix built by sampling the k^{th} row of $D(q, \dot{q}, \ddot{q})$, $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ defines the sampled joint positions, velocities and accelerations for each axis k from the trajectory, and W is the regressor matrix. Thus, the second common feature of adopting a strategy to find this exciting trajectory gives us a well conditioned W. But, it poses the challenge to find these trajectories, and it is proven to be a common problem in the field of parameter identification.

Different approaches has been proposed in the literature. [PFEI95] proposed an online optimization of the trajectory by minimizing the condition number of the regressor matrix. In [GAUT91; KHAL07; PRES93] and [GAUT97] the condition number of the dynamic energy model is optimized off-line to obtain a set of optimum points. In the second step, these optimum points are interpolated using fifth-order polynomial functions assuming zero initial and final velocity and acceleration. The interpolation is obtained by a further optimization problem to fit acceleration limits in all joints. Following, a third optimization to fit all velocity constraints. Finally, a fourth optimization on position constraints are verified to fit all position limits. If they do not, a new sequence of optimum points with smaller constraints must be started. This trial-and-error algorithm that has some uncertainty in terms of finding practicable solutions can be summarized as follows,

$$\begin{aligned} \min_{\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}} \quad & cond(W(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})) \\ \text{s.t.} \quad & q_{lb} \leq \mathbf{q} \leq q_{ub} \\ & \dot{q}_{lb} \leq \dot{\mathbf{q}} \leq \dot{q}_{ub} \\ & \ddot{q}_{lb} \leq \ddot{\mathbf{q}} \leq \ddot{q}_{ub} \\ & internalCollision(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = 0 \\ & externalCollision(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, environment) = 0 \end{aligned} \tag{7}$$

Further, efficient techniques based on optimization to find the trajectories for robotics manipulators are detailed in [SWEV97; PARK06; RACK12]. While the recent optimization-based approaches are expected to be useful [BONN16], the large number of degree of freedom leads to computational complexity. For those reasons, popular approaches [VENT09; JOVI15] are such that, at first, several physically possible motions are generated in advance, and then, the optimal set of motions are selected from those candidates. Though it is simple and convenient, many motion candidates need to be generated without guarantee to obtain optimal trajectories. Accordingly, an efficient method to find the optimal trajectories is needed. Such an optimal trajectory is known as a persistently exciting trajectory. To obtain an exciting trajectory, two schemes are generally used: calculation of a trajectory satisfying some optimization criteria; and use of sequential sets of special test motions, where each motion will excite some inertial parameters. Above discussed methods generates trajectories for $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ (which populates the regressor matrix $W(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$), have a common goal to achieve a lower condition number on the pseudo inverse of the regressor matrix, which helps us to achieve lower variance in iden-

tification. All these methods also compute torques, using state-of-the-art sensor technologies, during the traversal of these trajectories.

In contrast, the strategy to design trajectories in this research thesis is frequency-based, especially short, constant, low velocity offset trajectories, where just one axis is moved at a time. Therefore, Coriolis effects do not occur and centrifugal forces are absorbed by the bearings, hence $\tau_{ck}(q_k, \dot{q}_k)$ can be neglected. The low velocity induce slow change of gravity forces leads to the assumption that gravity effects $\tau_{ck}(q_k, \dot{q}_k)$ correspond to frequencies close to $0Hz$ and thus can be neglected. For an undamped system, the angular velocity would follow the drive torque with a phase shift of -90°. Since the analysis of the phase response reveals a deviation from this assumption, frictional damping effects cannot be neglected. There are various ways to describe friction with all its complex aspects [BONA05]. An adequate and often used model is the combination of Coulomb and viscous friction [VAND95]. The constant velocity offset trajectories causes a force excitation in a constant direction and the discrete Fourier transformation of the Coulomb friction would generate a spectral line at $0Hz$ leaving only the viscous friction, represented by $\mu_{\nu k}$, to be included. The described assumptions lead to the following differential equation that helps us reduce the complexity of equation (1) as follows [GRÜN21]:

$$J_{AX_k}(q_{k+1}, \ldots, q_n)\ddot{q}_k + \mu_{\nu k}\dot{q}_k = \tau_k \tag{8}$$

By applying the Laplace transform to equation (8), leads to the following transfer function (we leave out the axis index k for simplicity). Further decomposing into amplitude and phase results in two equations:

$$\frac{\Omega}{\Gamma} = \frac{\mu}{\mu^2 + \omega^2 J^2} - i\frac{\omega J}{\mu^2 + \omega^2 J^2} \Longrightarrow \left|\frac{\Omega}{\Gamma}\right| = \frac{1}{\sqrt{\mu^2 + \omega^2 J^2}} \tag{9}$$

$$\psi = \arctan\left(\frac{\Omega}{\Gamma}\right) = \arctan\left(-\frac{\omega J}{\mu}\right) \Longrightarrow \mu = -\frac{\omega J}{\tan(\psi)} \tag{10}$$

Inserting equation (10) into equation (9) leads to the final equation of the total load-side inertia:

$$J = \frac{1}{\omega} \cdot \left|\frac{\Gamma}{\Omega}\right| \cdot \frac{1}{\sqrt{1 + \frac{1}{\tan^2(\psi)}}} \tag{11}$$

Here $\left|\frac{\Gamma}{\Omega}\right|$ is the reciprocal of the amplitude and ψ is the phase of the measured response of the frequency-based excitation trajectories. The motivation to compute inertia is two-fold. First, from equation (8), inertia is a function of only the joint configurations, q. It significantly reduces the numerical errors induced by approximating $\dot{q}$ and $\ddot{q}$, and gives an advantage to achieve numerical accuracy by just measuring joint positions using the position encoders. Second, all the inertial parameters are modelled in the inertia, J_{AX}, and still we can express them in the regressor form analogous to equation (5) as follows,

$$J_{AX} = D(q)\Phi_{min} \tag{12}$$

$$\begin{bmatrix} J_{AX_1} & \ldots & J_{AX_n} \end{bmatrix}^T = D(q) \begin{bmatrix} \phi_1 & \ldots & \phi_n \end{bmatrix}^T$$

The choice of trajectory greatly influenced the reduction of the complexity of the previous model. Obtaining equation (12) is precisely the first sub-objective of the research, as mentioned in the section (1.2). As $D(q)$ is dependent only on the joint configurations q, we can develop an offline condition number optimization routine to obtain a set of optimum configurations in the workspace of the IR, subjected to internal and external constraints similarly to [GAUT91; KHAL07; PRES93] and [GAUT97]. Analogous to equation (6) one obtains a compatible overdetermined linear system of r equations and n_c (number of elements of Φ_{min}) unknowns,

$$Y(J_{AX}) = W(\mathbf{q})\Phi_{min} \tag{13}$$

$$\text{where, } Y = \begin{bmatrix} J_{AX_1}(1:e_1) & \dots & J_{AX_n}(1:e_n) \end{bmatrix}^T, \; W(\mathbf{q}) = \begin{bmatrix} D_1(1:e_1) & \dots & D_n(1:e_n) \end{bmatrix}^T$$

where, r is the number of equations (13) such that $r = e_1 + \cdots + e_n \geq n_c$, $J_{AX_k}(1:e_k)$ is the $e_k \times 1$ torque measurements of joint k, $D_k(1:e_k)$ is $e_k \times n_c$ matrix built by sampling the k^{th} row of $D(q)$, $(\mathbf{q})$ defines the optimum joint positions for each axis k, and W is the regressor matrix. Thus, the second and third sub-objectives are inter-related that together optimizes the condition number of $D(q)$ at each axis k, and simultaneously populating it. So, the previous optimization problem is simplified as follows,

$$\begin{aligned} \min_{\mathbf{q}} \quad & cond(W(\mathbf{q})) \\ \text{s.t.} \quad & \mathbf{q}_{lb} \leq \mathbf{q} \leq \mathbf{q}_{ub} \\ & internalCollision(\mathbf{q}) = 0 \\ & externalCollision(\mathbf{q}, environment) = 0 \end{aligned} \tag{14}$$

At each of these optimal configurations for that particular axis k, a few particular base parameters from Φ_{min} are persistently excited using an offset sinusoidal excitation trajectory (detail explanation in [GRÜN21]). From each of these trajectories (corresponding to each configuration), computing the inertia from its magnitude and phase response using equation (11) populates $Y(J_{AX})$ of equation (13).

2.5 Estimation Techniques

The third common feature from section (2.2), i.e, selection of estimation techniques, solely depends on the modeling step. Since an analytical approach to model the dynamics is chosen, the best approach that fits well is the least squares estimation (LSE); especially single value decomposition (SVD) is applied to find both the minimum norm and least squares solution. The rectangular system of equations (13), $Y(J_{AX}) = W(\mathbf{q})\Phi_{min}$ has two possible difficulties: *dependent columns* and *dependent rows* in $W(\mathbf{q})$. If $W(\mathbf{q})$ has dependent columns then $W^T(\mathbf{q})W(\mathbf{q})$ is not invertible and $\hat{\Phi}_{min}$ is not determined, because any vector from the nullspace can be added to $\hat{\Phi}_{min}$. With dependent rows, $Y(J_{AX}) = W(\mathbf{q})\Phi_{min}$ may have no solutions. That happens when $Y(J_{AX})$ is outside the column space of $W(\mathbf{q})$. Therefore, instead of $Y(J_{AX}) = W(\mathbf{q})\Phi_{min}$, we solve $W^T(\mathbf{q})Y(J_{AX}) = W^T(\mathbf{q})W(\mathbf{q})\hat{\Phi}_{min}$ and choose $\hat{\Phi}_{min}$ such that the vector $W(\mathbf{q})\hat{\Phi}_{min}$ is closest to $Y(J_{AX})$. The core of SVD is exactly to solve these two

difficulties. In SVD, the regressor matrix $W(\mathbf{q})$ is decomposed as follows,

$$W(\mathbf{q}) = U\Sigma V^T \tag{15}$$

where the columns of U provide an orthonormal basis for the column space of $W(\mathbf{q})$ and columns of V provide an orthonormal basis for the row space of $W(\mathbf{q})$. Σ is a square diagonal matrix of size $r \times r$ (r is the rank of $W(\mathbf{q})$) consisting of singular values arranged in the descending order by magnitude, and thus the columns of U are hierarchically ordered by how much correlation they capture in the columns of $W(\mathbf{q})$; V similarly captures correlation in the rows of $W(\mathbf{q})$. Using SVD, the solution to $\hat{\Phi}_{min}$ which is usually the pseudoinverse as follows can be also expressed in terms of U, V, and Σ by substituting equation (15),

$$\begin{aligned}\hat{\Phi}_{min} &= \underbrace{(W^T(\mathbf{q})W(\mathbf{q}))^{-1}W^T(\mathbf{q})}_{V\Sigma^{-1}U^T}\ Y(J_{AX}) \\ &= V\Sigma^{-1}U^T\ Y(J_{AX})\end{aligned} \tag{16}$$

But, the use of SVD is not just to find the solution for the overdetermined systems of equations but also to evaluate the estimated parameters. For, the level of excitation of each parameters, we use the 2-norm condition number, which depends on the smallest and largest singular value of $W(\mathbf{q})$, that is already ensured in the third sub-objective. The significance of smaller condition number is that it increases the excitation of each parameter and lowers the impact of disturbances and noise on the estimation. Furthermore, the accuracy of an estimated parameter can be expressed with the relative standard deviation. Since $W(\mathbf{q})$ is deterministic as we have accurate measurement of joint positions using the position encoders, the only assumption made here is that the inertia error ρ is a zero-mean additive independent Gaussian noise with σ_ρ as standard deviation. The variance-covariance matrix $\mathbf{C}_\rho$ can be calculated as [KHAL04a],

$$\mathbf{C}_\rho = E(\rho\rho^T) = \sigma_\rho^2\mathbf{I_r} \tag{17}$$

E is called the expectation operator. $\mathbf{I_r}$ is the $(r \times r)$ identity matrix and σ_ρ is given by the unbiased estimation

$$\sigma_\rho^2 = \frac{\|\mathbf{Y(J_{AX})} - W(\mathbf{q})\hat{\Phi}_{min}\|^2}{r - n_c} \tag{18}$$

Thus, the variance- covariance matrix of the estimation error is given as

$$\begin{aligned}\mathbf{C}_{\hat{\Phi}_{min}} &= \mathbf{E}[(\Phi_{min} - \hat{\Phi}_{min})(\Phi_{min} - \hat{\Phi}_{min})^T] \\ &= W(\mathbf{q})\mathbf{C}_\rho(W(\mathbf{q})^+)^T = \sigma_\rho^2(W(\mathbf{q})^TW(\mathbf{q}))^{-1}\end{aligned} \tag{19}$$

With the variance-covariance matrix we can extract the standard deviation of the estimation error $\sigma_{\hat{\Phi}_{min,j}}$ ad its relative value $\sigma_{\hat{\Phi}_{min,jr}}$ as the (j,j) element of $\mathbf{C}_{\hat{\Phi}_{min}}$

$$\sigma_{\hat{\Phi}_{min,j}} = \sqrt{\mathbf{C}_{\hat{\Phi}_{min}}(j,j)} \tag{20}$$

$$\sigma_{\hat{\Phi}_{min,jr}} = 100 \frac{\sigma_{\hat{\Phi}_{min,j}}}{|\hat{\Phi}_{min,j}|} \tag{21}$$

A parameter is assumed to be poorly identified if the relative error deviation is in the area between 5 to 15% or above

3 Approach

Based on the discussion of the state-of-the-art approaches in the previous chapter, the solution approach of this research is derived in the following flowchart (see Figure 3.1).

