Dynamically feasible data-driven trajectory generation for high performance control of robot manipulators

Liliana Barbulescu

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Dynamically feasible data-driven trajectory generation for high performance control of robot manipulators

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Liliana Barbulescu

April 27, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Airborne under the supervision of SAM|XL. Their cooperation is hereby gratefully acknowledged.

Abstract

Robot manipulators are significantly more accurate than their human counterparts and enhance the repeatability of various tasks. However, manufactures still provide reduced information regarding the robot controller functions, typically affecting robots' predictability. Additionally, industrial examples show that system transparency could be improved, helping users to understand the process and apply corrective actions. One step in the direction of improved predictability is the identification of the Limiter block associated with the KUKA robot controllers that limit the acceleration and jerk obtained during motions. This work aims to provide dynamically feasible trajectories with reduced performance deterioration. Specifically, the Limiter's behaviour was analyzed and later predicted in a iterative manner, aiming for the maximum velocity, acceleration and jerk that can be achieved, starting from the current robot state and given specific robot information. Using the predicted bounds, the trajectory generator will determine the maximum achievable point that fulfills all the constraints, running an optimization problem in an iterative fashion. Nonetheless, the practical experiments represent a significant component allowing data collection and results validation.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1-1 Background

Over the span of years, the manufacturing process benefited from the technology progress in the form of automated manufacturing. At its core, the automated manufacturing relies on control systems governed by computers to conduct and supervise the production processes, meaning that in an automatic facility, the tools and technologies are interconnected [6]. Although the humans are still required for the programming and management tasks, fewer humans are involved in the actual production processes. This permits an increase in productivity, less defects and safer manufacturing facilities.

On a high level, the automation contributed to the society's development by providing improved control and consistency as a result of higher production rates, efficient use of materials and increased quality of products. From an individual perspective, the automation brought a higher standard of living by improving the work safety in factories, reducing the workweek and decreasing the manufacturer's lead time [7]. However, the automation might imply several disadvantages such as workers displacement, decrease in the flexibility of the production strategy or a high capital required for the initial investment and further maintenance.

According to [7], the automated manufacturing processes can be split into three groups: fixed, programmable and flexible automation. Fixed automation demonstrates the smallest flexibility using machines that are specifically designed for a given task. An example of fixed automation is represented by automated assembly machines. Programmable automation implies the usage of machines that can be programmed to execute tasks and presents higher adjustment capabilities. As a result, the industrial robots might easily serve the requirements in this category, while the most well-know example of programmable automation is represented by the Computer Numerical Control (CNC) tool. Lastly, in the flexible automation setup, the manufacturing process can be easily adjusted to a slightly different product. Subsequently, multiple industries recently adapted their manufacturing processes to produce smaller batches of customized products. For example, according to [8], the car manufacturing industry started to produce customized cars, using a flexible automation approach that covers painting, welding or assembly.

Industrial robots are widely used in the manufacturing processes for two reasons. First, they are repetitive and accurate, decreasing the production defects. Second, they can replace humans in dangerous tasks or hazardous environments such as handling heavy tools or dealing with harmful fumes produced by different types of paint.

According to [9], the applications industrial robots are mostly involved in can be split into three categories: material handling, processing operations and assembly and inspection. In material handling, robots typically move materials or work parts from one location to another. This mostly implies the execution of pick and place operations and the usage of specifically designed grippers. Processing operations involve processes effectuated on a specific work part, using a designed tool, to execute a very specific task. Some examples of processing operations might be welding or painting. Assembly and inspection is sometimes accomplished by robots to guarantee product requirements being fulfilled.

Out of all industrial robots used in manufacturing, robot manipulators gained popularity over the years. According to [10] a robot manipulator consists of a sequence of rigid bodies, called links, interconnected by means of articulations, called joints. They are characterized by an arm ensuring mobility, a wrist conferring dexterity and an end-effector performing the task. The number of axes in a robot manipulator typically varies between 2 and 10, but commonly, 6-axes robot manipulators are used as a result of their similarity to the human arm that enhances intuitiveness. Moreover, based on the different joint types, robot manipulators can also be further differentiated.

