Integration of Physics into Machine Learning for Enhanced Robot Dynamic Modeling

by

Erfaan Rezvanfar

B.Sc., University of Tehran, 2021

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

September 2024

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Integration of Physics into Machine Learning for Enhanced Robot Dynamic Modeling

submitted by **Erfaan Rezvanfar** in partial fulfillment of the requirements

for the degree of **Master of Applied Science**

in **Mechanical Engineering**

Examining Committee:

Dr. Clarence W. de Silva, Mechanical Engineering, UBC
Supervisor

Dr. Xiaoliang Jin, Mechanical Engineering, UBC
Supervisory Committee Member

Dr. Kefei Wen, Mechanical Engineering, UBC
Supervisory Committee Member

Abstract

Precise modeling of dynamical systems can be crucial for engineering applications. Traditional analytical models often have shortcomings when capturing real-world complexities, particularly due to challenges in system nonlinearity representation and model parameter determination. Data-driven models, such as deep neural networks (DNNs), offer better accuracy and generalization but require large quantities of high-quality data. The present thesis introduces a novel method termed the Synthesized-Data Neural Network (SDNN), which integrates analytical models, which represent physics, with DNNs to enhance the dynamic model. The main steps of the present method are given below, with particular reference to the practical situation of a physical robot. The first three degrees of freedom (DOF) of a Kinova Gen3 Lite manipulator are formulated using the Euler-Lagrange equations of motion. The experimental data are recorded from the manipulator. Simulated data from the analytical model are combined with the experimental data to train the neural network. The model's performance is evaluated using the Mean Squared Error (MSE) in real-time experiments with the Kinova Gen3 Lite manipulator. Training datasets represent 14 robotic trajectories, with the MSE calculated for four testing trajectories. The results obtained have led to the following conclusions: The SDNN model has shown improved performance in predicting joint torques when compared to the purely analytical model or the purely data-driven model. The SDNN, when trained with synthesized data from 14 trajectories (SDNN-14) achieved the lowest MSE of 2.1442, outperforming the analytical model (MSE of 2.8054) and the neural network trained solely on experimental data (MSE of 3.0521).

Lay Summary

For precise control of complex systems like robotic arms, scientists develop models that predict and guide their behavior. These models fall into two categories: *Analytical models*, based on physical laws, and *Experimental models*, which are mainly AI models trained on real-world data.

Analytical models, while based on fundamental physics principles, can be inaccurate due to a lack of knowledge, complexities, and necessary simplifications. On the other hand, Experimental models can suffer from issues like poor-quality or insufficient data. Our research presents a new technique that develops a model by combining the real-world pertinence of experimental approaches with reliable physical principles of analytical methods.

Our synthesized model has been experimentally verified to outperform individual models, with an accuracy improvement of 29.7% over experimental models and 23.6% over analytical models. This combined approach is particularly useful in controlling complex systems where neither analytical models nor experimental models alone provide sufficient accuracy.

Preface

The research presented in this thesis was conducted by Erfaan Rezvanfar at the Industrial Automation Laboratory (IAL), under the supervision of Professor Clarence W. de Silva, at the University of British Columbia (UBC). Dr. de Silva suggested the research topic, provided the basis of the research, instructed my colleague Jing Wang to work on the project, supervised the project throughout, and revised the draft thesis.

This thesis aims to bridge the gap between analytical models based on physical laws and experimental models trained on real-world data, by combining the two approaches. The research presented here demonstrates improvements achieved by our approach, in model accuracy, which will have positive practical implications in the modeling of complex dynamical systems.

My contributions to this project included developing the analytical model, operating the robotic manipulator and conducting experiments, reducing the degrees of freedom of the robot for simulation, and writing the thesis.

Sections 2.5 and 2.6, which involve the development of the neural network and the evaluation of its performance, were primarily done by my colleague, Jing Wang.

The experiments were conducted on a Kinova Gen3 Lite manipulator, which was provided by the laboratory of Dr. Christoph Sielmann.

I hope that the findings of this thesis will contribute to further advancements in the field and inspire future research in the modeling and control of dynamical systems.

Table of Contents

Appendices

List of Tables

List of Figures

List of Symbols

$\boldsymbol{\alpha}$ The angular acceleration of a body (vector).

$\mathbf{I}$ The inertia tensor in a body.

$\boldsymbol{v}$ The linear velocity of a body at its origin (vector).

$\boldsymbol{v}_c$ The linear velocity of a body at its center of mass (vector).

$\boldsymbol{\omega}$ The angular velocity of a body (vector).

$\dot{q}$ The angular velocity of a joint (scalar). It may be referred to as "joint velocity".

$\ddot{q}$ The angular acceleration of a joint (scalar). It may be referred to as "joint acceleration".

τ The value of the torque of a joint (scalar).

List of Abbreviations

ANN: Artificial Neural Network

CNN: Convolutional Neural Network

DOF: Degree-of-Freedom

NN: Neural Network

PINN: Physics Informed Neural Network

SDNN: Synthesized-Data Neural Network

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Clarence W. de Silva, for his invaluable guidance and support, including financial support, throughout this research. I also extend my thanks to my lab mates for their kindness and support in my research and life.

Special thanks are due to my family, the pillars of my strength, for their unwavering belief in me and for providing the foundation upon which I have built my academic career. I am also grateful to my friends for their support and companionship during my master's studies.

Finally, I would like to acknowledge the funding support, through the grant RGPIN-2023-03243, from the Natural Sciences and Engineering Research Council of Canada (NSERC), held by Dr. Clarence de Silva, which made this research possible.

Chapter 1

Introduction

1.1 Background

The mechanics of a dynamical system, such as a robotic arm or a drone, can be modeled in several ways. The model of a dynamical system represents the physical behavior of that system and is an essential tool for its design and control. Therefore, modern engineering and industrial automation rely on the development of a precise model to optimize the performance [1–3] and the safety [4, 5] of the dynamical system.

1.2 Current Approaches and the Existing Gaps

There are various ways to model a dynamical system. A mathematical model (also known as an analytical model) is widely used in this context. The mathematical model represents the system's dynamics through a set of equations that come from the fundamental principles of physics [6]. Mathematical modeling has been used in engineering and scientific research for decades, offering a methodical way to model and understand dynamical systems. However, mathematical models come with their limitations. They simplify the real-world system and may not completely consider the system's specifications. The resulting abstractions may limit their ability to replicate the behavior of a complex dynamical system entirely. In response to this gap, some researchers have turned to Artificial Neural Network (ANN) models, or simply Neural Network (NN) models, as an alternative approach [7]. Neural network models are computational architectures inspired by the human brain, using interconnected nodes to process data and make predictions. Instead of using mathematical equations, this method directly observes and learns from data obtained from the real system itself. For example, an input, such as force or current, is applied to the real physical dynamic system, and the corresponding output is recorded. The Neural Network model is trained using this input-output data, thereby creating a model that can adapt to the system's unique characteristics better than a mathematical model. How-

ever, while NN models do well in capturing the specifications of the system from data, they lack interpretability and an understanding of the physical laws governing the system. It is like a black box that predicts the output for a specific input to the box. Additionally, the dataset that the NN model is trained on may not be big enough, or the data may be too noisy, or in some cases, if the operating environment of the robot changes the data will not represent the corresponding behavior, and hence the model will not perform well. While such problems limit the use of NN models, mathematical models do not face such issues. So a model solely based on NN may not always be the most effective approach to model a dynamical system. Then it is likely that we can benefit from the strengths of both mathematical models and NN-based models and compensate for the limitations of each approach.

1.3 Research Objectives

In the present study, we introduce a new model that we call the Synthesized Data Neural Network (SDNN) model. In this approach, we merge the data from physics-based analytical models with real-world data and train a Neural Network over this synthesized dataset. Although the resulting model may not compensate for all the limitations of one another, it reduces their individual shortcomings. In our methodology, first, we derive a physics-based analytical model to simulate the system under ideal conditions and collect input-output data from this model. Second, we gather real-world data (experimental data) from the real system's operation. The two datasets are then combined and used to train a Neural Network model. The achieved model will rely partly on the laws of physics (from the analytical model's data) and partly on the specific conditions of the system (from the real-world data). As a result, the developed model is less likely to deviate from the expected natural performance (physical laws) of a dynamical system and can compensate for the simplifications made in a mathematical model.

1.4 Related Work

In the field of Dynamical Systems, the merging of physical principles with machine learning techniques has seen varied approaches. This integration primarily revolves around two aspects: how physics is embedded into machine learning models and the choice of machine learning methods like Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs), among others.

One widespread use of ANNs involves solving Partial Differential Equations (PDEs) derived from the physical equations of Fluid Dynamic Systems [8, 9]. Alternatively, researchers have also utilized physical equations to enhance feature engineering within ANNs [10]. The adoption of CNNs is on the rise, with some being employed to formulate algebraic equations representing the dynamics of specific systems, as seen for example in bearing systems [11] Moreover, researchers have utilized simulation results grounded in physics to define parameters within CNN architectures [12].

The work by Rackauckas et al. introduces the concept of Universal Differential Equations (UDEs), which integrate neural networks and differential equations into a unified modeling framework. At the core of the UDE approach is the use of neural networks to parameterize the right-hand side of a differential equation, allowing the model to learn the underlying dynamics from data while retaining the structure and interpretability of the differential equation. The authors propose several strategies for constructing UDE architectures, including the use of residual connections, Runge-Kutta integrators, and adjoint-based sensitivity analysis for efficient training. Importantly, UDEs enable the incorporation of known physical principles and constraints into the model structure, ensuring that the learned dynamics adhere to relevant scientific laws and principles [13].