First, the necessary data to model the kinematics of the robot model, here MABI Max 100 is acquired. Using that, we extract the homogeneous transformation matrices to capture the kinematics of the robot. Using these transformations, algorithm 4.1 is developed. Second, the load-side inertia as per equation (23) is derived using algorithm 4.2. Third, this initial load-side inertia model is further synthesized into a reduced linear model as per equation (12) using the algorithms 4.3– 4.8, following the procedure explained in figure 4.3.

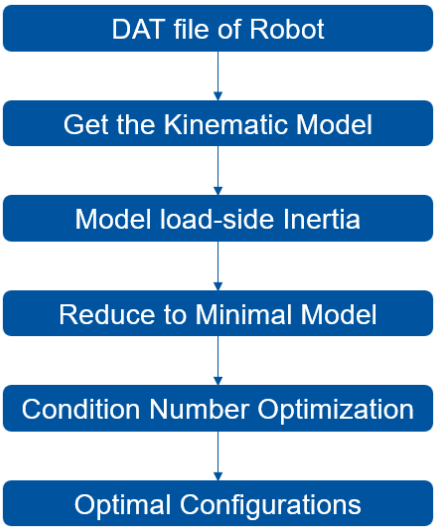

Figure 3.1: Research Approach

Lastly, the regressor matrix $D(q)$ from the reduced linear model (equation (12)) is further synthesized in the condition number optimization routine (chapter 5), using the algorithms 5.1– 5.3, to finally find the optimal configurations that will be used in the designing trajectories to excite the parameters to be estimated.

4 Analytical modelling of Load-side Inertia

The main target of modeling is the pursuit of finding a minimal model that realizes the physical processes in a system with a preferably small number of degrees of freedom and a minimum of model parameters. A minimal model leads to fundamental equations of motion that interpret the simulation results easily. Thus, with a minimum effort, the perception of the physical processes and the essential model parameters can be achieved. In general, a physical model that describes the characteristics of the technical system, which is responsible for the system performance, can be realized by different mathematical models. While creating the mathematical model, there is a necessity of specifying which physical effects and which characteristics are important and must be calculated.

Here, the interest is in the accurate prediction of the motion of the tool center point of a serial industrial manipulator, the physical effect of applying torques at the joints, and the characteristics that govern this physical effect are the inertial parameters that map the dynamics of the manipulator. Generally, mathematical models are a crude description of the real system. Thus, the final goal of mathematical modeling is to build up a model that is as simple as possible and as complex and elaborate as necessary. During the modeling process, complex characteristics of the real system have to be assigned to the characteristics of the mathematical model in a systematic approach such that it can easily be used for the subsequent analysis.

Sufficient knowledge of the behavior of the real system is necessary to choose the components which are included in these models. Therefore, the aim of the analysis influences the choice. Often, the influence of single effects on the behavior of the total system is desirable to know. Therefore, the mathematical model should comprise these effects in addition to the known effects with influence on the system's characteristics. In the following, section (4.1) describes the kinematic modelling of the *MABI MAX 100*. Further, section (4.2) describes the derivation of equation (8) from section (2.4) in detail with the underlying assumptions made. Next, section (4.3) explains the breakdown of each term that goes into modeling the load-side inertia in order to reduce it to a minimal model (equation (12)). For that purpose, the recommended algorithmic tools are discussed in the last section (4.4), where the last algorithm 4.8 combines all of the algorithms in the workflow described in the Figure 4.3.

4.1 Kinematic Model

Most kinematic modelling owes it to the development of the oldest branch of physics called *Classical mechanics*. Its purpose is to predict the future and reconstruct the past, to determine the history of every particle in the Universe. Classical mechanics is all about the motion of particles. In robotics, the interest is not in the motion of individual particles, but in the collective motion of a set of particles, more precisely the link of a robot manipulator [MURR17]. To this end, a perfectly rigid body can be loosely defined as a completely "undistortable" body. More formally, a *rigid body* is a collection of particles such that the distance between any two particles remains fixed, regardless of any motion of the body or forces exerted on the body. Thus, if p

and q are any two particles on a rigid body then, as the body moves, p and q must satisfy,

$$\|p(t) - q(t)\| = \text{constant}$$

A *rigid motion* of any rigid body is a continuous movement of the particles such that the distance between any two particles of the rigid body remains fixed at all times. The net movement of a rigid body from one location to another via a rigid motion is called a *rigid displacement*. In general, a rigid displacement may consist of both translation and rotation of the rigid body. In order to describe rigid displacements for a 6-axis robotic manipulator, perfectly defined reference frames are needed at first. There's always a a choice of origin, together with a set of axes

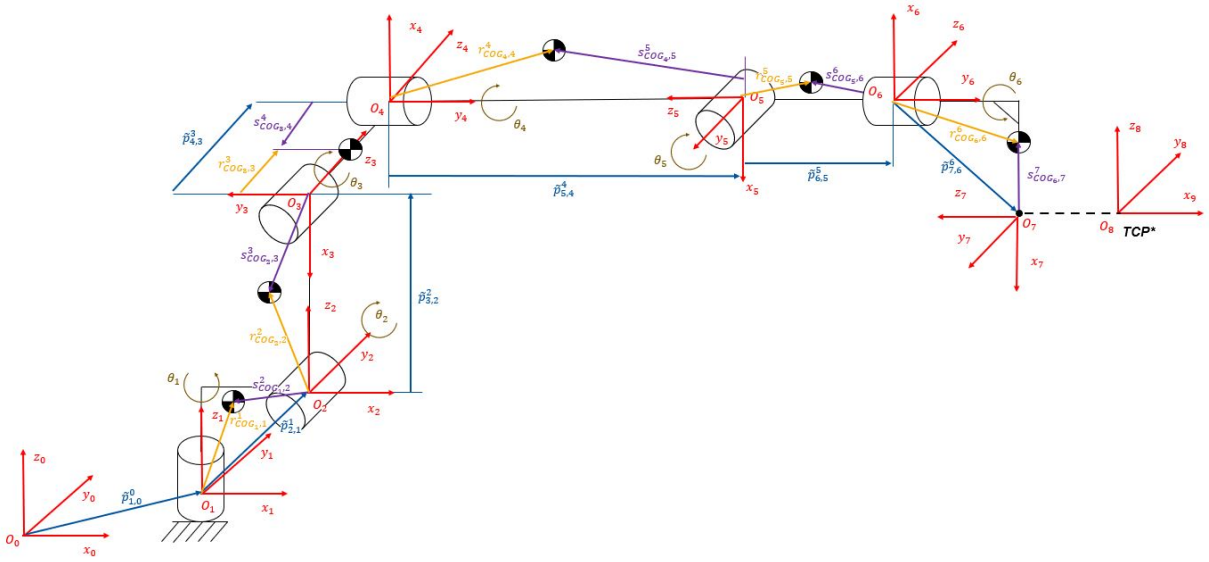

Figure 4.1: Kinematic model of MABI MAX 100

which helps us define *rigid body transformations*. Hence, for convenience, the following set of reference frames were defined (refer Figure 4.1) by placing them at the joints of the manipulator, chiefly because the joints are torque actuated. They induce a simple additional rotational transformation given there's no displacement of these reference frames, thus simplifying the further synthesis of the manipulator as a whole.

4.1.1 Homogeneous Transformations

Rigid body transformations essentially summarizes kinematics of manipulators by dealing with transformation of points and vectors, denoted by simple representation in terms of matrices and vectors in $\mathbb{R}^4$,

$$\tilde{p}^i = \begin{bmatrix} p_x^i & p_y^i & p_z^i & 1 \end{bmatrix}^T$$

These are called the *homogeneous coordinates* of the point p in the reference frame i. Let's assume that the i^{th} frame of reference is the origin. Considering point p is that origin, gives us $\tilde{p}^i = (0, 0, 0, 1)^T$. Considering another frame of reference j, we wish to know the distance of the origin q of the reference frame j from that of frame i. Vectors are the difference of points, which

helps us map this distance as follows.

$$\tilde{r}^i_{j,i} = \begin{bmatrix} r^i_{j,i} \\ r^i_{j,i} \\ r^i_{j,i} \\ 0 \end{bmatrix} = \tilde{q}^i - \tilde{p}^i = \begin{bmatrix} q^i_x \\ q^i_y \\ q^i_z \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} q^i_x \\ q^i_y \\ q^i_z \\ 0 \end{bmatrix}$$

The above vector r is the difference between the origin q and origin p in frame i (refer Figure 4.2). Note that the form of the vector is different from that of a point. The 0 and 1 in the

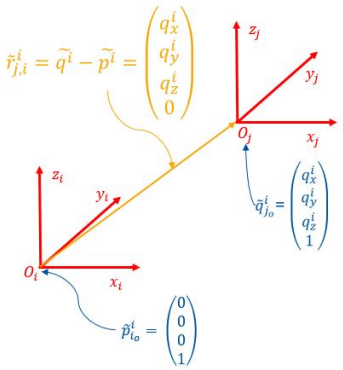

Figure 4.2: Points and vectors in homogeneous representation

fourth component of vectors and points, respectively, will remind us of the difference between points and vectors and enforce a few rules of syntax: sums and differences of vectors are vectors; the sum of a vector and a point is a point; the difference between two points is a vector; and the sum of two points is meaningless. Further, a relation between an arbitrary point s with respective to both frames i and j, i.e., s^i and s^j [MURR17] can be obtained. It can be expressed as an *affine* transformation:

$$\tilde{s}^i = {}^iT_j \; \tilde{s}^j$$

where ${}^iT_j$ is called a *homogeneous* transformation matrix that is denoted by following combination of rotation matrix ${}^iR_j$ and a translation vector $r^i_{j,i}$,

$${}^iT_j = \begin{bmatrix} {}^iR_j & r^i_{j,i} \\ 0 & 1 \end{bmatrix}$$

whereas an arbitrary vector $\tilde{s}^j_{b,a}$ is transformed just by rotation because of the 0 in its fourth component:

$$\tilde{s}^i_{b,a} = {}^iT_j \tilde{s}^j_{b,a} = {}^iR_j \tilde{s}^j_{b,a}$$

4.1.2 Chain of Transformations

Writing down the whole transformation chain from the base reference frame O_0 (which can be anything you want) to the frame of tool center point (TCP) O_8 can be thus easily expressed. An

initial guess would be to multiply all the transformations one after the other as follows:

$$^{0}T_{8} = \Big(\prod_{n=0}^{8} {}^{n}T_{n+1} \Big)$$

This is correct under the assumption that the joints are fixed and not actuated, but the manipulators loses its essence if the joints are not actuated. Hence, it is necessary to add the homogeneous rotation matrices in between the homogeneous transformations for accounting for the rotating joints as follows:

$$^{0}T_{8} = \Big(\prod_{n=0}^{5} {}^{n}T_{n+1} \; Rot_{n+1_{j\in[x,y,z]}} \Big) {}^{6}T_{8} \tag{22}$$

where $Rot_{n+1_{j\in[x,y,z]}}$ belongs to a set of homogeneous rotation matrices along the axis j of rotation for the joint $n+1$, so it depends on the actuated axis of the reference frame $n+1$. The following explains the compact form of $Rot_{n+1_{j\in[x,y,z]}}$.

$$Rot_{n+1_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Rot_{n+1_y} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{n+1_z} = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 & 0 \\ sin(\gamma) & cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation (22) can be expressed as a full set of transformations from the base frame O_0 to the TCP O_8, denoted as a *transformSet*,

$$tranformSet = \{ {}^{0}T_1, Rot_{1_z}, {}^{1}T_2, Rot_{2_y}, {}^{2}T_3, Rot_{3_z}, {}^{3}T_4, Rot_{4_y}, {}^{4}T_5, Rot_{5_y}, {}^{5}T_6, Rot_{6_y}, {}^{6}T_7, {}^{7}T_8 \}$$

The homogeneous rotation matrices of the above set are dependent on the configuration q of the manipulator, and the rest of the homogeneous transformations are constants that depend upon the geometric parameters and the choice of coordinate systems. This chain of transformations can be used as a benchmark to develop a more generalised algorithm that can transform from any starting reference frame to any end reference frame.