Being relatively flexible systems that can be programmed to deal with various tasks by moving the end-effector on specific trajectories, robot manipulators are involved in multiple manufacturing industries. As a result, one of the growing fields, the aerospace industry, adopted robot manipulators in the manufacturing processes to replace more manual tasks. In the case of airplanes, robot manipulators can cover various functions regarding the airplane structure itself such as drilling and fastening, sealing, inspection (both during production and periodic inspections), welding, painting, sanding or washing.

On the other hand, thanks to their accuracy, robot manipulators are also involved in the composite manufacturing process helping in various automatic deposition operations such as Automated Tape Layering (ATL) or Automated Fiber Placement (AFP). This is directly related to the production of airplanes since composite materials are highly used in various parts of the aircraft such as in their fuselage structure. Composites represent rather recently researched materials, aggregating two or more distinctive elements, resulting into improved material properties such as lower mass, high-stiffness and enhanced durability. As an example, two long-range airplanes, Boeing 787 and Airbus A350XWB, already contain more than half of their mass in composites.

1-2 Motivation

Although robot manipulators present a higher degree of predictability and accuracy compared to their human counterparts, there is still room for improvement. One example in this sense, comes precisely from the aerospace industry and is represented by FAUB (Fuselage Automated Upright Build), a project of Boeing. This process aimed to avoid the rather complex and tedious task of rotating the airplanes fuselage when attaching the exterior metal panels by

using robots and keeping the fuselage in an upright position. In this case, two pairs of robots had to synchronously work from both the outside and inside of the fuselage to drill holes and insert fasteners for fixing the metal panels. Unfortunately, the entire project failed since the pair of inside-outside robots could not properly synchronize. A picture depicting a pair of robots working in the FAUB project is presented in Figure 1-1. This loss could have been avoided if the robots' behaviour was more transparent to the user.

Figure 1-1: A pair of robots working on an airplane's fuselage in the Boeing FAUB project. [1]

Additionally, time represents a crucial aspect in the airplanes production processes. Considering the high level of safety required by the aerospace domain, the extensive tests necessary for a new airplane to reach serial production phase typically last for prolonged times. Every additional modification demands tedious testing and while the relevance of the tests is undeniable, considerable efforts were recently undergone to reduce the analysis duration. This could also benefit from improved predictability to ensure that a slight change in the process will not cause a major effect in the final product.

In this sense, the limited knowledge and prediction capabilities of the user upon the manipulator's trajectory generation and internal states does not contribute to improving the production process. The current status indicates that the robot users might achieve increased predictability in the robot manipulator processes by understanding the functions of the robot controller. More specifically, by determining a set of attainable operations or feasible trajectories, corrective actions could be identified and applied.

1-3 Problem statement

In the context of rather reduced available information on the robot inner processes, some important mentions should be stated before defining the research problem tackled in the current work. Although this might apply to multiple robot manufactures, the current work will cover only KUKA robots and their provided documentation.

In the case of KUKA robot systems, the manufacturer does not provide any information about the robot controller and its functionality. Additionally, KUKA only permits position corrections to be provided to the robot, but with the lack of information about the robot controller, one cannot predict the effect of its actions. According to the literature and robot forums, various trials to provide improved commands to KUKA robots were performed. However, the typical procedure implied bypassing the robot controller but this is highly discouraged since it might result into various safety issues.

In a typical usage of the robot, the robot controller will compute and execute a trajectory given the start and target positions and tool information. For protecting the robot from overtime deterioration, KUKA imposed several damage reduction mechanisms. Empirically, when some of the limits enforced in order to reduce potential damage are exceeded, the robot will reach a controlled stop meant to halt the trajectory execution.

One of the damage reduction mechanisms imposed by KUKA is represented by a Limiter block characterized by velocity, acceleration and jerk limitations depending on various motion parameters. According to their documentation, KUKA supports its existence but provides no details about its implementation. According to to robots experiments, the Limiter block will permit the achievement of the rated maximum velocity during all motions, but might increase the acceleration and deceleration time as a result of limits imposed on acceleration and jerk.

As a result, the current work does not aim to provide an improved control method for the KUKA robots, but to use the given robot information in a data-driven approach to provide input commands to the robot, without affecting the robot system. Thus, the main research question can be formulated as follows:

How to improve the predictability of robot manipulators without significant performance deterioration, by only providing more educated positional commands?