A paper by Brunton et al. presents a new approach that combines data-driven machine learning with traditional physics-based modeling. Their key innovation is a technique called "sparse identification of nonlinear dynamics" (SINDy), which can automatically discover the underlying mathematical equations that govern a complex system, using only observational data. Rather than relying on prior physics knowledge, SINDy analyzes patterns in the data to determine the minimal set of mathematical terms needed to describe the system's behavior. By keeping the final model sparse and interpretable, the authors show how this data-driven approach can provide insights into the physical mechanisms at play [14].

The study by Weinan et al. focuses on two main methodological aspects of integrating machine learning with physics-based modeling. Firstly, the authors highlight the importance of properly imposing physical constraints within the machine learning models in order to develop interpretable and reliable physics-based representations. This is a key challenge, as machine learning models need to respect the underlying physical principles and laws. Secondly, the paper emphasizes the criticality of obtaining optimally representative datasets for training the machine learning components. The quality and coverage of the training data are crucial for successfully applying machine learning techniques to physics problems [15].

Compared to the available approaches to integrate physics into machine learning, our method is different. We employ a conventional neural network trained on synthesized data, a blend of physics-based data and real-world data. This strategy represents a data-fusion methodology, aiming to establish a link between theoretical frameworks and practical applications in Dynamical Systems.

Chapter 2

Methodology

This section outlines the methodology used in the present study to integrate experimental data with an analytical model for precise modeling of the mechanics of a dynamical system, focusing on a Kinova Gen3 Lite robotic manipulator (Figure 2.1). The process of experimental data collection is described first, highlighting the reduction of the robot's degrees of freedom for analysis. Subsequently, the development of a mathematical model of its mechanics using Lagrange's method is detailed, emphasizing the derivation of the equations of motion for a three-degree-of-freedom (3-DOF) robot arm. Next, the Synthesized-Data Neural Network (SDNN) model is introduced, which merges the physics-based analytical model with real-world experimental data. Finally, the integration of data and model validation techniques are discussed to assess the prediction accuracy of the SDNN model. In the present study, the dynamical system is a Kinova Gen3 Lite, reduced to three degrees of freedom from its full (6-DOF) capability (Figure 2.2).

Figure 2.1: Kinova Gen3 Lite manipulator [1]

[1] https://www.kinovarobotics.com/product/gen3-robots

The methodology consists of the following steps:

1. **Experimental Data Collection:** The process of gathering experimental data from the robot.

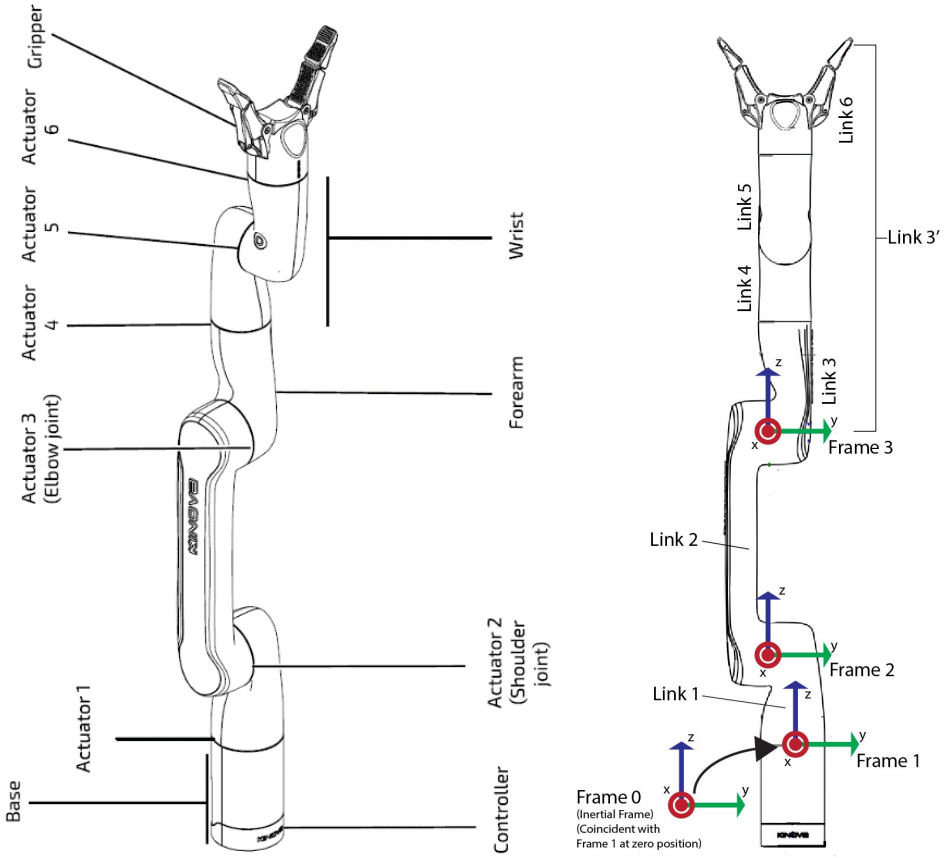
2. **Reduction of Degrees of Freedom:** The computations needed to reduce the robot's degrees of freedom from 6 DOF to 3 DOF, for the present modeling purposes.

3. **Analytical Model Development:** The derivation of the mechanics-based analytical model using Lagrange's method for a 3 DOF robot arm.

4. **Analytical Data Generation:** The process of generating data using the analytical model.

5. **Synthesized-Data Neural Network (SDNN) Model:** Introduction of the structure of the SDNN model that merges the physics-based analytical model with experimental data.

6. **Evaluation of SDNN Model Effectiveness:** Evaluation of the effectiveness of the SDNN model when compared to both a standalone neural network and an analytical model.



(a) (b)

Figure 2.2: a) Main components of the Kinova Gen3 Lite manipulator. b) Frame assignments for reduced degrees of freedom (composed of links 1, 2, and 3')

2.1 Experimental Data Collection

Experimental data were collected using a Kinova Gen3 Lite robotic manipulator (Figure 2.1). The robotic manipulator is equipped with sensors to capture real-time response data for feedback. Our research required data on joint angle, joint velocity, and torque. While specific sensors were available for joint angle and joint velocity measurements, the manipulator lacked a dedicated torque sensor. Therefore, torque data were estimated based on each joint's current using the API provided by Kinova.

As previously specified, only the initial three joints of the robot are designated as movable joints, while the remaining three are set to their zero positions. Consequently, experimental data is exclusively gathered by manipulating the first three joints while keeping the other three joints in their fixed configuration. To achieve this, random trajectories comprising angle data were independently assigned to joints 1, 2, and 3. The torques generated during these movements were recorded using the robot's built-in sensors. A total of 14 experiments were conducted using the Reduced-Degree-of-Freedom Kinova Gen3 Lite to create a dataset encompassing a variety of dynamic behaviors. Each of these experiments was 20 seconds long and was taken at a frequency of 1kHz. This high frequency was more than adequate and, therefore, was reduced to 100 Hz in the present research.

The input to the robot was the required position sequence. The robot's built-in controller then generated the appropriate torques to execute the specified trajectory, which were collected as the robot's output.

It was immobilizing specific joints that resulted in the reduction of the robot's degrees of freedom. However, to analytically model the reduced-degree-of-freedom robot, it is necessary to calculate the properties (mass, inertia, length of the links, and center of mass location) of the robot with the new bodies the reduction of degrees of freedom has formed. While the properties of the first two links remain known, a new third link has been introduced, which is the combination of the fourth through sixth links. Determining the properties of the new third link requires calculating them based on the properties of each of the four individual links that form it.

2.2 Reduction of Degrees of Freedom

2.2.1 Advantages of using 3 DOF instead of 6 DOF

Although optimizing the model for a 6 DOF robot may seem advantageous due to its comprehensive industrial applications, the complexity involved

in developing an analytical model for such a system can detract the focus from the primary objective of the present research. Thus, a 3 DOF system was chosen as a more suitable alternative for the physical system in this research. A 3 DOF system presents a compromise between simplicity and complexity, allowing for meaningful analysis without overwhelming the study with unnecessary complications and problems particularly caused by computational errors. The compromised level of complexity also ensures that the performance differences observed are significant and not obscured by overly simplistic models or masked by non-focal effects.

The primary aim is to determine whether combining physics-based and experimental data can enhance the system model. For this purpose, a less complex system like the 3 DOF robot would provide a more desirable platform for experimentation, observation, and analysis.

Additionally, third-party software packages for simulating the behavior of serial robots (such as MATLAB Robotics System Toolbox[2] and RoboDK[3]) were considered. However, these tools function as black boxes, giving limited insight into the factors influencing the discrepancies between simulated and experimental data. Due to this lack of transparency, they were deemed unreliable for the present study and the robot was modelled after reducing it to 3 degrees of freedom.

2.2.2 The reduction of the degrees of freedom

The degrees of freedom of the serial robot were reduced by immobilizing specific joints, resulting in a system with three active joints and three corresponding links. Upon reducing the degrees of freedom of the robot, links 3 through 6 are considered to be rigidly connected to each other, introducing a new link. The new link is referred to as link $3'$ and is shown in Figure 2.2b. The properties -mass, center of mass, moment of inertia, and length— of the new link were calculated based on the properties of the individual links comprising it.

The necessary equations for the needed calculations are provided below.