4.1.3 General-purpose Kinematic Transform Algorithm

The way of thinking of coordinate system transformations can be summarized into the following first algorithm called the *General-purpose Kinematic Transform*.

Algorithm 4.1 General-purpose Kinematic Transform (kinematicTransform)

Inputs: q, $\tilde{r}^{i}_{cog_i,o_i}$ transformSet, s, e, targetPoint, returnMode
Outputs: transform

$e = \begin{cases} e-1, & \text{if targetPoint = 'cog'} \quad or \quad \text{targetPoint = 'joint'} \\ 5, & \text{if targetPoint = 'tcp'} \end{cases}$

$transform = \begin{cases} {}^{0}T_1, & \text{if } s = 0 \\ I, & \text{otherwise} \end{cases}$

$s = \begin{cases} s+1, & \text{if } s = 0 \\ s, & \text{if otherwise} \end{cases}$

$transform = \begin{cases} transform * \Big(\prod_{n=s}^{e} Rot_{n_{j\in[x,y,z]}}(q(n))\ {}^{n}T_{n+1} \Big), & \text{if } e \neq 1 \quad and \quad e \leq 6 \\ transform * \Big(\prod_{n=s}^{e} Rot_{n_{j\in[x,y,z]}}(q(n))\ {}^{n}T_{n+1} \Big) * {}^{6}T_8, & \text{if targetPoint = 'tcp'} \end{cases}$

if targetPoint = 'cog' and e $\leq$ 6 **then**
  $rotationJoint \leftarrow Rot_{e_{j\in[x,y,z]}}(q(e))$
  $transform(:,4) \leftarrow transform(:,4) + transform * rotationJoint * \tilde{r}^{e}_{cog_e,o_e}$
end if
if returnMode = 'homogeneous' **then**
  $transform \leftarrow transform$
else if returnMode = 'rotation' **then**
  $transform \leftarrow transform(1:3, 1:3)$
else
  $transform \leftarrow transform(1:3, 4)$
end if
  return transform

The inputs for the algorithm are as follows: $q = [q_1, q_2, q_3, q_4, q_5, q_6]$, the configuration of the IR; $\tilde{r}^{i}_{cog_i,o_i}$ is the set of vectors of center of gravity (COG) of i^{th} link with respective to the i^{th} frame; $tranformSet = \{{}^{0}T_1, Rot_{1_z}, {}^{1}T_2, Rot_{2_y}, {}^{2}T_3, Rot_{3_z}, {}^{3}T_4, Rot_{4_y}, {}^{4}T_5, Rot_{5_y}, {}^{5}T_6, Rot_{6_y}, {}^{6}T_7, {}^{7}T_8\}$; s $\in \{0, 1, .., 6\}$; e $\in \{1, \ldots, 6, 7\}$, where 7 is the index for tool center point (TCP); targetPoint $\in$ {'joint', 'cog', 'tcp'}, whether you want to transform to the joint or to the COG, or the TCP; returnMode $\in$ {'homogeneous', 'rotation', 'translation'}, whether you want the output as either homogeneous, rotation, or translation part of the overall transformation. This algorithm will be invoked in the following modelling of the load-side total inertia.

4.2 Dynamic Model

The field of dynamics consists of kinematic equations describing the motion of a system via position, velocity, and acceleration, as well as kinetics, which evolves around the causal effects of generalized forces on the motion of body masses. Having a thorough understanding of the system's kinematics, when it comes to analytical modeling of the dynamics, further interest lies in utilizing knowledge on the causal relationships between the system's mass, impressed forces, and its motion. In most modern analytical mechanics textbooks [FEAT14], the term dynamics is used as a direct substitute for kinetics. Here a simplification of the dynamic equations (1) to equations (8) is revisited first and the necessary algorithms are unfolded. Revisiting

equation (1),

$$J_{AX_k}(q_{k+1},\ldots,q_n)\ddot{q}_k + \tau_{ck}(q_k,\dot{q}_k) + \tau_{gk}(q_k) + \tau_{fk}(q_k,\dot{q}_k) = \tau_k$$

which is a second-order vector differential equation for the motion, where $k = \{1,\ldots,n\}$, q_k, $\dot{q}_k$ and $\ddot{q}_k$ are the angular position, velocity and acceleration respectively of joint k, $J_{AX_k}(q_{k+1},\ldots,q_n)$ is the total inertia effecting axis k, $\tau_{ck}(q_k,\dot{q}_k)$ is the torque due to Coriolis and centripetal forces, $\tau_{ck}(q_k,\dot{q}_k)$ is the gravity torques and $\tau_{fk}(q_k,\dot{q}_k)$ is the friction-related torques (which comprises of the Coulomb-viscous friction forces). All of the terms on the left hand side of equation (1) together add up to the applied torque τ_k. Using the adopted strategy to excite using frequency-based approaches, just one axis is moved at a time. Therefore, Coriolis effects do not occur and centrifugal forces are absorbed by the bearings, hence $\tau_{ck}(q_k,\dot{q}_k)$ can be neglected in further simplification. The low velocity trajectories and therefore slow change of gravity forces leads to the assumption that gravity effects $\tau_{ck}(q_k,\dot{q}_k)$ correspond to frequencies close to $0H$ and therefore can be neglected here. For an undamped system, the angular velocity would follow the drive torque with a phase shift of -90°. Since the analysis of the phase response reveals a deviation from this assumption, frictional damping effects cannot be neglected. There are various ways to describe friction with all its complex aspects [BONA05]. An adequate and often used model is the combination of Coulomb and viscous friction [VAND95]. Due to the constant velocity offset during the experiments, which causes a force excitation in a constant direction the discrete Fourier transformation of the Coulomb friction would generate a spectral line at 0 Hz, so that only the viscous friction, represented by $\mu_{\nu k}$, has to be included. The described assumptions lead to the following differential equation (8),

$$J_{AX_k}(q_{k+1},\ldots,q_n)\ddot{q}_k + \mu_{\nu k}\dot{q}_k = \tau_k$$

4.3 Load-side Total Inertia Model

Applying the Laplace transform to equation (8) the load-side inertia at each axis ca be measured using equations (9)(10), and (11). This section focuses on modelling the load-side inertia, $J_{AX_k}(q_{k+1},\ldots,q_n)$. The load-side total inertia at axis k, i.e., $J_{AX_k}(q_{k+1},\ldots,q_n)$ is composed of the sum of the following link inertia tensors $J^k_{I,i}$ (transformed into the corresponding joint coordinate system), the inertia due to the parallel displacement of the rotational axes described by the Huygens-Steiner theorem $J^k_{HS,i}$, and load-side inertia of the motor J^k_{mls} (expressed by the inertia of the motor on the drive side $J_{mms,k}$ multiplied by the square of the gear ratio u_k). Therefore, the load-side total inertia at an axis k leads to equation (23), where z_k is the vector of the rotational axis, R^k_{ki} is the rotation matrix of coordinate frame k to i, m_i is the mass of the link i, $\tilde{v}^k_i$ is the skew symmetric matrix, and I^i_i is the inertia tensor described in coordinate frame i.

$$\begin{aligned} J_{AX_k}(q_{k+1},\ldots,q_n) &= \sum_{i=k}^{n}\left(J^k_{I,i} + J^k_{HS,i}\right) + J_{mls,k} \\ J_{AX_k}(q_{k+1},\ldots,q_n) &= \sum_{i=k}^{n}\left(z'_k(R^k_{ki}I^i_i R^{k'}_{ki} + m_i\tilde{\nu}^k_i\tilde{\nu}^k_i)z_k\right) + u^2_k J_{mms,k} \end{aligned} \tag{23}$$

These inertia equations can be expressed as a linear combination of its base parameters, a set of minimum number of combined inertial parameters, precisely given by equation (12),

$$J_{AX} = D(q)\Phi_{min}$$

where $D(q)$ is a matrix of base functions arranged in a $6 \times n^*_{BP}$, Φ_{min} is a vector of base parameters of the form $n^*_{BP} \times 1$, and n^*_{BP} are the minimum number of base parameters needed to govern the dynamics of the manipulator. As aforementioned, J_{AX} are only dependent on the configuration of the IR, i.e., q. This will form the basis of conditioning, later described the following chapter.

4.3.1 Algorithm: Load-side Inertia

The second algorithm presented is called *Inertia Transform* that will help achieve the first objective of this thesis (i.e., symbolical modeling of the load-side Inertia).

Algorithm 4.2 Load-side Inertia (inertiaTransform)

Inputs: $q, j, l, m_l, I_l, I_m, g_R$
Outputs: totalInertia

Set $totalInertia = 0, \quad jTensor = 0, \quad jSteiner = 0, \quad JMotor = 0$
if $1 \leq j \leq 6$ **then**
  $jMotor \leftarrow g_R(j)^2 * I_m(j)$
  $linkList = j : 1 : 6$
  for i = linkList **do**
    $R \leftarrow kinematicTransform(q,\ j,\ i,$ 'joint', 'rotation'$)$
    $v \leftarrow kinematicTransform(q,\ j,\ i,$ 'cog', 'translation'$)$
    $jTensor \leftarrow jTensor + R(j)' * (R * I_l(i) * R') * R(j)$
    $skewMat \leftarrow vec2skew(v)$ ▷ A simple function that yields a skew symmetric matrix
    $jSteiner \leftarrow jSteiner + R(j)'(-m_l(i) * skewMat * skewMat) * R(j)$
  end for
  $totalInertia = jTensor + jSteiner + jMotor$
end if
  return totalInertia

The inputs are as follows: $q = [q_1, q_2, q_3, q_4, q_5, q_6]$ is the configuration of the IR; $i \in \{1, \ldots, 6\}$ is the joint at which inertia is to be found; $l \in \{j, \ldots, 6\}$ is a particular link if the inertia of that particular link is only needed; m_l is a vector of all link masses; I_l is an array of all link inertia tensors; I_m is a vector of all motor inertia; g_R is a vector of gear ratios at each axis. The output is the total inertia at the joint j, considering whether the total load-side inertia at that joint (j) or of just a link (denoted by l) is of interest. For the sake of simplicity, a shorter argument list for *kinematicTransform* function is used to derive the total inertia. The arguments, $\tilde{r}^i_{cog_i,o_i}$, and *transformSet* can be computed and used in that function. After modelling the inertia using above algorithms 4.1 and 4.2, in order to achieve the first sub-objective, a further task is to reduce the minimal model to get equation (12). Total Inertia equations for axis 5 and 6 can be derived by hand for the validation of symbolic calculations using any software of choice. Here, MATLAB's symbolic toolbox was employed to calculate the symbolic form of the inertia equations (23) and further validated. Thus, the next section is wholly dedicated to introducing

a set of algorithms to derive the analytical model (equation (12) symbolically.

4.4 Direct Calculation of a Minimal Model

Similar to equation (12), firstly the symbolic expressions of load-side inertia for each axis using the *inertiaTransform* algorithm 4.2 implemented using the Symbolic Math Toolbox of MATLAB needs to be transformed into an axiswise linear system,

$$J_{AX_k} = d_k(q)\phi_k, \tag{24}$$

where for each axis k, $d_k(q)$ and ϕ_k have different dimensionality. The vectors $d_k(q)$ and ϕ_k are the base functions and base parameters for that particular axis k. For illustration, consider the following symbolic expression of load-side inertia for axis 5.

$$\begin{aligned}
J_{AX_5} =& I_{5_{zz}} + 0.01m_6 cos^2(q_5) + 0.01m_6 sin^2(q_5) + J_{mot_5} i^2_{gear_5} + I_{6_{yy}} cos^2(q_6) + I_{6_{xx}} sin^2(q_6) \\
&+ 2I_{6_{xy}} cos(q_6) sin(q_6) - 0.2m_6 r_{6_y} cos^2(q_5) - 0.2m_6 r_{6_y} sin^2(q_5) + m_5 r^2_{5_x} cos^2(q_5) \\
&+ m_5 r^2_{5_z} cos^2(q_5) + m_6 r^2_{6_y} cos^2(q_5) + m_5 r^2_{5_x} sin^2(q_5) + m_5 r^2_{5_z} sin^2(q_5) + m_6 r^2_{6_y} sin^2(q_5) \\
&+ m_6 r^2_{6_x} cos^2(q_5) cos^2(q_6) + m_6 r^2_{6_x} cos^2(q_6) sin^2(q_5) + m_6 r^2_{6_z} cos^2(q_5) sin^2(q_6) \\
&+ m_6 r^2_{6_z} sin^2(q_5) sin^2(q_6) - 2m_6 r_{6_x} r_{6_z} cos^2(q_5) cos(q_6) sin(q_6) \\
&- 2m_6 r_{6_x} r_{6_z} cos(q_6) sin^2(q_5) sin(q_6).
\end{aligned}$$