[2]Robotics System Toolbox - MATLAB (mathworks.com)

[3]Simulator for industrial robots and offline programming - RoboDK

Computing the new mass (m)

The mass of a link that is a combination of several consecutive links is the sum of the masses of those links:

$$\mathrm{m}_{\text{combined}} = \sum_{i=j}^{n} \mathrm{m}_i \tag{2.1}$$

In Equation (2.1), $\mathrm{m}_{\text{combined}}$ denotes the total mass of the combination of links j through n of an n-link serial robot. Therefore, the mass of link $3'$ in our case will be:

$$\mathrm{m}_{3'} = \sum_{i=3}^{6} \mathrm{m}_i \tag{2.2}$$

Computing the new Center of Mass (c)

The center of mass of a link that is a combination of several consecutive links is calculated using the following equation, as clear from the moment balance:

$$\mathbf{c}_{\text{combined}} = \frac{\sum_{i=j}^{n} \mathrm{m}_i \mathbf{c}_i}{\mathrm{m}_{\text{combined}}} \tag{2.3}$$

Here, $\mathbf{c}_{\text{combined}}$ represents the coordinates of the new center of mass, and $\mathbf{c}_i$ denotes the coordinates of the center of mass of each individual link. The center of mass of link $3'$ in our case will be:

$$\mathbf{c}_{3'} = \frac{\sum_{i=3}^{6} \mathrm{m}_i \mathbf{c}_i}{\mathrm{m}_{3'}} \tag{2.4}$$

Computing the new Moment of Inertia (I)

The inertia of a link that is a combination of several consecutive links is calculated using the equation:

$$\mathbf{I}_{\text{combined}} = \sum_{i=j}^{n} \mathbf{I}_i + \sum_{i=j}^{n} \mathrm{m}_i \mathbf{d}_i^2 \tag{2.5}$$

This Equation results from the well-known parallel axis theorem. Here, $\mathbf{I}$ denotes the moment of inertia. It is important to note that all the inertia $\mathbf{I}_i$ are represented about the center of mass of their corresponding link's center of mass $\mathbf{c}_i$ and with respect to the inertial frame. The inertial frame and the body-attached frames of the three links of the robot are depicted in

Figure 2.2b. The inertial frame coincides with frame 1 when $q_1 = 0$. The vector $\mathbf{d}_i$ represents the distance between link i's center of mass and the new "combined" center of mass:

$$\mathbf{d}_i = \mathbf{c}_i - \mathbf{c}_{\text{combined}} \tag{2.6}$$

According to Equation (2.5), the moment of inertia of our new link can be computed by calculating the moment of inertia of each of the old links around the new center of mass (the parallel axis theorem) and summing them up. Therefore,

$$\mathbf{I}_{3'} = \sum_{i=3}^{6} \mathbf{I}_i + \sum_{i=3}^{6} \mathrm{m}_i \mathbf{d}_i^2 \tag{2.7}$$

Computing the new Length (l)

The length of a link that is a combination of several consecutive links is calculated by adding the lengths of all those links. Similar to the moment of inertia, all the lengths must be represented in the directions of the inertial frame and not each link's body frame orientation, assuming that the last several links are fully stretched and aligned (as true in the conducted experiments),

$$\mathbf{l}_{\text{combined}} = \sum_{i=j}^{n} \mathbf{l}_i \tag{2.8}$$

Therefore, the length of link $3'$ is calculated as:

$$\mathbf{l}_{3'} = \sum_{i=3}^{6} \mathbf{l}_i \tag{2.9}$$

Mass m is scalar, moment of inertia $\mathbf{I}$ is a 3×3 matrix, center of mass $\mathbf{c}$, distance $\mathbf{d}$ and length $\mathbf{l}$ are 1×3 row vectors. The corresponding elements (with subscripts) are scalars. The links' lengths could be considered scalar, however, they are considered as vectors for ease of calculations. The above equations will result in the properties of link $3'$ which will be used along with the properties of links 1 and 2 for deriving the analytical model of the robot.

2.3 Analytical Model Development

To complement the experimental data, an analytical model, also referred to as a mathematical model or a physics-based model, was developed using Lagrange's method for the reduced-degree-of-freedom Kinova Gen3 Lite

manipulator. This results in the equations of motion of the robotic system, providing a set of differential equations that represent the dynamics of that system. Lagrange's method involves expressing the kinetic energy K and potential energy U of the system in terms of generalized coordinates and their time derivatives. In the context of our robotic system, these generalized coordinates correspond to the joint angles q, which define the configuration of the robot at any given time.

The Lagrange equation is given by,

$$\boldsymbol{\tau} = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial L}{\partial \boldsymbol{q}} \tag{2.10}$$

Here, $\boldsymbol{\tau}$ is the joint torque vector, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are the joint angle and joint velocity vectors respectively, and L is the Lagrangian. Notably, $\boldsymbol{\tau}$, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are $n \times 1$ vectors where n indicates the degrees of freedom of the serial robot and is equal to 3 in our case. The Lagrangian is computed from the kinetic energy K and potential energy U of the system, as

$$L = K - U \tag{2.11}$$

Given the properties of the three links of the robot, Equation 2.10 provides an analytical model for the required torque of all joints ($\boldsymbol{\tau}$), as a function of joint angles, joint velocities, and joint accelerations ($\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$, and $\ddot{\boldsymbol{q}}(t)$).

2.3.1 Kinetic energy K

In a mechanical dynamic system, the total kinetic energy K is the sum of the kinetic energies associated with each of its degrees of freedom.

$$K = \sum_{i=1}^{3} K_i \tag{2.12}$$

Here, K is the kinetic energy of the robot, which is needed for the Lagrangian. K_i is the kinetic energy of the i^{th} link, and n is the number of links. In our case, n is 3.
The kinetic energy of a single link i (rigid body), is calculated as,

$$K_i = \frac{1}{2}\mathrm{m}_i \boldsymbol{v}_{\mathrm{c}_{i,0}}^T \boldsymbol{v}_{\mathrm{c}_{i,0}} + \frac{1}{2}\boldsymbol{\omega_i}^T \boldsymbol{R}_{i,0}\mathbf{I_i}\boldsymbol{R}_{i,0}^T\boldsymbol{\omega_i} \tag{2.13}$$

Here, m_i is the mass of the link (scalar), $\boldsymbol{v_i}$ is the linear velocity vector (3x1), $\boldsymbol{\omega_i}$ is the angular velocity vector (3x1), $\boldsymbol{R}_{i,0}$ is the rotation matrix

(3x3) between the body-attached frame and the inertial frame, and $\mathbf{I_i}$ is the moment of inertia tensor (3x3) around the center of mass of the body and with respect to the body-attached frame.
The mass m and inertia $\mathbf{I}$ values for the first two links are known and those of the third link are computed, as indicated in section 2.2.2. The rotation matrices $\boldsymbol{R}_{i,0}$, the angular velocities $\boldsymbol{\omega}_i$, and the linear center of mass velocities $\boldsymbol{v}_{c_{i,0}}$ are the unknowns of Equation 2.13, which will be computed now.

Rotation Matrix $\boldsymbol{R}$

With the frame assignments shown in Figure 2.2, the rotation matrices between the four frames (inertial frame being frame 0) are as follows:

$$\begin{aligned} \boldsymbol{R}_{0,1} &= \boldsymbol{R}(-q_1, \hat{\mathbf{k}}) \\ \boldsymbol{R}_{1,2} &= \boldsymbol{R}(-q_2, \hat{\mathbf{j}}) \\ \boldsymbol{R}_{2,3} &= \boldsymbol{R}(q_3, \hat{\mathbf{j}}) \end{aligned} \tag{2.14}$$

The rotation matrix $\boldsymbol{R}$ is computed from Rodrigues' rotation formula which is presented below:

$$\mathbf{R} = \mathbf{I} + \sin q\mathbf{S} + (1 - \cos q)\mathbf{S}^2 \tag{2.15}$$

In this equation, $\mathbf{S}$ is the skew-symmetric matrix of the unit vector representing the axis of rotation. More information about the definition of rotation matrix can be found in Appendix A.1.
Taking into account that,

$$\boldsymbol{R}_{i,j} = \boldsymbol{R}_{j,i}^T \tag{2.16}$$

the rotation matrices $\boldsymbol{R}_{i,0}$, which are required in Equation 2.13, are computed as follows:

$$\begin{aligned} \boldsymbol{R}_{2,0} &= \boldsymbol{R}_{1,0} \times \boldsymbol{R}_{2,1}, \\ \boldsymbol{R}_{3,0} &= \boldsymbol{R}_{1,0} \times \boldsymbol{R}_{2,1} \times \boldsymbol{R}_{3,2} \end{aligned} \tag{2.17}$$

Angular Velocity $\boldsymbol{\omega}$

The angular velocity vector $\boldsymbol{\omega}_i$ of the three rigid links are obtained as follows:

$$\begin{aligned} \boldsymbol{\omega}_1 &= \dot{q}_1 \hat{\boldsymbol{k}}, \\ \boldsymbol{\omega}_2 &= \boldsymbol{R}_{1,2}\boldsymbol{\omega}_1 + \dot{q}_2 \hat{\boldsymbol{j}}, \\ \boldsymbol{\omega}_3 &= \boldsymbol{R}_{2,3}\boldsymbol{\omega}_2 - \dot{q}_3 \hat{\boldsymbol{j}}. \end{aligned} \tag{2.18}$$

It is noted that $\dot{q}_i$ is the derivative of the i^{th} joint angle. In other words, $\dot{q}_i$ (scalar) is the joint velocity of link i, while $\boldsymbol{\omega}_i$ (vector) is the angular velocity of that link in its body-attached frame.

Linear Velocity at the center of mass $\boldsymbol{v}_\mathbf{c}$

The linear velocity at the center of mass of each link with respect to its body-attached frame $\boldsymbol{v}_{\mathrm{c}_{i,i}}$ is calculated as:

$$\boldsymbol{v}_{\mathrm{c}_{i,i}} = \boldsymbol{\omega}_i \times \mathbf{c}_i \tag{2.19}$$

In Equation 2.19, $\mathbf{c}_i$ refers to the coordinates of the center of mass of link i in that link's body-attached frame.
Consequently, the linear velocity of each center of mass with respect to the inertial frame $\boldsymbol{v}_{\mathrm{c}_{i,0}}$ is calculated:

$$\begin{aligned} \boldsymbol{v}_{\mathrm{c}_{1,0}} &= \boldsymbol{R}_{1,0} \times \boldsymbol{v}_{\mathrm{c}_{1,1}} \\ \boldsymbol{v}_{\mathrm{c}_{2,0}} &= \boldsymbol{v}_{2,0} + \boldsymbol{R}_{2,0} \times \boldsymbol{v}_{\mathrm{c}_{2,2}}, \\ \boldsymbol{v}_{\mathrm{c}_{3,0}} &= \boldsymbol{v}_{3,0} + \boldsymbol{R}_{3,0} \times \boldsymbol{v}_{\mathrm{c}_{3,3}}. \end{aligned} \tag{2.20}$$

Where, $\boldsymbol{v}_{i,0}$ indicates the linear velocity of the frame (origin) of link i:

$$\begin{aligned} \boldsymbol{v}_{2,0} &= \boldsymbol{R}_{1,0} \times \boldsymbol{\omega}_1 \times \mathbf{l}_1, \quad (\mathbf{l}_1\text{: length of link 1}) \\ \boldsymbol{v}_{3,0} &= \boldsymbol{v}_{2,0} + \boldsymbol{R}_{2,0} \times \boldsymbol{\omega}_2 \times \mathbf{l}_2, \quad (\mathbf{l}_2\text{: length of link 2}) \end{aligned} \tag{2.21}$$

Using Equations 2.13 through 2.21, and substituting into Equation 2.12, the kinetic energy of the robot is computed as a function of $q(t)$, $\dot{q}(t)$, and $\ddot{q}(t)$.

2.3.2 Potential energy U

The potential energy of the entire robot U is the sum of the potential energy of all three links.

$$U = \sum_{i=1}^{3} U_i \tag{2.22}$$

The potential energy of a single link i (rigid body) is computed as follows:

$$U_i = \mathrm{m}_i \mathrm{g} h_i \tag{2.23}$$

In this equation, g is the gravitational acceleration (scalar), and h_i is the height of the center of mass of link i (scalar), which is the only value that must be computed here. As shown in Figure 2.3, the plane containing the

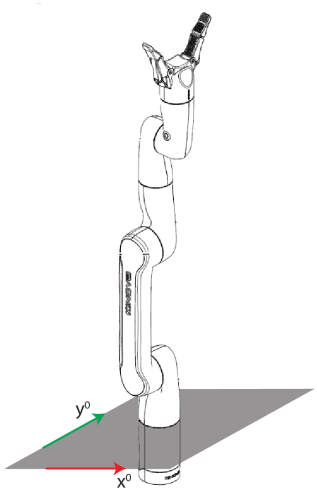


Figure 2.3: The reference plane for computing the potential energy. The plane coincides with Frame 0

x and y axes of the inertial frame (frame 0) is considered the reference for computing the potential energy. The height of the center of mass of each link (h_i) is the distance it has with the described reference plane:

$$\begin{aligned} h_1 &= \mathbf{c}_1^z, \\ h_2 &= \|\mathbf{l}_1\| + \mathbf{c}_2^z \cos q_2, \\ h_3 &= \|\mathbf{l}_1\| + \|\mathbf{l}_2\| \cos q_2 + \mathbf{c}_3^z \cos(q_2 - q_3). \end{aligned} \tag{2.24}$$

Here, $\mathbf{c}_i^z$ represents the z-component of the center of mass of link i. Additionally, $\mathbf{l}_i$ is the length of link i, and $\|\mathbf{l}_i\|$ denotes the norm (magnitude) of the $\mathbf{l}_i$ vector.
The potential energies of the three links are therefore calculated as follows:

$$\begin{aligned} U_1 &= \mathrm{m}_1 \mathrm{g} \mathbf{c}_1^z, \\ U_2 &= \mathrm{m}_2 \mathrm{g} (\|\mathbf{l}_1\| + \mathbf{c}_2^z \cos q_2), \\ U_3 &= \mathrm{m}_3 \mathrm{g} (\|\mathbf{l}_1\| + \|\mathbf{l}_2\| \cos q_2 + \mathbf{c}_3^z \cos(q_2 - q_3)). \end{aligned} \tag{2.25}$$

By substituting from Equation 2.25 into Equation 2.22, the potential energy of the robot is obtained as a function of $q(t)$, $\dot{q}(t)$, and $\ddot{q}(t)$.

2.3.3 Formulating the Lagrange equation

Knowing T and U, the Lagrangian L is determined using Equation 2.11 and substituted into Equation 2.10, to obtain the equations of motion. Because the simplified robot has only three degrees of freedom, Equation 2.10 will be in the form,

$$\boldsymbol{\tau} = \begin{pmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{pmatrix} = \begin{pmatrix} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_1}\right) - \frac{\partial L}{\partial q_1} \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_2}\right) - \frac{\partial L}{\partial q_2} \\ \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_3}\right) - \frac{\partial L}{\partial q_3} \end{pmatrix} \tag{2.10}$$

These steps were carried out and the analytical model was developed for the Reduced Kinova Gen3 Lite robot in Python.

2.4 Analytical Data Generation

2.4.1 Analytical Data - Collection

The analytical model provides equations for joint torques τ as functions of $q, \dot{q}$, and $\ddot{q}$. The same inputs were given to the analytical model as to the physical robot. The same inputs were provided to both the analytical model and the physical robot. For the physical robot, the input consisted of a joint angle trajectory (a time sequence of q), while the time sequences of $\dot{q}$ and $\ddot{q}$ are required as inputs to the analytical model to obtain the output τ. The joint velocities $\dot{q}$ were read from the robot's relevant built-in sensors. However, joint accelerations $\ddot{q}$ are also required, but not directly provided by the robot sensors. Therefore, the joint accelerations were computed using the joint velocity data.

The time derivative of the joint velocity $\dot{q}$ was calculated to obtain the joint acceleration $\ddot{q}$. The Finite Difference method was used which approximates the derivative as,

$$\ddot{q}(t) \approx \frac{\dot{q}(t + \Delta t) - \dot{q}(t - \Delta t)}{2\Delta t} \tag{2.26}$$

where:

- $\ddot{q}(t)$ is the joint acceleration at time t.
- $\dot{q}(t)$ is the joint velocity at time t.

- Δt is the time interval between consecutive data points (1 ms in the present application).

For boundary points, forward and backward differences are used. This method provides a reliable and efficient way to derive the joint accelerations from the given joint velocity data.

The joint velocity data collected from the robot are raw and not suitable for theoretical models. The reason for using feedback data for the analytical model is that the real data (the trajectories) are generated based on the robot's controller. This means that the joint velocities are not defined theoretically, but generated by the robot's controller. Alternative approaches are discussed in Section 4.3.1 for future consideration. In view of this, before utilizing the feedback angular velocities, preprocessing them is necessary.

2.4.2 Analytical Data - Preprocessing

The feedback data has a frequency of 1 kHz, resulting in discrete data points spaced 1 ms apart. This high frequency is, however, more than necessary and was reduced to 100 Hz for the present study. Additionally, the joint velocity data extracted from the robot sensors were highly noisy. A simple moving average (SMA) filter was applied to the data to reduce the noise. This step is essential for obtaining cleaner joint velocity data and hence, for accurately computing the joint acceleration data. The moving average is a simple yet effective method for smoothing noisy data, where averaging over a specified window of data points is performed. Our joint velocity data were gathered over 20 seconds at a frequency of 100 Hz. The window size for the filter was 40 data points, equivalent to averaging over 0.4 seconds of data.

As mentioned earlier, the filtered joint velocity data was used to create the joint acceleration data. Since the joint velocity data is not perfect even after passing through the SMA filter, and given that taking the derivative tends to amplify noise, the resulting joint acceleration data is also noisy. Therefore, the joint acceleration data was further filtered to reduce noise. The same SMA filter applied to the joint velocity data was used on the joint acceleration data. That is because derivation causes the noise in the velocity data to amplify and a filter on the acceleration data can help reduce that noise.

The units of the data were converted to the desired units. The desired units are presented in Table 2.1:

Table 2.1: Desired units for the data

Quantity	**Symbol**	**Unit**
Joint Angle	q	rad
Joint Velocity	$\dot{q}$	rad/s
Joint Acceleration	$\ddot{q}$	rad/s^2
Torque	τ	Nm

2.5 Synthesized-Data Neural Network Model

As mentioned in Section 2.1, 14 trajectories were executed on the Kinova Gen3 Lite robot using only its first three joints. The sensor readings for joint angles $\boldsymbol{q}$, joint velocities $\dot{\boldsymbol{q}}$, and joint torques $\boldsymbol{\tau}$ were collected.

Out of the 14 trajectories, 10 were randomly selected for training, and the remaining 4 were used for testing the model. Additionally, as explained in Section 2.4, analytical data were generated corresponding to the 10 experimental training datasets.