All the terms that are dependent on the configuration of the IR, i.e., q, are always in some combination of $\sin$ and $\cos$, for instance $\sin^m(q_i)\cos^n(q_i)$. Thus, simplifying and combining all the terms bearing the same trigonometric functions, will give us the base functions and the corresponding base parameters as follows.

$$\begin{aligned}
J_{AX_5} =& I_{5_{zz}} + I_{6_{xx}} + 0.01m_6 - 0.2m_6 r_{6_y} + J_{mot_5} i^2_{gear_5} + m_5 r^2_{5_x} + m_5 r^2_{5_z} + m_6 r^2_{6_y} + m_6 r^2_{6_z} \\
&+ (-I_{6_{xx}} + I_{6_{yy}} + m_6 r^2_{6_x} - m_6 r^2_{6_z}) cos^2(q_6) + (2I_{6_{xy}} - 2m_6 r_{6_x} r_{6_z}) cos(q_6) sin(q_6)
\end{aligned}$$

It can be easily rewritten as a matrix equation (24), which results in $d_5(q)$ and ϕ_5 as follows.

$$d_5(q) = \begin{bmatrix} cos(q_6)sin(q_6) & cos^2(q_6) & 1 \end{bmatrix}$$

$$\phi_5 = \begin{bmatrix} 2I_{6_{xy}} - 2m_6 r_{6_x} r_{6_z} \\ -I_{6_{xx}} + I_{6_{yy}} + m_6 r^2_{6_x} - m_6 r^2_{6_z} \\ I_{5_{zz}} + I_{6_{xx}} + 0.01m_6 - 0.2m_6 r_{6_y} + m_5 r^2_{5_x} + m_5 r^2_{5_z} + m_6 r^2_{6_y} + m_6 r^2_{6_z} + J_{mot_5} i^2_{gear_5} \end{bmatrix}$$

4.4.1 Algorithm: Axiswise Linear Systems

A basic algorithm that collects the unique base functions would be beneficial to build the linear model (24) and it is summarized in the algorithm 4.3. The input for the following algorithm are: q (configuration of the IR), and the inertia (at a particular axis) which is the output of the *inertia-*

Transform algorithm 4.2. In algorithm 4.3, first the inertia is simplified using basic trigonometric equations. Next, all the terms common to the unique base functions are collected into brackets. Further, this form of inertia equation is split at the +/- operators with their respective +/- signs preserved. Then lastly, going through each of these split terms, the base function and the base parameters are factored out and converted to the matrix equation (24).

Algorithm 4.3 Axiswise Linear Systems (axisLinSys)

Inputs: q, inertia
Outputs: baseFunctions, baseParamters

$inertiaEq \leftarrow simplify(inertia)$ ▷ Simplify using trigonometric equations
$inertiaEq \leftarrow collect(inertiaEq, sin, cos)$ ▷ Collect the common terms
$splitInertiaEq \leftarrow split(inertiaEq)$ ▷ Splits equation at +/- operators
for i = 1 : length($splitInertiaEq$) **do**
    $f(1), f(2) \leftarrow factor(splitInertiaEq(i), q)$ ▷ Splits the base functions and base parameters
    $baseFunctions(i) \leftarrow f(1)$
    $baseParameters(i) \leftarrow f(2)$
end for
    return baseFunctions, baseParameters

4.4.2 Algorithm: Eliminate Repeating Base Parameter

The results of algorithm 4.3 may have repeating base parameters that correspond to different base functions in each axis. These base functions associated with the repeating base parameters needs to be added and the repeating base parameters needs to be eliminated leaving a single instance of it. This operation still preserves the inertia equation for each axis. The algorithm 4.4 helps us reduce it to a succinct form of equation (24) which is devoid of any repeating base parameters.

Algorithm 4.4 Eliminate repeating Base Parameter (eliminateRepeatBaseParam)

Inputs: baseFunctions, baseParameters
Outputs: newBaseFunctions, uniqueBaseParameters

$newBaseFunctions \leftarrow []$
$uniqueBaseParameters \leftarrow unique(baseParameters)$
for i = 1 : length($uniqueBaseParameters$) **do**
    Compare $uniqueBaseParamters(i)$ with all $baseParameters$
    Find the indices j of $baseParameters$ which are equal to $uniqueBaseParameters(i)$
    Add correponding $baseFunctions(j)$ together and append to $newBaseFuntions$.
end for
    return newBaseFunctions, uniqueBaseParameters

The algorithm 4.4 first takes the vector of base parameters and finds a unique set out of them, which is going to be the new base parameter vector for that particular axis and is stored in a new vector. Second, comparing each term of this unique vector of base parameters in a sequential manner with the old vector of base parameters the indices where they are repeating are found. Then, the base functions corresponding to those indices are added together and appended to the new base function vector at the index of compared unique base parameter.

4.4.3 Algorithm: Pre-Minimal Model

Equation (24) can be combined to form a system of equations of all axes by creating a unique base parameters vector that combines all base parameters from all axes in Φ of the size $n_{BP} \times 1$.

$$J_{AX} = D(q)\Phi \tag{25}$$

Then, inserting the associated base functions at the right index in the matrix $D(q)$ of the size $6 \times n_{BP}$ yields equation (25).

Algorithm 4.5 Pre-Minimal Model (preMinimalModel)

Inputs: baseFunctions$_{axis}$, baseParameters$_{axis}$, where $axis = \{1, \ldots, 6\}$
Outputs: baseFunctionsMatrix, uniqueBaseParameters

$uniqueBaseParameters \leftarrow unique(baseParameters_{axis \in \{1,\ldots,6\}})$
$n_{BP} \leftarrow length(uniqueBaseParameters)$
$baseFunctionsMatrix \leftarrow []_{6 \times n_{BP}}$
for $axis$ = 1 : 6 **do**
    for i = 1 : length($baseParameters_{axis}$) **do**
        Compare $baseParameters_{axis}(i)$ with $uniqueBaseParameters$
        Find the index j of $uniqueBaseParameters$ equal to $baseParameters_{axis}(i)$
        $baseFunctionsMatrix(axis, j) \leftarrow baseFunctions_{axis}(i)$
    end for
end for
    return baseFunctionsMatrix, uniqueBaseParameters

This can be achieved using the algorithm 4.5, that takes in the output of the algorithm 4.4 as its inputs (i.e., baseFunctions$_{axis}$ and baseParameters$_{axis}$), and outputs the base function matrix and the unique base parameter vector.

4.4.4 Algorithm: Calculate Minimal Model

The reason for developing the previous and the following algorithms can be summarized in the Figure 4.3 Thus, three more algorithms: two helper algorithms called *findLinDep* and

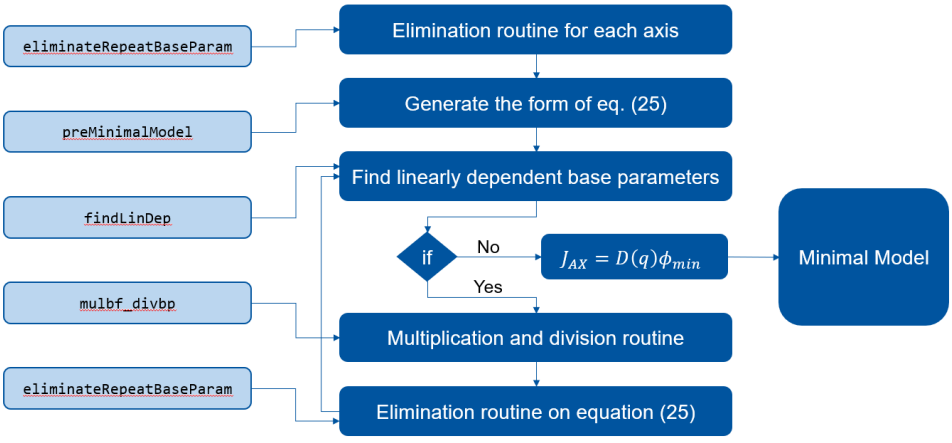

Figure 4.3: Direct Calculation of a Minimal Model

mulbf_divbp, and the final one called *calculateMinimalModel* completes the first sub-objective

of this thesis. The first helper algorithm 4.6 called as *findLinDep* takes in an input of base parameters vector (i.e., the output vector of unique base parameters from algorithm 4.5). It takes each element from this vector and divides every other element and checks whether there is a numerical factor. If there exists a numerical factor, then a linear dependency is found between those elements and their indices are appended to the dependencies matrix of the size $n_D \times 2$.

Algorithm 4.6 Find Linearly Dependent Base Parameters (findLinDep)

Inputs: baseParametersVec
Outputs: dependencies

$dependencies \leftarrow []$
$n_{BP} \leftarrow length(baseParametersVec)$
for i = 1 : n_{BP} **do**
  for j = i+1 : n_{BP} **do**
    $factor \leftarrow simplify(baseParametersVec(j)\ /\ baseParametersVec(i))$
    if $isnumeric(factor)$ and $j \notin dependencies(:, 2)$ **then**
      append(dependencies, $[i, j]$)
    end if
  end for
end for
  return dependencies

The second one, algorithm 4.7 (*mulbf_divbp*) first takes those numerical factors of the linear dependencies found using the algorithm 4.6 (i.e., *dependencies*) as the input. Next, it uses those numerical factors to normalize the base parameters (dividing the dependent base parameter indexed in the second column of dependencies matrix by the numerical factor) and the corresponding base functions (multiply the base function corresponding to those normalized base parameters by the numerical factor). The idea of doing this is to get to a form of repeating base parameters which can be eliminated and the corresponding base functions can be added by reusing the algorithm 4.4. Thus, a new reduced form of this unique base parameter vector is obtained. And again, linear dependencies in this newly reduced base parameter vector are found using the algorithm 4.6 and this loop is repeated until no linear dependencies are found as depicted in the Figure 4.3.

Algorithm 4.7 Coefficient multiplication and division (mulbf_divbp)

Inputs: baseFunctionsMatrix, baseParametersVec, dependencies
Outputs: normedbaseFunctionsMatrix, normedbaseParametersVec

$n \leftarrow rows(dependencies)$
$normedbaseFunctionsMatrix \leftarrow baseFunctionsMatrix$
$normedbaseParametersVec \leftarrow baseParametersVec$
for i = 1 : n **do**
  $j \leftarrow dependencies(i, 1)$
  $k \leftarrow dependencies(i, 2)$
  $factor \leftarrow simplify(baseParametersVec(k)\ /\ baseParametersVec(j))$
  $normedbaseFunctionsMatrix(:, i) \leftarrow baseFunctionsMatrix(:, i) * factor$
  $normedbaseParametersVec(i) \leftarrow baseParametersVec(i)/factor$
end for
  return normedbaseFunctionsMatrix, normedbaseParametersVec

Final part of this section is the main algorithm 4.8 which combines all of the above algorithms in the workflow described in the Figure 4.3. The inputs needed are the base functions and base parameters of each axis. First, it uses algorithm 4.4 to reduce the dimensionality of the base parameters vectors axiswise. Next, it uses algorithm 4.5 to convert the axiswise linear systems into a combined linear system (as per equation (25)). Finally, a while loop executes algorithms 4.6, 4.7, and 4.4 until no dependencies in the base parameters are found. That's where the loop exits and the desired linear system as per equation (12) and the first sub-objective of the thesis is accomplished.

Algorithm 4.8 Calculate Minimal Model (calcMinimalModel)

Inputs: bf$_{axis}$, bp$_{axis}$
Outputs: bfMatmin, bpmin

for axis = 1 : 6 **do**
  $bf_{axis}, bp_{axis} \leftarrow eliminateRepeatBaseParam(bf_{axis}, bp_{axis})$
end for
$bfMatmin, bpmin \leftarrow preMinimalModel(bf_{:}, bp_{:})$
$dependencies \leftarrow findLinDep(bpmin)$
while $dependencies \neq []$ **do**
  $bfMatmin, bpmin \leftarrow mulbf\_divbp(bfMatmin, bpmin, dependencies)$
  $bfMatmin, bpmin \leftarrow eliminateRepeatBaseParam(bfMatmin, bpmin)$
  $dependencies \leftarrow findLinDep(bpmin)$
end while
  return bfMatmin, bpmin

All the above algorithms were implemented in MATLAB. The resulting linear model in comparison to equation (12) has the base function matrix $D(q)$ of dimesionality 6×32, and the base parameter vector Φ_{min} of dimensionality 32×1 (explained in the chapter 6). This resulting linear model will be the basis of further identification routine, especially the base function matrix will be first used to perform conditioning of the regressor matrix, and further those set of configurations that make up the regressor matrix will be used for designing exciting trajectories. But, this thesis is going to cover till the part of finding the set of configurations that yield a well-conditioned regressor matrix.