The present research aims to analyze the effectiveness of adding analytical data to the training process. Therefore, the mapping function (the Neural Network model) is chosen to be a Multi-Layer Perceptron (MLP), a lightweight and straightforward method for training the robot model. If the addition of analytical data proves effective, more complex mapping functions can be implemented to improve the model quality.

The primary goal has been demonstrating the effectiveness of the Neural Network model trained on the synthesized dataset, which combines both experimental and analytical data (1). For comparison, the Neural Network was also trained separately on each dataset alone: on experimental data only (2) and on analytical data only (3). Additionally, the analytical model (not a Neural Network) was included in the analysis (4). Consequently, there are four models to be compared:

1. Neural Network trained on Synthesized Data
2. Neural Network trained on Experimental Data only
3. Neural Network trained on Analytical Data only
4. Analytical Model (not Neural Network)

2.6 Evaluation of SDNN Model's Effectiveness

The performance of each of the four models described in the previous section was evaluated using a set of experimental data. Despite its limitations, experimental data is a practical reference for initial evaluation. A more robust and reliable method for evaluating the effectiveness of the SDNN model is suggested in Section 4.3.2, which outlines future work and potential improvements.

The performance of the proposed SDNN model was assessed using the mean square error (MSE) as the evaluation metric. The MSE was determined by comparing the predicted joint torques generated by the SDNN model with the ground truth joint torques (known from the experiments) required for the end-effector to follow the desired trajectory. Lower MSE values indicate a closer match between the predicted and actual joint torques.

For the four models mentioned in the previous section, each evaluation experiment was repeated five times. Different initialization seeds were used for the neural network parameters in each of the five experiments.

Chapter 3

Results and Discussion

3.1 Reduction of Degrees of Freedom

Given equations 2.1-2.8 and the properties of the links of the 6-DOF Kinova Gen3 Lite robot, the properties for the reduced-degrees-of-freedom 3-DOF robot are established, as given in Table 3.2. In this table, the moment of inertia matrices are computed around the center of mass of each link with respect to their body-attached frame. The length of each link is considered in the sole direction of its longest length. The center of mass of each link is with respect to the body-attached frame of that link. In this thesis, all the body-attached frames are considered parallel to the inertial frame when at their zero positions ($\boldsymbol{q} = 0$), as depicted in Figure 2.2b.

3.2 Analytical Model Development

The analytical model, i.e., the equations of motion, for the reduced (3-DOF) serial manipulator was derived. The inputs to the model are nine values in total: the angles, joint velocities, and joint accelerations of the three joints ($\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$), each represented as a 3×1 vector. The output is the joint torques vector $\boldsymbol{\tau}$ (3×1).

The Python code used to derive the equations of motion is provided in Appendix B. The resulting equations of motion themselves are too lengthy to be included here and can be obtained by running the provided code.

3.3 Analytical Data Generation

The results of data generation using the analytical model are presented in this section. As explained in section 2.4, the inputs to the analytical model are joint angles, joint velocities, and joint accelerations of the three joints, and the outputs are the joint torques of the three joints. From the mentioned inputs, the joint angles and joint velocities are in this case generated by the Kinova Gen3 Lite's controller, and the joint accelerations are calculated as

the derivative of the joint velocities. The preprocessing steps, including the application of the SMA filter to the joint velocity and acceleration data, significantly improved the quality of the analytical results. In Figure 3.1a, the original and filtered joint velocity data are compared for joint no. 2 of the robot, with a random trajectory. Figure 3.1b shows the comparison of the joint acceleration data of joint no.2, for the same random trajectory, before and after filtering.

The filtering process reduced noise, thereby making the data more suitable for analytical purposes. This step was crucial, as the derivation of acceleration from velocity data amplifies noise, which could lead to significant inaccuracy in the torque calculations.

3.4 Evaluation of the SDNN Model

The average MSE errors for testing the Synthesized-Data Neural Network (SDNN) model, as well as the other three models employed for comparison, are given in Table 3.1.

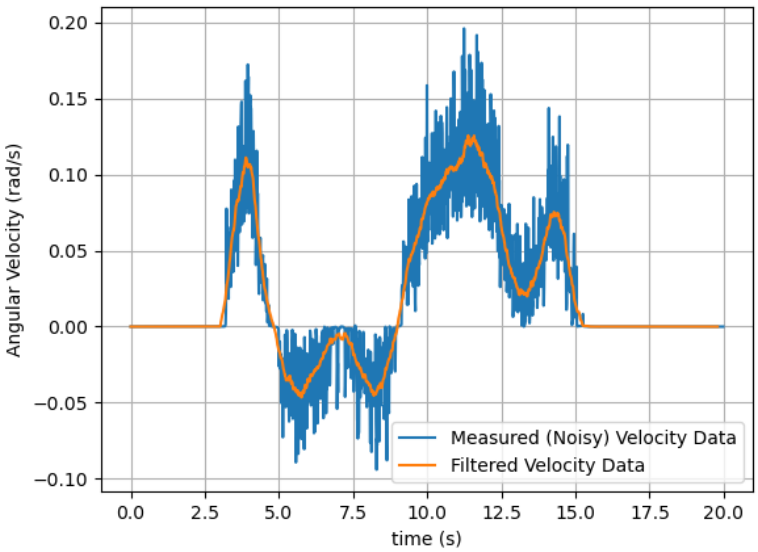
Table 3.1: The Averaged Mean Squared Errors of different models compared to experimental data. A lower value corresponds to better performance.

Models	**Mean Squared Error (RMS)**
Analytical Model (Euler-Lagrange)	2.8054 ± 0.0
NN trained on Experimental Data only	3.0521 ± 0.4235
NN trained on Synthesized-Data (SDNN)	**2.1442** ± 0.1159

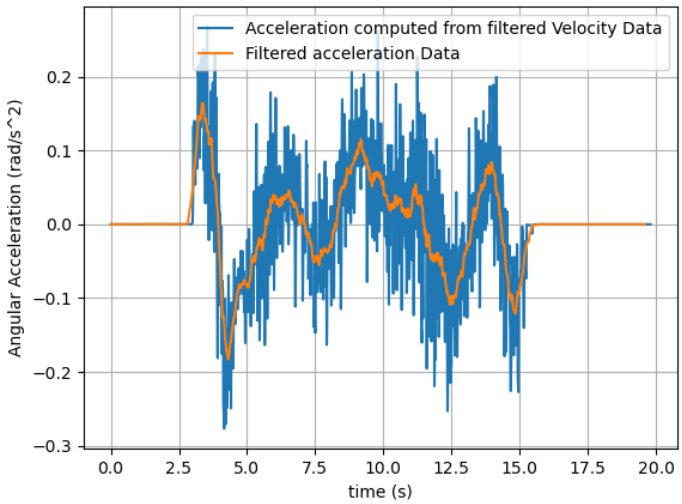
The experiments for evaluating the performance of the four models were repeated five times. The standard deviations of these repeated experiments are included in Table 3.1. The Lagrange analytical model provides the best robustness, with zero variance.

The evaluation results show a significant improvement in the accuracy with the proposed SDNN model over the baseline methods. The mean square error (MSE) on the testing set for the Lagrange analytical model, which is a conventional analytical model, is 2.81.

The model trained solely with experimental data results in an MSE of 3.05, which is a weaker performance than the Analytical model. Several factors contribute to the weaker performance of the data-driven model when compared to the Lagrange model:



(a) SMA Filter applied on Angular Velocity



(b) SMA Filter applied on Angular Acceleration

Figure 3.1: (a) Noisy joint-velocity data of joint no.2 in a random trajectory, and the result of moving average filtering; (b) Noisy joint-acceleration data of joint no.2 in the random trajectory, and the result of moving average filtering.

- **Noise and Inconsistencies**: The presence of noise and inconsistencies in the data can significantly impact model performance. For instance, shaking disturbances of the robot arm during data collection can introduce significant noise, leading to inaccurate measurements of joint angles, velocities, and torques.
- **Missing Data and Sensor Errors**: Missing data points or errors in sensor readings can degrade the quality of the training data, resulting in a model that struggles to generalize well to new data.
- **Limited Diversity and Quantity of Training Data**: In real-world applications, it is challenging to collect comprehensive datasets that cover all possible operational conditions and trajectories. This limited data availability can prevent the data-driven model from learning the full range of system dynamics, making it less reliable in unobserved conditions.

It is apparent that the NN models mentioned here and the analytical model each have their benefits and limitations. These factors point to the key contribution of the present thesis, showing that training data-driven models using experimental data alone might yield sub-optimal solutions, even compared to conventional analytical models. This also underscores the importance of incorporating analytical data to enhance the robustness and accuracy of data-driven models.

The experimental results demonstrate that SDNN, being a Neural Network trained on both experimental and analytical data, models the dynamics of the Kinova Gen3 Lite more effectively. This indicates the model's robustness and reliability in handling the complex dynamics of robot manipulators.

Table 3.2: Reduced Kinova Gen3 Lite Properties

Link	**Mass** (kg)	**Inertia Matrix** (kg.m^2)	**Length** (m)	**Center of Mass** (m)
1	0.9597	$\begin{pmatrix} 0.00166 & 0 & 0 \\ 0 & 0.0014 & 0 \\ 0 & 0 & 0.000895 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0.115 \end{pmatrix}$	$\begin{pmatrix} 0 & 0.0221 & 0.0994 \end{pmatrix}$
2	1.1776	$\begin{pmatrix} 0.0115 & 0 & 0 \\ 0 & 0.0113 & 0 \\ 0 & 0 & 0.00103 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0.28 \end{pmatrix}$	$\begin{pmatrix} 0.03 & -0.0483 & 0.0966 \end{pmatrix}$
3	1.907	$\begin{pmatrix} 4.35e-03 & 6.67e-05 & 1.13e-03 \\ 6.67e-05 & 3.96e-03 & 1.80e-03 \\ 1.13e-03 & 1.80e-03 & 1.73e-02 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0.48 \end{pmatrix}$	$\begin{pmatrix} 0.031 & -0.0136 & 0.186 \end{pmatrix}$

Chapter 4

Conclusion

4.1 Objectives of the Research

This thesis presented a novel approach to modeling dynamic systems by merging analytical physics models with deep neural networks through the proposed Synthesized-Data Neural Network (SDNN). The experiments with the Kinova Gen3 Lite manipulator demonstrated that the SDNN significantly improved the accuracy of dynamics predictions when compared to standalone models. By combining simulated (analytical/physics-based) data and experimental (real) data, the SDNN offered a more reliable framework for engineering applications such as precise control and operation of robots in diverse environments.

4.2 Main Contributions

The evaluation results showed that the SDNN model achieved the lowest mean square error (MSE) of 2.1442, outperforming the Lagrange analytical model (MSE of 2.8054) and the NN trained with experimental data only (MSE of 3.0521). These findings emphasized the importance of incorporating analytical data to enhance the robustness and accuracy of data-driven models.

The SDNN's ability to accurately predict joint torques for the Kinova Gen3 Lite manipulator demonstrates its robustness and reliability in handling complex robotic dynamics. The proposed method advances the understanding of system dynamics and provides a valuable tool for developing advanced robotic control strategies, thereby contributing to the fields of industrial automation and robotic manipulation.

4.3 Possible Future Work

Future work will explore the application of SDNN to systems with higher degrees of freedom and carrying out more complex tasks, further validating

its potential in broader robotic applications. In addition, some approaches in this research may be modified to perform more reliable comparisons and possibly achieve even better performance from the model. For this purpose, the following modifications can be made in the future work:

1. Construct a fully theoretical trajectory for the analytical model.

2. Asses the quality of each model using torque control.

A detailed explanation of each of these activities is provided next.

4.3.1 Constructing a fully theoretical trajectory for the analytical model

Currently, certain way-points are defined for the robot, and a trajectory -a set of joint angles and joint velocities- is generated by the robot. This trajectory, called the experimental trajectory, is then processed and fed to the analytical model. The process of gathering this trajectory and feeding it to the analytical model is shown in Figure 4.1. However, the data collected from the experiment is prone to noise which may lead to miscalculation. Therefore, constructing a fully theoretical trajectory and not relying on the data from experiments can boost the effectiveness of the analytical model.

4.3.2 Assesment of the quality of each model using torque control

Currently, the performance of each model is compared to a set of experimental data. However, due to inherent noise, experimental data does not represent a perfect trajectory. A potentially superior evaluation method involves feeding the predicted torques from each model back into the robot. The robot then follows a trajectory using open-loop torque control. The model that produces a trajectory most closely matching the initially planned trajectory is considered to have best captured the system's dynamics.

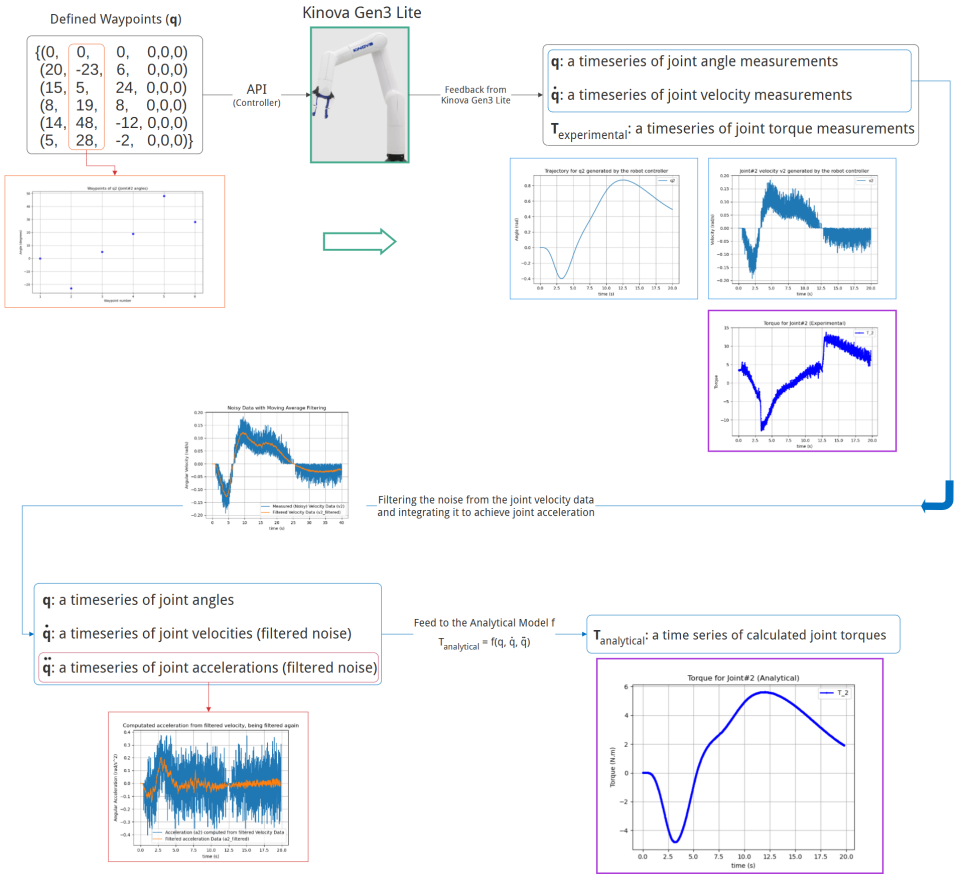


Figure 4.1: The current process of generating analytical data. The trajectory generated in the experiment is filtered and used as an input to the analytical model, resulting in analytical torque outputs.

Bibliography

[1] H. Fang, B. Xin, and X. Zhang, "Workspace-constrained optimal design of three-degrees-of-freedom parallel manipulators with minimum parasitic motions by integrating interval analysis, region mapping, and differential evolution," *Engineering Optimization*, vol. 47, no. 3, pp. 407-428, 2015, doi: 10.1080/0305215X.2014.895337.

[2] P. K. Jamwal, A. Kapsalyamov, S. Hussain, and M. H. Ghayesh, "Performance-based design optimization of an intrinsically compliant 6-DOF parallel robot," *Mechanics Based Design of Structures and Machines*, vol. 50, no. 4, pp. 1237-1252, 2022, doi: 10.1080/15397734.2020.1746669.

[3] J. N. Martínez-Castelán and M. G. Villarreal-Cervantes, "Integrated structure-control design of a bipedal robot based on passive dynamic walking," *Mathematics*, vol. 9, no. 13, p. 1482, 2021. [Online]. Available: https://doi.org/10.3390/math9131482.

[4] E. A. Alandoli, T. S. Lee, V. Vijayakumar, Y. J. Lin, and M. Q. Mohammed, "Dynamic model and integrated optimal controller of hybrid arms robot for laser contour machining," *Journal of Vibration and Control*, vol. 29, no. 13-14, pp. 3006-3024, 2023, doi: 10.1177/10775463221090000.

[5] K. S. Yakovlev, A. A. Andreychuk, J. S. Belinskaya, et al., "Safe interval path planning and flatness-based control for navigation of a mobile robot among static and dynamic obstacles," *Autom Remote Control*, vol. 83, pp. 903–918, 2022. [Online]. Available: https://doi.org/10.1134/S000511792206008X.

[6] A. Palacios, *Mathematical Modeling: A Dynamical Systems Approach to Analyze Practical Problems in STEM Disciplines*, Springer International Publishing, 2022.

[7] S. Kamalasadan and A. A. Ghandakly, "A neural network parallel adaptive controller for dynamic system control," *IEEE Transactions on In-*

strumentation and Measurement, vol. 56, no. 5, pp. 1786-1796, 2007. [Online]. Available: https://doi.org/10.1109/TIM.2007.895674.

[8] J. Blechschmidt and O. G. Ernst, "Three ways to solve partial differential equations with neural networks – A review," *GAMM-Mitteilungen*, vol. 44, no. 2, p. e202100006, 2021. [Online]. Available: https://doi.org/10.1002/gamm.202100006.

[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[10] M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink, "Hybrid deep fault detection and isolation: Combining deep neural networks and system performance models," *arXiv preprint arXiv:1908.01529*, 2019.

[11] M. Sadoughi and C. Hu, "Physics-based convolutional neural network for fault diagnosis of rolling element bearings," *IEEE Sensors Journal*, vol. 19, no. 11, pp. 4181-4192, 2019.

[12] K.-H. Lee, G. Ros, J. Li, and A. Gaidon, "Spigan: Privileged adversarial learning from simulation," *arXiv preprint arXiv:1810.03756*, 2018.

[13] C. Rackauckas, et al., "Universal differential equations for scientific machine learning," *arXiv preprint arXiv:2001.04385*, 2020.

[14] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932-3937, 2016. doi: 10.1073/pnas.1517384113.

[15] W. E, J. Han, and L. Zhang, "Integrating machine learning with physics-based modeling," *arXiv preprint arXiv:2006.02619*, 2020. [Online]. Available: https://doi.org/10.48550/arxiv.2006.02619.

[16] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238-1274, 2013.

[17] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, et al., "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[18] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, et al., "Improving image generation with better captions," *Computer Science*, vol. 2, no.