5 Condition Number Optimization

In contrast to the second common feature of the identification process discussed in section 2.3, since the regressor matrix of the linear model (equation (12)) is only dependent on the configuration of the IR, the problem boils down to finding configurations of IR that ensure a good level of excitation. The following questions are going to be the focus of this chapter and an attempt to accomplish the second and third sub-objectives of this research project thesis.

1. What criteria to use to ensure the level of excitation for each parameters ?
2. How many equations to consider ?

From most linear algebra literature [STRA06], the best criterion associated with estimation of a vector x using linear model $A\mathrm{x} = b$, is the condition number. It gives a bound on how inaccurate the solution x will be after approximation. Note that this is before the effects of round-off error are taken into account; conditioning is a property of the matrix, not the algorithm or floating-point accuracy of the computer used to solve the corresponding system. In particular, the condition number is (very roughly) the rate at which the solution x will change with respect to a change in b,

$$\frac{\|\delta x\|}{\|x\|} = \frac{\lambda_{max}}{\lambda_{min}} \frac{\|\delta b\|}{\|b\|} \tag{26}$$

where the ratio $\lambda_{max}/\lambda_{min}$ is the condition number, and λ_{max} and λ_{min} are the largest and the smallest singular values of the regressor matrix A respectively. Thus, if the condition number is large, even a small error in b may cause a large error in x. On the other hand, if the condition number is small, then the error in x will not be much bigger than the error in b. The condition number is defined more precisely to be the maximum ratio of the relative error in x to the relative error in b. In hindsight, a problem with a low condition number is said to be well-conditioned, while a problem with a high condition number is said to be ill-conditioned. Thus, the criterion used here is called the 2-norm condition number, which depends on the largest and smallest singular values of the regressor matrix $W(\mathbf{q})$ of equation (13).

Axis	Number of Base Parameters
1	27
2	19
3	17
4	7
5	3
6	1

Table 5.1: Number of Base Parameters

The number of equations depends upon the number of base parameters at each axis. From equation (24) for axis 5 there are three base parameters that have three corresponding base functions. Thus, three equations are sufficient for axis 5. This results in three configurations that are plugged into the base function vector of axis 5, which corresponds to the fifth row of the

base function matrix, to get three rows of the regressor matrix. The following table summarizes the number of base parameters calculated for each axis for the *MABI Max 100* IR using the algorithm 4.8 (code is given in the chapter 6). Thus, in total 74 equations, corresponding to 74 configurations is sufficient for estimating the 32 base parameters. Therefore, the problem boils down to finding 74 configurations, and the simplest approach to find them is to go axis-wise. Let's examine the base functions of axis 5. They were computed analytically in section 4.4 as follows,

$$d_5(q) = \begin{bmatrix} cos(q_6)sin(q_6) & cos^2(q_6) & 1 \end{bmatrix}$$

Plotting them (see Figure 5.1) within the range of q_6 (given in the product data sheet of MABI MAX 100), a set of minimum three configurations, $(q_{6,1}, q_{6,2}, q_{6,3})$ is needed that when substituted in the above base function vector should yield a matrix having the lowest condition number. Axis 5 is much easier to visualize because it has base functions dependent on just

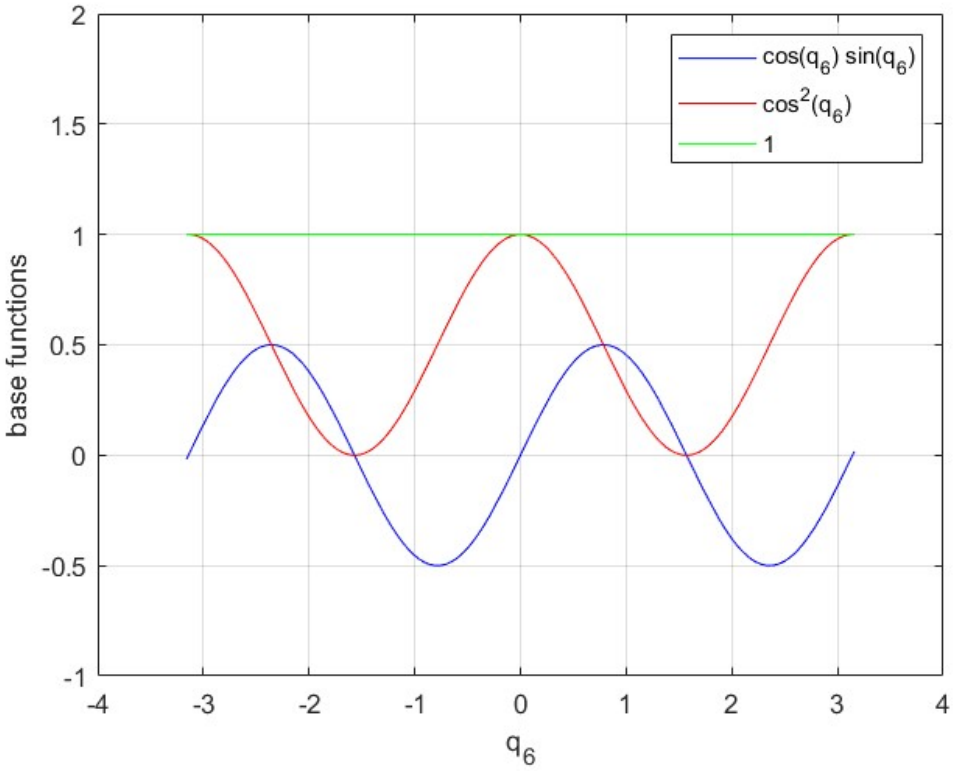

Figure 5.1: Base functions for axis 5 in the range -181°to 181°

one variable, i.e., q_6. Heuristically, selecting the set of three configurations, $(q_{6,1}, q_{6,2}, q_{6,3})$, is enough such that it is close to an identity matrix I. But, since there is always have a '1' in the last column of the regressor matrix, it would be enough to find configurations that yield a matrix close to any one of the following,

$$I_{mod} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{27}$$

This above matrix I_{mod} gives a 2-norm condition number of 3.7321, which is good enough but if all of them are combined by taking all the unique rows and concatenating them, a much better conditioning of 1.5811 is achieved.

$$I_{mod} = \begin{bmatrix} 1 & 0 & 1 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{28}$$

These matrices in equations (27) and (28) are ideal cases; and will yield a configuration set of at the most five configurations, i.e, $(q_{6,1}, q_{6,2}, q_{6,3}, q_{6,4}, q_{6,5})$, two for first and second base parameter (the second one for compensation), and one for the last; or at least three configurations, $(q_{6,1}, q_{6,2}, q_{6,3})$, one for each base parameter. Thus, the number of equations lies in the range of 1 to 2 times the number of base parameters at each axis. These are ideal cases because from Figure 5.1, for the given domain of q_6, i.e., $-181°$ to $181°$, which is constrained by the robot mechanism, the domain of the base functions is different. For instance, the domain of the base function $\cos(q_6)\sin(q_6)$ lies between -0.5 to 0.5. Thus, it is not possible to achieve any of those ideal matrices if the domain of the base functions does not allow it to go to any of the ideal rows of the above ideal matrices.

On the contrary, a closer look at Figure 5.1, gives more insights for the base function $\cos^2(q_6)$. It can reach a value of '1' but not '-1'. In particular, at angles $-180°$, $0°$, and $180°$, it is '1' and the other base function $\cos(q_6)\sin(q_6)$, is '0'. Thus the ideal row $[0, 1, 1]$ for that base function can be achieved. Next, for compensating for the base function $\cos(q_6)\sin(q_6)$, again from observations, the ranges of ($-135°$ to $-45°$) and ($45°$ to $135°$) suits better for further optimization, since in these ranges the base function $\cos(q_6)\sin(q_6)$ is more closer to '1' or '-1' than the base function $\cos^2(q_6)$. Thus, I_{mod} needs to be further modified taking into account the maximum and minimum values of each base parameters. This yields the following,

$$I_{mod} = \begin{bmatrix} 0.5 & 0 & 1 \\ -0.5 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{29}$$

This analysis was easier for axis 5, since the base functions are univariate functions of q_6. Applying the same strategy to axis 4, adds another dimension, since all the base functions for axis 4, would be multivariate functions of q_5 and q_6. Thus, lowering the axis would indeed become a very hard optimization problem to find the desired combined ranges of the independent variables (see Table 5.2). Thus, stochastic methods to find the ranges of the base parameters at each axis and to get the desired target matrices like the one for axis 5 (equation (29)) would

be best suited. With this analysis, depending upon the target matrix, utilizing condition number as the criterion to ensure the level of excitation of each parameter, and the number of rows in the regressor matrix needed to ensure well-conditioning of the linear system of equations are well-reasoned.

Axis	Independent Variables
1	q_1, q_3, q_4, q_5, q_6
2	q_3, q_4, q_5, q_6
3	q_4, q_5, q_6
4	q_5, q_6
5	q_6
6	-

Table 5.2: Independent Variables in Base Functions

5.1 Algorithm: Generate Poses

To determine the range of each base function at each axis, the joint space must be discretized and each base function evaluated. The discretization captures the range of each joint and, as a result, the range of the base functions. As a result, an algorithm that aids in the creation of an array of this discretization would be advantageous. Furthermore, finding the maximum and minimum values of base functions requires a randomly generated dataset that captures them. Randomly generated datasets are especially useful when discretizing the joint space for axis 3 and below, owing to the *curse of dimensionality*. The common theme of this problem is that as dimensionality increases, the volume of space increases so quickly that discretization results in an exponential blowup (see Figure 5.2).

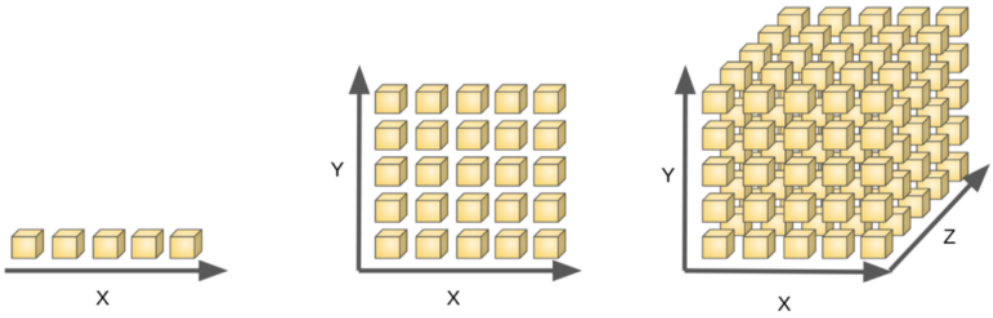

Figure 5.2: Curse of Dimensionality

For illustration, the base functions at axis 1 are multivariate functions of $(q_2, q_3, q_4, q_5, q_6)$. Therefore, the uniform discretization of N intervals for all variables yields the discretized space that has N^5 points. Thus, as N grows larger, the discretization results in an exponential blowup. The following algorithm 5.1 dicretizes the independent variables (as per Table 5.2) at a particular axis either uniform, weighted, or random. The inputs for the algorithm are as follows: $axis$

where you want to perform the discretization; $refPose$ is the configuration of the IR that ensures full range of all the subsequent axes after the $axis$; q_{lb} is the vector of lower bound of all joints; q_{ub} is the vector of upper bound of all joints; $mode$ is the argument that helps to discretize the range of the independent variables either *uniform*, *weighted*, or *random*; N is the number of discretization points. If $mode$ = '*uniform*', all the independent variables are uniformly discretized at N points within their joint limits and all possible combinations of the individual discretization is the output. If $mode$ = '*random*', a random sample is generated within the joint limits until there are N samples. And if $mode$ = '*weighted*', N needs to be a vector argument (for e.g., $axis = 4$ has two independent variables q_5 and q_6, thus $N = [n_5, n_6]$. Further, q_5 and q_6 are discretized at n_5 and n_6 points respectively, and all possible combinations of this individual discretization is the output). Thus, this algorithm gives a liberty to choose which type of discretization is needed (see algorithm 5.1).