3, p. 8, 2023. [Online]. Available: https://cdn.openai.com/papers/dall-e-3.pdf.

[19] K. Saito, K. Watanabe, Y. Ushiku, and T. Harada, "Maximum classifier discrepancy for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3723-3732, 2018.

[20] M. Lutter and J. Peters, "Combining physics and deep learning to learn continuous-time dynamics models," *The International Journal of Robotics Research*, vol. 42, no. 3, pp. 83-107, 2023. [Online]. Available: https://doi.org/10.1177/02783649231169492.

[21] M. Lutter, et al., "Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning," *arXiv preprint arXiv:1907.04490*, 2019.

[22] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[23] J. Liu, P. Borja, and C. Della Santina, "Physics-Informed Neural Networks to Model and Control Robots: A Theoretical and Experimental Investigation," *Advanced Intelligent Systems*, vol. 6, no. 5, p. n/a, 2024.

Appendix A

Rotation Matrix

A.1 Definition

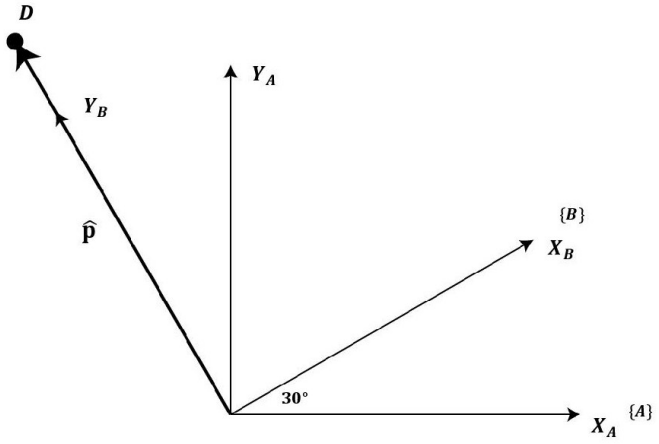


Figure A.1: Defining a vector $\hat{P}$ in two coordinate systems A and B

Figure A.1 shows two frames. Frame B results from rotating frame A in the positive Z direction. Vector $\hat{\boldsymbol{p}}$ originates from the origin and ends at point D. $\hat{\boldsymbol{p}}_A$ represents $\hat{\boldsymbol{p}}$ with respect to frame A, while $\hat{\boldsymbol{p}}_B$ represents $\hat{\boldsymbol{p}}$ with respect to frame B. The rotation matrix $\mathbf{R}_{\mathrm{A,B}} = \mathbf{R}(30°, \hat{\mathbf{k}})$ is used in,

$$\hat{\boldsymbol{p}}_A = \mathbf{R}_{\mathrm{A,B}} \hat{\boldsymbol{p}}_B \tag{A.1}$$

The rotation matrix $\mathbf{R}_{\mathrm{A,B}}$ can be interpreted in two ways:

1. $\mathbf{R}_{\mathrm{A,B}}$ rotates the vector $\hat{\boldsymbol{p}}_B$ by 30° around the vector $\hat{\mathbf{k}}$ (Z direction), while the coordinate system B remains unchanged.

2. $\mathbf{R}_{\mathrm{A,B}}$ rotates the coordinate system B by −30° around the vector $\hat{\mathbf{k}}$, while the vector $\hat{\boldsymbol{p}}_B$ remains unchanged.

In both cases, $\hat{\boldsymbol{p}}_B$ with respect to frame B ends up in the same position.

A.2 Rodrigues Rotation Formula

The Rotation matrix $\mathbf{R}$ is computed using Rodrigues' rotation formula, as given in Equation 2.15.

$$\mathbf{R}(q, \mathbf{s}) = \mathbf{I} + \sin q\mathbf{S} + (1 - \cos q)\mathbf{S}^2 \tag{2.15}$$

As Matrix $\mathbf{S}$ is the skew-symmetric matrix of the unit vector representing the axis of rotation:

$$\mathbf{S} = \begin{pmatrix} 0 & -\mathrm{s}_z & \mathrm{s}_y \\ \mathrm{s}_z & 0 & -\mathrm{s}_x \\ -\mathrm{s}_y & \mathrm{s}_x & 0 \end{pmatrix} \tag{A.2}$$

The vector $\mathbf{s} = \begin{pmatrix} \mathrm{s}_x \\ \mathrm{s}_y \\ \mathrm{s}_z \end{pmatrix}$ is the unit vector along the axis of rotation.

For example, $\mathbf{s} = \hat{\mathbf{k}} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ when the rotation is about the Z axis, resulting in:

$$\xrightarrow{\text{A.2}} \mathbf{S} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \xrightarrow{2.15} \mathbf{R}(q, \hat{\mathbf{k}}) = \begin{pmatrix} \cos q & \sin q & 0 \\ \sin q & -\cos q & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Appendix B

Lagrange Equations of Motion; Python Code

The first code presented in this section is a Python script that generates a symbolic equation of motion for the Kinova Gen3 Lite manipulator. This manipulator has its first three degrees of freedom active, while the remaining links are fixed at their zero positions. The second code is a modified version of the first, where specific values replace the symbols. The specific values are given in Table 3.2. The resulting equation from the second code uses the joint angles, joint torques, and joint accelerations of the first three joints of the Kinova Gen3 Lite robot as inputs.

B.1 Python program that computes the symbolic equations of motion for the reduced Kinova Gen3 Lite

```
import sympy as sp
import numpy as np

t = sp.symbols('t')

# Define q as a function of t
q1 = sp.Function('q_1')(t)
q2 = sp.Function('q_2')(t)
q3 = sp.Function('q_3')(t)

# Length
l1, l2, l3 = sp.symbols('l_1, l_2, l_3')

# Center of mass
c1_x, c1_y, c1_z = sp.symbols('cx_1 cy_1 cz_1')
c2_x, c2_y, c2_z = sp.symbols('cx_2 cy_2 cz_2')
c3_x, c3_y, c3_z = sp.symbols('cx_3 cy_3 cz_3')
com1 = sp.Array([0, 0, c1_z])
com2 = sp.Array([0, 0, c2_z])
com3 = sp.Array([0, 0, c3_z])

# Mass
m1, m2, m3 = sp.symbols('m_1 m_2 m_3')
g = sp.symbols('g')

# Define symbols for inertia matrix I1
I1xx, I1xy, I1xz, I1yx, I1yy, I1yz, I1zx, I1zy, I1zz = sp.symbols('I_{1xx}
I_{1xy} I_{1xz} I_{1yx} I_{1yy} I_{1yz} I_{1zx} I_{1zy} I_{1zz}')

# Create inertia matrix I1
I1 = sp.Matrix([[I1xx, I1xy, I1xz],
             [I1yx, I1yy, I1yz],
             [I1zx, I1zy, I1zz]])

# Define symbols for inertia matrix I2
```

```
I2xx, I2xy, I2xz, I2yx, I2yy, I2yz, I2zx, I2zy, I2zz = sp.symbols('I_{2xx}
I_{2xy} I_{2xz} I_{2yx} I_{2yy} I_{2yz} I_{2zx} I_{2zy} I_{2zz}')

# Create inertia matrix I2
I2 = sp.Matrix([[I2xx, I2xy, I2xz],
             [I2yx, I2yy, I2yz],
             [I2zx, I2zy, I2zz]])

# Define symbols for inertia matrix I2
I3xx, I3xy, I3xz, I3yx, I3yy, I3yz, I3zx, I3zy, I3zz = sp.symbols('I_{3xx}
I_{3xy} I_{3xz} I_{3yx} I_{3yy} I_{3yz} I_{3zx} I_{3zy} I_{3zz}')

# Create inertia matrix I3
I3 = sp.Matrix([[I3xx, I3xy, I3xz],
             [I3yx, I3yy, I3yz],
             [I3zx, I3zy, I3zz]])

# Define Rodrigues' Rotation Matrix
def R(angle, axis):
    axis = axis / np.linalg.norm(axis)
    ux = axis[0]
    uy = axis[1]
    uz = axis[2]

    if isinstance(angle, sp.Function) or isinstance(angle, sp.Mul):
        cosTheta = sp.cos(angle)
        sinTheta = sp.sin(angle)
    else:
        cosTheta = round(np.cos(angle),2)
        sinTheta = round(np.sin(angle),2)

    oneMinusCosTheta = 1 - cosTheta
    R = np.array([[cosTheta + ux**2*oneMinusCosTheta,
        ux*uy*oneMinusCosTheta - uz*sinTheta,
        ux*uz*oneMinusCosTheta + uy*sinTheta],
        [uy*ux*oneMinusCosTheta + uz*sinTheta,
        cosTheta + uy**2*oneMinusCosTheta,
        uy*uz*oneMinusCosTheta - ux*sinTheta],
        [uz*ux*oneMinusCosTheta - uy*sinTheta,
        uz*uy*oneMinusCosTheta + ux*sinTheta,
```

```
        cosTheta + uz**2*oneMinusCosTheta]])

    return R

# Calculations for the Kinetic energy K
R_1_0 = R(q1,[0,0,1])
w_1 = np.array([[0],[0],[sp.diff(q1,t)]])
R_2_1 = R(q2,[0,1,0])
R_1_2 = R(-q2,[0,1,0])
w_2 = R_1_2@w_1 + np.array([[0],[sp.diff(q2,t)],[0]])
R_3_2 = R(-q3,[0,1,0])
R_2_3 = R(q3,[0,1,0])
w_3 = R_2_3@w_2 + np.array([[0],[-sp.diff(q3,t)],[0]])

R1 = R_1_0
R2 = R_1_0@R_2_1
R3 = R_1_0@R_2_1@R_3_2

P_c1_1 = com1
v_c1 = np.cross(w_1.reshape(1,3),P_c1_1).reshape(3,1)
P_2_1 = np.array([0, 0, l1])
v_2_1 = np.cross(w_1.reshape(1,3),P_2_1).reshape(3,1)
v_2_0 = R1 @ v_2_1
P_c2_2 = com2
v_c2 = np.cross(w_2.reshape(1,3),P_c2_2).reshape(3,1)
P_3_2 =  np.array([0, 0, l2])
v_3_2 = np.cross(w_2.reshape(1,3),P_3_2).reshape(3,1)
v_3_0 = v_2_0 + R2 @ v_3_2
# P_c2_2 = np.array([lc2, 0, 0])
P_c3_3 = com3
v_c3 = np.cross(w_3.reshape(1,3),P_c3_3).reshape(3,1)

v_c1_0 = R1 @ v_c1
v_c2_0 = v_2_0 + R2 @ v_c2
v_c3_0 = v_3_0 + R3 @ v_c3

a11 = m1*sp.transpose(v_c1_0)@v_c1_0
a22 = m2*sp.transpose(v_c2_0)@v_c2_0
a33 = m3*sp.transpose(v_c3_0)@v_c3_0
```

```
b11 = w_1.T@R1@I1@R1.T@w_1
b22 = w_2.T@R2@I2@R2.T@w_2
b33 = w_3.T@R3@I3@R3.T@w_3

K1 = 1/2*(a11+b11)
K2 = 1/2*(a22+b22)
K3 = 1/2*(a33+b33)
K = K1 + K2 + K3

# Calculation for the Potential energy U
U1 = m1*g*com1[2]
U2 = m2*g*(l1 + com2[2]*sp.cos(q2))
# com1 is wrt coordinate 1 and com2 wrt coordinate 2
U3 = m3*g*(l1 + l2*sp.cos(q2) + com3[2]*sp.cos(q2-q3))
U = U1 + U2 + U3

# Calculations for the equations of motion (tau) based on Lagrange equation
dK_dq1d = sp.diff(K[0,0],sp.diff(q1))
d_dt_dk_dq1d = sp.diff(dK_dq1d,t)
dK_dq2d = sp.diff(K[0,0],sp.diff(q2))
d_dt_dk_dq2d = sp.diff(dK_dq2d,t)
dK_dq3d = sp.diff(K[0,0],sp.diff(q3))
d_dt_dk_dq3d = sp.diff(dK_dq3d,t)
d_dt__dK_dqd = np.vstack((d_dt_dk_dq1d, d_dt_dk_dq2d, d_dt_dk_dq3d))

dK_dq1 = sp.diff(K[0,0],q1)
dK_dq2 = sp.diff(K[0,0],q2)
dK_dq3 = sp.diff(K[0,0],q3)
dK_dq = np.vstack((dK_dq1, dK_dq2, dK_dq3))

dU_dq1 = sp.diff(U,q1)
dU_dq2 = sp.diff(U,q2)
dU_dq3 = sp.diff(U,q3)
dU_dq = np.vstack((dU_dq1, dU_dq2, dU_dq3))

tau_Kinova = sp.simplify(d_dt__dK_dqd - dK_dq + dU_dq)
```