Algorithm 5.1 Generate Poses(generatePoses)

Inputs: axis, refPose, q_{lb}, q_{ub}, mode, N
Outputs: Poses, nPoses

Poses = []
if mode = 'random' **then**
  for i = 1:axis **do**
    $q_{lb}(i) = refPose(i)$
    $q_{ub}(i) = q_{lb}(i)$
  end for
  while size(poses,2) < N **do**
    q = zeros(1,6)
    for j = 1:6 **do**
      $q(j) = (q_{ub}(j) - q_{lb}(j)) * rand() + q_{lb}(j)$
    end for
  end while
  nPoses = N
else
  if mode = 'uniform' **then**
    for k = axis+1:6 **do**
      $q_k = linspace(q_{lb}(k), q_{ub}(k), N)$
    end for
  else
    if size(N,2) == size(axis+1:6) **then**
      for k = axis+1:6 **do**
        $q_k = linspace(q_{lb}(k), q_{ub}(k), N(k))$
      end for
    end if
  end if
  Poses $\longleftarrow$ Combinations of q_k and concatenate with refPose(1:axis)
  nPoses = size(Poses,1)
end if
  return Poses, nPoses

5.2 Algorithm: Generate Target Matrix

Algorithm 5.1 ensures that the first constraint of the optimization problem (14) from section 2.4 is met. But, the generated poses should not collide with the environment of the working cell as well with itself, i.e., they must satisfy the second and third constraints of the optimization problem (14). Removing the poses that are self-colliding and colliding with the working cell, the

Algorithm 5.2 Generate Target Matrix (generateTargetMatrix)

Inputs: axis, baseFunctionsMat, refPose, q_{lb}, q_{ub}, mode, N
Outputs: identMod, nonCollPoses, bfVals

idx = baseFunctionsMat(axis,:) $\neq$ 0
bfs = baseFunctionsMat(axis,idx)
nBP = length(bfs)
for i = 1:nBP **do** ▷ Create function handles of base functions
    bfsFuntionHandles(i) = @($q_1, q_2, q_3, q_4, q_5, q_6$) bfs(i)
end for
if axis = 6 **then**
    poses = [refPose]
else
    poses = generatePoses(axis, refPose, q_{lb}, q_{ub}, mode, N)
end if
status = checkCollisionStatus(poses) ▷ Checking for collisions (internal and external)
nonCollPoses = pose(status == 0, :)
bfVals = zeros(size(nonCollPoses,1), nBP)
for j = 1:size(nonCollPoses,1) **do** ▷ Substitute non-colliding poses in the function handles
    q = nonCollPoses(j,:)
    for k = 1:nBP **do**
        bfVals(j,k) = bfsFuntionHandles(k)($q_1, q_2, q_3, q_4, q_5, q_6$)
    end for
end for
bfVals$_{max}$ = max(bfVals) ▷ A vector of max values of all columns in bfVals
bfVals$_{min}$ = min(bfVals) ▷ A vector of min values of all columns in bfVals
ident = I$_{nBP}$
identMod = repeatRows(ident,2)
for m = 1:nBP **do** ▷ Multiply the max and min values
    identMod(2*m-1,:) = round(bfVals(m),1)*identMod(2*m-1,:)
    identMod(2*m,:) = round(bfVals(m),1)*identMod(2*m,:)
end for
identMod(:,end) = 1 ▷ From here remove the repeating last row
identMod(end,:) = []
vec = zeros(1,size(identMod,2))
vec(end) = 1
isMem = ismember(identMod, vec, 'rows')
idxVec = find(isMem == 1);
identMod(idxVec(1:end-1),:) = []
return identMod, nonCollPoses,bfVals

constrained optimization problem (14) is converted to an unconstrained one as follows.

$$\min_{\mathbf{q}} \quad cond(W(\mathbf{q})) \tag{30}$$

In the following algorithm 5.2, the above aforementioned second and third constraints must be satisfied to filter out the non-colliding poses and then find the desired target matrix. Further these non-colliding poses will be used in algorithm 5.3 (Condition Number Optimization). The inputs to algorithm 5.2 are same as the algorithm 5.1 except for *baseFunctionsMat* which is same as the output *bfMatmin* of the algorithm 4.8.

Algorithm 5.2 first creates the function handles of the base functions and a dataset of configurations (using algorithm 5.1) for that particular *axis* as per the input $mode$ (e.g., *random*). Then, using any simulation techniques and developing a function to check the collision status (for both internal or external), the ones that are non-colliding must be filtered out. Next, these non-colliding poses in the function handles of the base functions are substituted and the maximum and the minimum values of each base functions are found. These values are used to manipulate the identity matrix with simple methods as described in the later part of the algorithm, to get the desired outputs *identMod* (target matrix), *nonCollPoses* (non-colliding poses), and *bfVals* (array of base functions).

5.3 Algorithm: Condition Number Optimization

The final algorithm 5.3 uses the output of the above explained algorithm 5.2, i.e., *identMod*, *bfVals*, and *nonCollPoses*, alongwith the inputs *axis*, and *penInt* (used to discretize the variable *penalty*). Since the base functions are highly nonlinear, a numerical optimization technique based on the already evaluated base functions from the algorithm 5.2 is adopted. The following is an explanation of the cascaded optimization scheme of finding the best candidate among the evaluated base functions array (*bfVals*) using two simple metrics, *dot product*, and *distance*. First, the sum of all the elements of j^{th} row from the target matrix (*identMod)* is subtracted from the dot product between the whole array of *bfVals* and j^{th} row from the target matrix as follows. This is the first metric where the dot product captures the sum of the values corresponding to the non-zero values of the j^{th} row from the target matrix (*identMod)* and checks how far are they from the sum of the j^{th} row. If the values are close to the j^{th} row of the target matrix then the metric A goes to zero and if not then we get te residual error.

$$A = bfVals \cdot identMod(j,:)' - sum(identMod(j,:)) \tag{31}$$

Next, the second metric is simply the distance of each row in *bfVals* from the j^{th} row in the target matrix, as given below.

$$B = \|bfVals - identMod(j,:)\| \tag{32}$$

The above two metrics of rating are combined together with a *penalty* term (that lies between 0 and 1) as follows. The reason of combining in the following manner is to maximize A (meaning the dot product is to be maximized in order to be as close to the j^{th} row of the target matrix), and minimize B in order to minimize the distance between them. The penalty term penalizes A indirectly and forces A to go to as close as possible to 0. Thus, the best rated candidate is the

Algorithm 5.3 Condition Number Optimization (condNumOpt)

Inputs: axis, identMod, bfVals, nonCollPoses, penInt
Outputs: minCond, obsMat, configMat

```
penalty = linspace(0,1,penInt)
m = length(penalty)
n = size(identMod,1)
nBP = size(identMod,2)
condArr = zeros(size(penalty))
for i = 1:m do
    obsMatTemp = zeros(n,nBP)
    for j = 1:n do
        A = bfVals · identMod(j,:)' - sum(identMod(j,:))
        B = || bfVals - identMod(j,:) ||
        C = A - penalty(i) × B
        CmaxIdx = argmax(C)
        obsMatTemp(j,:) = bfVals(CmaxIdx,:)
    end for
    condArr(i) = cond(obsMatTemp)
end for
minCond, minCondIdx = min(condArr), argmin(condArr)
penalty_optim = penalty(minCondIdx)
mobsMat = zeros(n,nBP)
configMat = zeros(n,6)
for k = 1:n do
    A = bfVals · identMod(k,:)' - sum(identMod(k,:))
    B = || bfVals - identMod(j,:) ||
    C = A - penalty_optim × B
    CmaxIdx = argmax(C)
    obsMat(k,:) = bfVals(CmaxIdx,:)
    configMat(k,:) = nonCollPoses(CmaxIdx,:) ×180/π
end for
    return minCond, obsMat, configMat
```

$\arg\max$ of the column vector C (from equation (33)).

$$C = A - penalty \times B \tag{33}$$

Thus, looping over each row of the target matrix, and evaluating C using equations (31), (32), and (33), the best rated candidates from *bfVals* are populated in the regressor matrix *obsMatTemp* with a certain penalty. Therefore, the further optimization problem boils down to finding this optimal penalty. Hence, another overall loop is cascaded which takes different penalty values discretized between 0, and 1 (equation (34)).

$$C = A - penalty(i) \times B \qquad \forall i \in \{0, \ldots, \textit{penInt}\} \tag{34}$$

Collecting the corresponding condition number of the resulting *obsMatTemp* for each penalty in the array *condArr* and taking the $\arg\min$ of the *condArr* results in the optimal penalty.

$$penalty_{optim} = penalty(\arg\min condArr) \tag{35}$$

Finally, again using equations (31), (32), and (33) with $penalty = penalty_{optim}$, and finding the $\arg\max C$, the final objective of finding the optimal configurations and the corresponding regressor matrix is achieved as follows:

$$\begin{aligned} A &= bfVals \cdot identMod(j,:)' - sum(identMod(j,:)) \\ B &= \|bfVals - identMod(j,:)\| \\ C &= A - penalty_{optim} \times B \\ C_k &= \arg\max(C) \\ obsMat(j,:) &= bfVals(C_k,:) \\ configMat(j,:) &= nonCollPoses(C_k,:) \\ &\forall j \in \{1,\ldots,size(identMod,1)\} \end{aligned} \tag{36}$$

The above explanation is neatly summarized in algorithm 5.3 which is to be performed for each axis since the target matrix (*identMod*) for each axis is different. The overall implementation of the condition number optimization routine i.e., all the algorithms 5.1. 5.2, and 5.3 is demonstrated in the following chapter 6.

6 Implementation on MABI Max 100

This chapter is an example of how you can implement the above routine in MATLAB. One can implement the methods described in any software language of their choice. The goal of the research was to present the mathematical and algorithmic approaches, to the best possible, with the eventual goal of implementation in software of any choice. Here, an object-oriented method was used to include the methods in the specific class structures as follows.

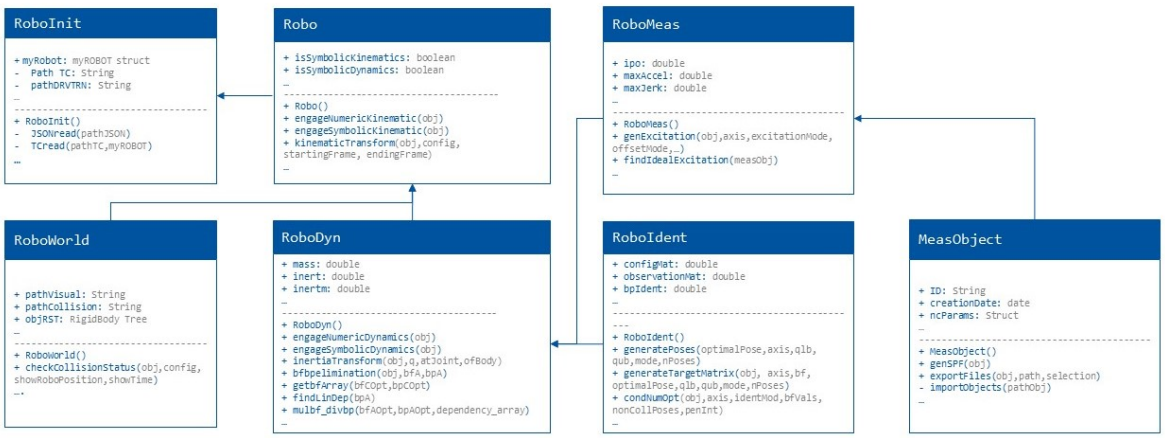

Figure 6.1: Object-oriented Programming in MATLAB (Class Structures)

RoboInit is used to initialize any robot model. *Robo* is used for performing kinematic analysis on the robot model, for instance, *kinematicTransform* can be use to find transformations for a particular configuration of the robot (algorithm 4.1). *RoboWorld* utilizes the *RigidBodyTree* structure of MATLAB and help in visualizing the robot model in its working environment, for instance, it is used for checking collision detection (used in algorithm 5.2) of a certain configuration pof the robot in its working environment. Further, *RoboDyn* is used for performing dynamic analysis on the robot model, for instance, finding the minimal model (equation (12)). Particularly, all the necessary algorithms 4.2– 4.8 are implemented in this class. *RoboIdent* class is used for the final identification of the base parameters, along with doing the condition number routine (algorithms 5.1– 5.3). *RoboMeas* and *MeasObject* are the classes pertaining to generating excitation trajectories and doing basic filtering tasks on the captured data to find the inertia. In Hindsight, *RoboInit* is the parent class. *Robo* is a derived class from *RoboInit*. *RoboWorld* and *RoboDyn* are further derived from *Robo*. *RoboIdent* and *RoboMeas* are derived from *RoboDyn*. And Lastly, *MeasObject* is derived from *RoboMeas*.

This is just a skeleton of the actual implementation summarized in the Figure 6.1. Following algorithms will depict the results of the implementing this class structure and using their functions/methods to accomplish the objectives of this research project thesis.

6.1 Minimal Model Calculation

First the problem is initialized by creating the object of the *RoboDyn* class and set the type of parameters for kinematics and dynamics. i.e., numeric and symbolic respectively. Next, the

Algorithm 6.1 Base Parameter Calculation (BaseParamCalc)

Inputs: axis, identMod, bfVals, nonCollPoses, penInt
Outputs: bpA, bfA

```
r1 = RoboDyn
r1.engageNumericKinematics
r1.engageSymbolicDynamics
q = [q1, q2, q3, q4, q5, q6]
for axis = 1:6 do
    inertia(axis) = r1.inertiaTransform(q,axis)
    [bf(axis),bp(axis)] = r1.getbfbpArray(inertia(axis),axis)
    [bfA(axis),bpA(axis)] = r1.getbpArray(bf(axis),bp(axis))
    [bfA(axis),bpA(axis)] = r1.bfbpelimination(bfA(axis),bpA(axis))
end for
[bfA,bpA] = r1.getbfArray(bfA,bpA)
dependency_cell = r1.findLinDep(bpA)
while dependency_cell ≠ [] do
    [bfA,bpA] = mulbf_divbp(bfA, bpA, dependencies)
    [bfA,bpA] = eliminateRepeatBaseParam(bfA, bpA)
    dependency_cell = findLinDep(bpA)
end while
    return r1, bfA, bpA
```

inertia (equation (23)) is modeled symbolically. Further, these individually equations are converted into matrix equations (equation (24)) in order to find the base functions and the base parameters for each axis. Next, all of these axiswise equations are combined in a single matrix equation (equation (25)). Finally, we convert it into the required minimal model (equation (12)).

Variable Name	Value
axis	6
bf	1 x 6 cell
bfA	6 x 32 sym
bp	1 x 6 cell
bpA	32 x 1 sym
dependency_cell	[]
inertia	1 x 6 cell
q	1 x 6 sym
r1	1 x 1 RoboDyn

Table 6.1: Base Parameter Calculation Results

Running the above pseudo code with appropriate adjusts gives the following workspace vari-

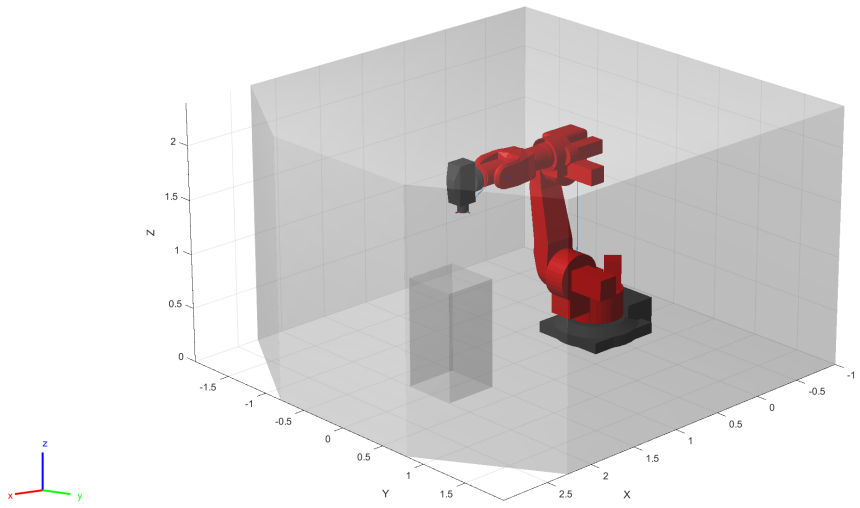

Figure 6.2: Home Position of the Robot in the Working Cell

ables (Table 6.1) and from that we can conclude that our minimal model $J_{AX} = D(q)\Phi_{min}$ consists base functions matrix ($D(q)$) bearing the size 6×32, and minimum number of base parameters (Φ_{min}) bearing the size 32×1. Lastly save the resulting minimal model, especially the variables bpA, bfA, and $r1$ for the following condition number optimization calculation routine.

6.2 Generate Configurations

According to equation (14), the approach taken is to first satisfy the constraints. For that, many configurations are created and checked for self-collisions and collisions with the working cell environment within the joint limits generated using algorithm 5.1. First, we initialize and visu-

Algorithm 6.2 Satisfy Constraints (satisfyConstraints)

Inputs: r1, bpA, bfA, box
Outputs: nonCollPoses, seedPose, r2, r3, qlb, qub

r2 = RoboIdent
r3 = RoboWorld
r3.worldCollisionArrayend+1,1 = box
r3.checkCollisionStatus ([0 ,0 ,0 ,0 ,0 ,0]) ▷ Used for visualizing the home position
qlb := lower bounds of the robot joints
qub := upper bounds of the robot joints
for axis = 1:6 **do** seedPose(axis) := initialize a non-colliding seed pose for each axis
end for
mode = 'random' ▷ can be 'uniform' , 'weighted', or 'random'
nPoses = [50000 ,40000 ,30000 ,20000 ,10000, 2] ▷ Can be changed
for axis = 1:6 **do**
    untestedPoses(axis) = r2.generatePoses(seedPose(axis),axis,qlb,qub,mode,nPoses(axis)) ▷ Algorithm 5.1
    isInCollision = r3.checkCollisionStatus (untestedPosesConc) ▷ User defined function for checking internal and external collisions
    nonCollPoses(axis) = untestedPosesConc(isInCollision(: ,1)==0,:) ▷ isInCollision==0 corresponds to non-colliding poses
end for
    return nonCollPoses, seedPose, r2, r3, qlb, qub

alize the robot (see Figure 6.2) in the working cell environment by adding collision bodies as necessary along with taking the minimal model saved from the previous computation as inputs. Further, the joint limits, *qlb* and *qub*, needs to be specified. Next, a good reference configuration (for example see Figures 6.3 and 6.4) that can ensure generation of as many possible non-colliding configurations for each axis for the robot is needed to be saved in the variable *seedPose*. Further, using these *seedPose*, algorithm 5.1 is invoked to generate *untestedPoses*. Then all these randomly generated configurations are checked for self-collisions and collision with the working cell environment. Lastly save the resulting non colliding configurations, especially the variables $nonCollPoses$, $seedPose$, $r2$, $r3$, qlb, and qub for the following condition number optimization routine.

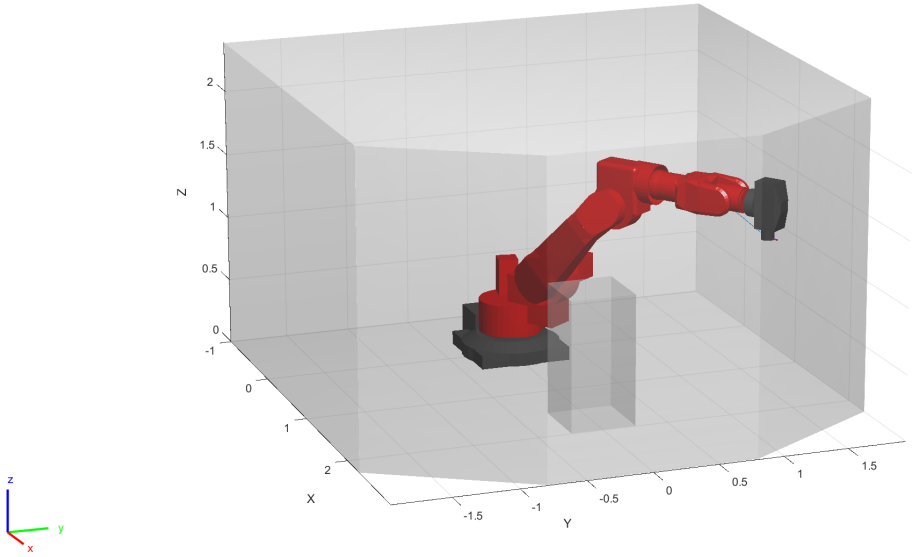

Figure 6.3: First Optimal Pose [35,45,-45,0,0,0]

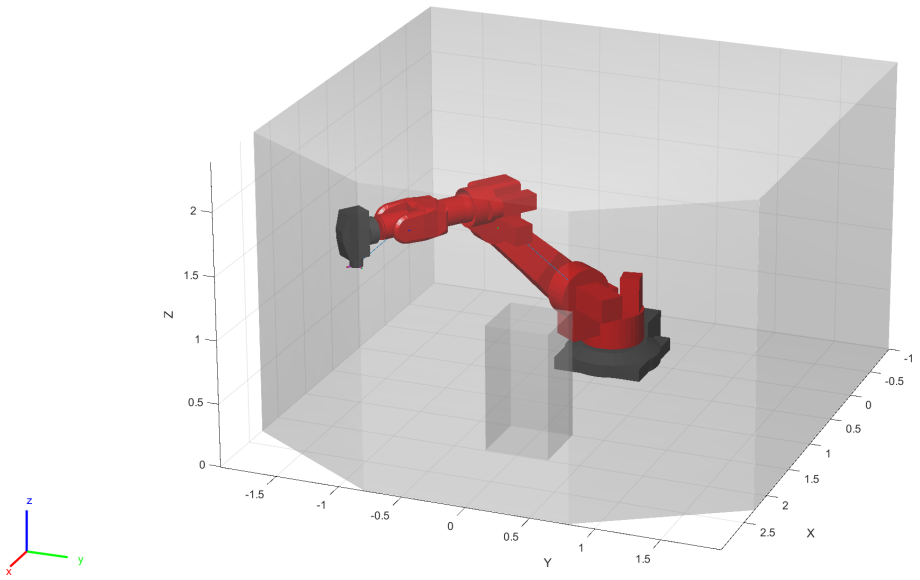

Figure 6.4: Second Optimal Pose [-35,45,-45,0,0,0]

6.3 Condition Number Optimization

First, an initialization by creating an object of *RoboIdent* class along with taking the calculation results of the previous two subsections along with a penalty interval *penInt* as input is needed. Next, the target matrix is computed for all axes as discussed in the section 5.2 with its nec-

Algorithm 6.3 Condition Number Optimization Routine (condNumOptRobot)

Inputs: r1, r2, r3, bfA, bpA, nonCollPoses, seedPose, qlb, qub penInt
Outputs: observationMat, configMat, totalCond, totalCond_init, minCond, obsMat_Cond

```
r2 = RoboIdent
for axis = 1:6 do
    [bfFunVals(axis),ident_mod(axis),ideal_cond_num(axis),obsMat_cond(axis),
q_init(axis), m(axis)] = r2.generateTargetMatrix (axis,bfA,nonCollPoses )
end for
totalCond_init = r2.totalConditionNumber(bfA,q_init)
for axis = 1:6 do
    [observationMat(axis),configMat(axis),minCond(axis),condArr(axis)] =
r2.condNumOpt(ident_mod(axis),bfFunVals(axis),nonCollPoses(axis), penInt)
end forreturn observationMat, configMat, totalCond, totalCond_init, minCond, obs-
Mat_Cond
```

essary outputs for further use. Finally, the condition number is optimized using the following function called *condNumOpt*; it performs the exact steps discussed in section 5.3. The resulting workspace (refer Table 6.2) of the above algorithm 6.3 gives an improved conditioning of 10.0068 from 52.1201.

Variable Name	Value
axis	6
configMat	1 x 6 cell
minCond	[71.0358,19.8997,21.7112,6.7113,3.2863,1]
nonCollPoses	1 x 6 cell
observationMat	1 x 6 cell
obsMat_cond	[56.0025,19.8997,42.9180,12.3127,5.7235,1]
penInt	400
totalCond	10.0068
totalCond_init	52.1201

Table 6.2: Condition Number Optimization Routine Results

7 Summary and Outlook

The literature survey in chapter 2 has demonstrated the progress made in inertial parameter estimation since the mid-1980's. Still, inertial parameter estimation is an ongoing field of research. It is not apparent this approach can be successfully applied to a wide variety of manipulator structures, especially parallel robots. Of the many difficulties confronting the inertial parameter estimation problem, following three can be considered within the scope of further research beyond this thesis. The first difficulty is the target matrix. An identity matrix would be the ideal case to achieve but the base functions do not satisfy the need. Hence, a strategy to include the extremities of the range of base functions and modify the identity matrix was adopted. Perhaps, this target matrix is not the optimal one, since there might be some optimal configurations (that contribute in adding a row in the target matrix) which can be called as compensating poses/configurations that may help in reducing the condition number further. Thus, how to find all those compensating poses that will ultimately answer the question of how many rows are needed in the regressor matrix.

The second difficulty arises through the combination of joint limits and the the joint space is discretize in order to find our initial target matrix and later augment it with a solution to the above first difficulty to include all the compensating poses. If the joint limits constraint the base functions from reaching their actually extremities, then it seems that the base parameters are not actively and fully excited and thus suffers in having ill-conditioning of the regressor matrix. Thus, knowing for sure that the joint limits are not going to help the base functions to reach their extremities then a basic deduction of the extremities found under this constrained domain to build our target matrix would give additional insights on the conditioning of the regressor matrix that can be achieved.