B.2 Python program that computes the equations of motion for the reduced Kinova Gen3 Lite, with values instead of symbols

```
import sympy as sp
import numpy as np

t = sp.symbols('t')

# Define q as a function of t
q1 = sp.Function('q_1')(t)
q2 = sp.Function('q_2')(t)
q3 = sp.Function('q_3')(t)

# Mass
m1 = 0.9597
m2 = 1.1776
m3 = 1.907

# Length
l1 = 0.115
l2 = 0.280
l3 = 0.140+0.105+0.105+0.130

# Center of Mass
com1 = np.array([0, 0.02214, 0.09938])
com2 = np.array([0.029983, -0.0453, 0.21155]) - np.array([0,-0.003,l1])
com3 = np.array([0.03096, -0.01334, 0.5811]) - np.array([0,-0.003,l1+l2])

g = 9.81 # Free fall acceleration

# Moment of Inertia
I1 = np.diag([0.001660, 0.001404, 0.0008949])
I2 = np.diag([0.01149, 0.01133, 0.001029])
I3 = np.array([[4.34927145e-03, 6.67148531e-05, 1.13243008e-03],
 [6.67148531e-05, 3.96456682e-03, 1.79733321e-03],
 [1.13243008e-03, 1.79733321e-03, 1.73430192e-02]])

# Rodrigues' Rotation Formula
```

```
def R(angle, axis):
    axis = axis / np.linalg.norm(axis)
    ux = axis[0]
    uy = axis[1]
    uz = axis[2]

    if isinstance(angle, sp.Function) or isinstance(angle, sp.Mul):
        cosTheta = sp.cos(angle)
        sinTheta = sp.sin(angle)
    else:
        cosTheta = round(np.cos(angle),2)
        sinTheta = round(np.sin(angle),2)

    oneMinusCosTheta = 1 - cosTheta
    R = np.array([[cosTheta + ux**2*oneMinusCosTheta,
        ux*uy*oneMinusCosTheta - uz*sinTheta,
        ux*uz*oneMinusCosTheta + uy*sinTheta],
        [uy*ux*oneMinusCosTheta + uz*sinTheta,
        cosTheta + uy**2*oneMinusCosTheta,
        uy*uz*oneMinusCosTheta - ux*sinTheta],
        [uz*ux*oneMinusCosTheta - uy*sinTheta,
        uz*uy*oneMinusCosTheta + ux*sinTheta,
        cosTheta + uz**2*oneMinusCosTheta]])

    return R

# Calculations for the Kinetic energy K
R_1_0 = R(q1,[0,0,1])
w_1 = np.array([[0],[0],[sp.diff(q1,t)]])
R_2_1 = R(q2,[0,1,0])
R_1_2 = R(-q2,[0,1,0])
w_2 = R_1_2@w_1 + np.array([[0],[sp.diff(q2,t)],[0]])
R_3_2 = R(-q3,[0,1,0])
R_2_3 = R(q3,[0,1,0])
w_3 = R_2_3@w_2 + np.array([[0],[-sp.diff(q3,t)],[0]])

R1 = R_1_0
R2 = R_1_0@R_2_1
R3 = R_1_0@R_2_1@R_3_2
```

```
P_c1_1 = com1
v_c1 = np.cross(w_1.reshape(1,3),P_c1_1).reshape(3,1)
P_2_1 = np.array([0, 0, l1])
v_2_1 = np.cross(w_1.reshape(1,3),P_2_1).reshape(3,1)
v_2_0 = R1 @ v_2_1
P_c2_2 = com2
v_c2 = np.cross(w_2.reshape(1,3),P_c2_2).reshape(3,1)
P_3_2 =  np.array([0, 0, l2])
v_3_2 = np.cross(w_2.reshape(1,3),P_3_2).reshape(3,1)
v_3_0 = v_2_0 + R2 @ v_3_2
# P_c2_2 = np.array([lc2, 0, 0])
P_c3_3 = com3
v_c3 = np.cross(w_3.reshape(1,3),P_c3_3).reshape(3,1)

v_c1_0 = R1 @ v_c1
v_c2_0 = v_2_0 + R2 @ v_c2
v_c3_0 = v_3_0 + R3 @ v_c3

a11 = m1*sp.transpose(v_c1_0)@v_c1_0
a22 = m2*sp.transpose(v_c2_0)@v_c2_0
a33 = m3*sp.transpose(v_c3_0)@v_c3_0

b11 = w_1.T@R1@I1@R1.T@w_1
b22 = w_2.T@R2@I2@R2.T@w_2
b33 = w_3.T@R3@I3@R3.T@w_3

K1 = 1/2*(a11+b11)
K2 = 1/2*(a22+b22)
K3 = 1/2*(a33+b33)
K = K1 + K2 + K3

# Calculations for the Potential energy
U1 = m1*g*com1[2]
U2 = m2*g*(l1 + com2[2]*sp.cos(q2))
# com1 is wrt coordinate 1 and com2 wrt coordinate 2
U3 = m3*g*(l1 + l2*sp.cos(q2) + com3[2]*sp.cos(q2-q3))
U = U1 + U2 + U3

dK_dq1d = sp.diff(K[0,0],sp.diff(q1))
d_dt_dk_dq1d = sp.diff(dK_dq1d,t)
```

```
dK_dq2d = sp.diff(K[0,0],sp.diff(q2))
d_dt_dk_dq2d = sp.diff(dK_dq2d,t)
dK_dq3d = sp.diff(K[0,0],sp.diff(q3))
d_dt_dk_dq3d = sp.diff(dK_dq3d,t)
d_dt__dK_dqd = np.vstack((d_dt_dk_dq1d, d_dt_dk_dq2d, d_dt_dk_dq3d))

dK_dq1 = sp.diff(K[0,0],q1)
dK_dq2 = sp.diff(K[0,0],q2)
dK_dq3 = sp.diff(K[0,0],q3)
dK_dq = np.vstack((dK_dq1, dK_dq2, dK_dq3))

dU_dq1 = sp.diff(U,q1)
dU_dq2 = sp.diff(U,q2)
dU_dq3 = sp.diff(U,q3)
dU_dq = np.vstack((dU_dq1, dU_dq2, dU_dq3))

tau_Kinova = sp.simplify(d_dt__dK_dqd - dK_dq + dU_dq)
```

The result of this code is $\boldsymbol{\tau}_{\text{Kinova}}$, which is a function of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$.