But, this hopefully has set a foundation to present the theory in the most clear yet mathematically rigorous manner for serial manipulators. The third difficulty maybe occur during the estimation step, where the question about the validity of estimated parameters arises. There is justification in why its validity should be questioned. Moreover, inertia equations and the measured inertia through the data gathered using the excitation trajectory can also be subject to non-negligible errors. Thus, given this potentially significant amount of error in the model and in the data, it is not unreasonable to suspect that significant errors in parameter estimates can result. Yes, it is always possible to compute parameter estimates by collecting more data, but there is still a question whether the estimates represent true physical parameters or are rather artifacts of mathematical curve fits. The literature survey has shown that validating estimates properly is not straightforward. The fit between model and data can be excellent, but the parameter estimates themselves can be subjected to a high level of error.

Further, this procedure can also be extended to kinematic calibration. This is possible since the developed algorithm 4.1 can be used to find equations of the TCP in terms of the kinematic parameters from the base frame. Converting that to a linear system just like the inertia equations, and then finding the optimal configurations to populate the regressor matrix, and then using estimation techniques to find these kinematic parameters might be possible. In hindsight, the ideology of the methods discussed in this research can be extended to kinematic parameter estimation as well.

8 References

[ATKE86] Christopher G. Atkeson, Chae H. An, and John M. Hollerbach. “Estimation of Inertial Parameters of Manipulator Loads and Links”. en. In: *The International Journal of Robotics Research* 5.3 (Sept. 1986), pp. 101–119. DOI: 10.1177/027836498600500306. URL: http://journals.sagepub.com/doi/10.1177/027836498600500306 (visited on 05/13/2022).

[BONA05] B. Bona and M. Indri. “Friction Compensation in Robotics: an Overview”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. Seville, Spain: IEEE, 2005, pp. 4360–4367. DOI: 10.1109/CDC.2005.1582848. URL: http://ieeexplore.ieee.org/document/1582848/ (visited on 05/17/2022).

[BONN16] Vincent Bonnet, Philippe Fraisse, Andre Crosnier, Maxime Gautier, Alejandro Gonzalez, and Gentiane Venture. “Optimal Exciting Dance for Identifying Inertial Parameters of an Anthropomorphic Structure”. In: *IEEE Transactions on Robotics* 32.4 (Aug. 2016), pp. 823–836. DOI: 10.1109/TRO.2016.2583062. URL: https://ieeexplore.ieee.org/document/7511777/ (visited on 05/13/2022).

[BOYE98] Frederic Boyer and Wisama Khalil. “An Efficient Calculation of Flexible Manipulator Inverse Dynamics”. In: *The International Journal of Robotics Research* 17.3 (1998), pp. 282–293. DOI: 10.1177/027836499801700305. eprint: https://doi.org/10.1177/027836499801700305. URL: https://doi.org/10.1177/027836499801700305.

[CRAI87] John J. Craig, Ping Hsu, and S. Shankar Sastry. “Adaptive Control of Mechanical Manipulators”. In: *The International Journal of Robotics Research* 6.2 (1987), pp. 16–28. DOI: 10.1177/027836498700600202. eprint: https://doi.org/10.1177/027836498700600202. URL: https://doi.org/10.1177/027836498700600202.

[FEAT14] R. Featherstone. *Rigid Body Dynamics Algorithms*. Springer US, 2014. URL: https://books.google.de/books?id=GJRGBQAAQBAJ.

[FEAT83] R. Featherstone. “The Calculation of Robot Dynamics Using Articulated-Body Inertias”. In: *The International Journal of Robotics Research* 2.1 (1983), pp. 13–30. DOI: 10.1177/027836498300200102. eprint: https://doi.org/10.1177/027836498300200102. URL: https://doi.org/10.1177/027836498300200102.

[GAUT90] M. Gautier and W. Khalil. “Direct calculation of minimum set of inertial parameters of serial robots”. In: *IEEE Transactions on Robotics and Automation* 6.3 (June 1990), pp. 368–373. DOI: 10.1109/70.56655. URL: http://ieeexplore.ieee.org/document/56655/ (visited on 05/13/2022).

[GAUT91] M. Gautier and W. Khalil. “Exciting trajectories for the identification of base inertial parameters of robots”. In: *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*. 1991, 494–499 vol.1. DOI: 10.1109/CDC.1991.261353.

[GAUT97] M. Gautier. “Dynamic identification of robots with power model”. In: *Proceedings of International Conference on Robotics and Automation*. Vol. 3. 1997, 1922–1927 vol.3. DOI: 10.1109/ROBOT.1997.619069.

[GEIS20] Andreas René Geist and Sebastian Trimpe. “Structured learning of rigid-body dynamics: A survey and unified view”. In: *CoRR* abs/2012.06250 (2020). arXiv: 2012.06250. URL: https://arxiv.org/abs/2012.06250.

[GRÜN21] L. Gründel, C. Reiners, L. Lienenlüke, S. Storms, C. Brecher, and D. Bitterolf. “Frequency-Based Identification of the Inertial Parameters of an Industrial Robot”. In: *Production at the leading edge of technology*. Ed. by Bernd-Arno Behrens, Alexander Brosius, Wolfgang Hintze, Steffen Ihlenfeldt, and Jens Peter Wulfsberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 429–438.

[HOLL80] John M. Hollerbach. “A Recursive Lagrangian Formulation of Maniputator Dynamics and a Comparative Study of Dynamics Formulation Complexity”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 10.11 (1980), pp. 730–736. DOI: 10.1109/TSMC.1980.4308393.

[ISO21] ISO. *ISO 8373:2021 Robotics — Vocabulary*. 2021. URL: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-3:v1:en:term:3.6.

[JOVI15] Jovana Jovic, Franck Philipp, Adrien Escande, Ko Ayusawa, Eiichi Yoshida, Abderrahmane Kheddar, and Gentiane Venture. “Identification of dynamics of humanoids: Systematic exciting motion generation”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 2173–2179. DOI: 10.1109/IROS.2015.7353668.

[KAHN71] M. E. Kahn and B. Roth. “The Near-Minimum-Time Control Of Open-Loop Articulated Kinematic Chains”. In: *Journal of Dynamic Systems, Measurement, and Control* 93.3 (Sept. 1971), pp. 164–172. DOI: 10.1115/1.3426492. eprint: https://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/93/3/164/5523696/164_1.pdf. URL: https://doi.org/10.1115/1.3426492.

[KANA84] Takeo Kanade, Pradeep Khosia, and Nobuhiko Tanaka. “Real-time control of CMU direct-drive arm II using customized inverse dynamics”. In: *The 23rd IEEE Conference on Decision and Control*. Las Vegas, Nevada, USA: IEEE, Dec. 1984, pp. 1345–1352. DOI: 10.1109/CDC.1984.272256. URL: http://ieeexplore.ieee.org/document/4048116/ (visited on 05/13/2022).

[KANE80] Thomas R. Kane and David A. Levinson. “Formulation of Equations of Motion for Complex Spacecraft”. en. In: *Journal of Guidance and Control* 3.2 (Mar. 1980), pp. 99–112. DOI: 10.2514/3.55956. URL: https://arc.aiaa.org/doi/10.2514/3.55956 (visited on 05/13/2022).

[KHAL04a] W. Khalil and E. Dombre. *Modeling, Identification and Control of Robots*. Kogan Page Science paper edition Modeling, identification & control of robots. Elsevier Science, 2004. URL: https://books.google.de/books?id=nyrY0Pu5kl0C.

[KHAL04b] W. Khalil and S. Guegan. “Inverse and direct dynamic modeling of Gough-Stewart robots”. In: *IEEE Transactions on Robotics* 20.4 (2004), pp. 754–761. DOI: 10.1109/TRO.2004.829473.

[KHAL04c] W. Khalil and O. Ibrahim. “General solution for the dynamic modeling of parallel robots”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 4. 2004, 3665–3670 Vol.4. DOI: 10.1109/ROBOT.2004.1308828.

[KHAL07] Wisama Khalil, Maxime Gautier, and Philippe Lemoine. “Identification of the payload inertial parameters of industrial manipulators”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 4943–4948. DOI: 10.1109/ROBOT.2007.364241.

[KHAL11] Wisama Khalil. “Dynamic Modeling of Robots Using Newton-Euler Formulation”. In: *Informatics in Control, Automation and Robotics*. Ed. by Juan Andrade Cetto, Jean-Louis Ferrier, and Joaquim Filipe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–20.

[KHAL86] W. Khalil and J. Kleinfinger. “A new geometric notation for open and closed-loop robots”. In: *Proceedings. 1986 IEEE International Conference on Robotics and Automation*. Vol. 3. 1986, pp. 1174–1179. DOI: 10.1109/ROBOT.1986.1087552.

[KHAL87] W. Khalil and J.-F. Kleinfinger. “Minimum operations and minimum parameters of the dynamic models of tree structure robots”. In: *IEEE Journal on Robotics and Automation* 3.6 (Dec. 1987), pp. 517–526. DOI: 10.1109/JRA.1987.1087145. URL: http://ieeexplore.ieee.org/document/1087145/ (visited on 05/13/2022).

[LUH80] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. “On-Line Computational Scheme for Mechanical Manipulators”. en. In: *Journal of Dynamic Systems, Measurement, and Control* 102.2 (June 1980), pp. 69–76. DOI: 10.1115/1.3149599. URL: https://asmedigitalcollection.asme.org/dynamicsystems/article/102/2/69/399118/On-Line-Computational-Scheme-for-Mechanical (visited on 05/13/2022).

[MAYE90] H. Mayeda, K. Yoshida, and K. Osuka. “Base parameters of manipulator dynamic models”. In: *IEEE Transactions on Robotics and Automation* 6.3 (June 1990), pp. 312–321. DOI: 10.1109/70.56663. URL: http://ieeexplore.ieee.org/document/56663/ (visited on 05/13/2022).

[MURR17] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 2017. URL: https://books.google.de/books?id=zDcPEAAAQBAJ.

[NAKA89] Y. Nakamura and M. Ghodoussi. “Dynamics computation of closed-link robot mechanisms with nonredundant and redundant actuators”. In: *IEEE Transactions on Robotics and Automation* 5.3 (1989), pp. 294–302. DOI: 10.1109/70.34765.

[PARK06] Kyung-Jo Park. “Fourier-based optimal excitation trajectories for the dynamic identification of robots”. en. In: *Robotica* 24.5 (Sept. 2006), pp. 625–633. DOI: 10.1017/S0263574706002712. URL: https://www.cambridge.org/core/product/identifier/S0263574706002712/type/journal_article (visited on 05/13/2022).

[PARK95] F.C. Park, J.E. Bobrow, and S.R. Ploen. “A Lie Group Formulation of Robot Dynamics”. In: *The International Journal of Robotics Research* 14.6 (1995), pp. 609–618. DOI: 10.1177/027836499501400606. eprint: https://doi.org/10.1177/027836499501400606. URL: https://doi.org/10.1177/027836499501400606.

[PFEI95] F. Pfeiffer and J. Holzl. “Parameter identification for industrial robots”. In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 2. 1995, 1468–1476 vol.2. DOI: 10.1109/ROBOT.1995.525483.

[PRES93] C. Presse and M. Gautier. “New criteria of exciting trajectories for robot identification”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. 1993, 907–912 vol.3. DOI: 10.1109/ROBOT.1993.292259.

[RACK12] Wolfgang Rackl, Roberto Lampariello, and Gerd Hirzinger. “Robot excitation trajectories for dynamic parameter estimation using optimized B-splines”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 2042–2047. DOI: 10.1109/ICRA.2012.6225279.

[STRA06] Gilbert Strang. *Linear algebra and its applications*. 4th ed. Belmont, CA: Thomson, Brooks/Cole, 2006.

[SWEV97] J. Swevers, C. Ganseman, D.B. Tukel, J. de Schutter, and H. Van Brussel. “Optimal robot excitation and identification”. In: *IEEE Transactions on Robotics and Automation* 13.5 (Oct. 1997), pp. 730–740. DOI: 10.1109/70.631234. URL: http://ieeexplore.ieee.org/document/631234/ (visited on 05/13/2022).

[UICK69] Jr. Uicker J. J. “Dynamic Behavior of Spatial Linkages: Part 1—Exact Equations of Motion, Part 2—Small Oscillations About Equilibrium”. In: *Journal of Engineering for Industry* 91.1 (Feb. 1969), pp. 251–265. DOI: 10.1115/1.3591539. eprint: https://asmedigitalcollection.asme.org/manufacturingscience/article-pdf/91/1/251/6498278/251_1.pdf. URL: https://doi.org/10.1115/1.3591539.

[VAND95] P.O. Vandanjon, M. Gautier, and P. Desbats. “Identification of robots inertial parameters by means of spectrum analysis”. In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 3. 1995, 3033–3038 vol.3. DOI: 10.1109/ROBOT.1995.525715.

[VENT09] Gentiane Venture, Ko Ayusawa, and Yoshihiko Nakamura. “A numerical method for choosing motions with optimal excitation properties for identification of biped dynamics - An application to human”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 1226–1231. DOI: 10.1109/ROBOT.2009.5152264.