To facilitate the generation of more educated commands, the main question can be split into three main challenges: the motion parameters causing the most noticeable Limiter's effect, the relation between the input parameters and the velocity, acceleration and jerk limits and the generation of position samples that the robot can follow. As a result, these aspects can be formulated as three research topics:

1. **"Which motion parameters play the most significant role in the robot controller limits?"**
2. **"How can the function defining the relation between the motion parameters and the limits be described?"**
3. **"Given the function defining the relation between the motion parameters and the limits, how to generate dynamically feasible trajectories?"**

By answering to all the research questions, this work aims to create dynamically feasible, minimum-time trajectories between the start and target positions, given specific tool information. The proposed solution will aid robot manipulators users to improve the predictability in their work by providing more transparent motions.

1-4 Thesis outline

In Chapter 2, a short description of the robot system is provided. This includes information about robot communication approaches, time-derivation methods that allow the computation of velocity, acceleration and jerk profiles using previously collected position data and an initial analysis regarding the Limiter's effect. Chapter 3 includes the procedure proposed for identifying the Limiter's effect as velocity, acceleration and jerk bounds. Chapter 4 covers the trajectory generation approach aiming to determine dynamically feasible position commands to be closely followed by the robot. Moreover, Chapter 5 describes the final solution aggregating the trajectory generation and bounds prediction. Finally, Chapter 6 concludes the work and presents future research ideas.

Chapter 2

Environment setup

Since the solution to the problem tackled in this work is mostly designated for KUKA robots, all the experiments were performed on a KUKA KR 210 R2700 robot. As a result, the robot environment will be described and the KUKA communication interfaces will be introduced, also covering the position recording and playback procedures. Furthermore, the custom experimental setup used for adjusting tool parameters will be characterized.

Moreover, since KUKA only permits the recording of position measurements, several time-derivation methods will be introduced. These are used for retrieving the velocity, acceleration and jerk profiles required for observing the Limiter's effect. After the position derivatives are computed, an initial analysis is provided to check its effect on the robot's behaviour and to determine the robot parameters that significantly influence the Limiter's effect. The results of the analysis are particularly important for deciding the requirements for an extended and representative dataset.

2-1 Robot setup

2-1-1 KUKA KR 210 R2700 extra

The KUKA KR 210 R2700 extra is a 6-axis robot manipulator. The robot axes and their direction of rotation are presented in Figure 2-1 and its principal components are presented in Figure 2-2. In parallel to the latter, a short description of these main assemblies is provided in Table 2-1. Furthermore, the robot design parameters that describe the maximum capabilities are provided in the robot model name itself. In this sense, one can deduce the rated payload of 210 kg and the working radius of 2700 mm. Additionally, to extend the number of DoF of the system to seven, the robot manipulator is mounted on a linear track.

According to the KUKA documentation [3, 5], the software-limited range of motion and the maximum speed in the presence of payload for each axis are presented in Table 2-2. On the other hand, no further limits on the higher time-derivatives of position such as acceleration or jerk are mentioned in the provided manuals.

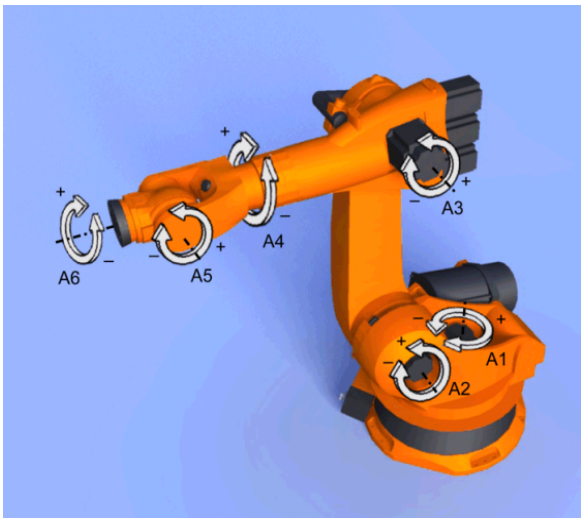

Figure 2-1: Robot axes of KUKA KR 210 R2700 extra. [2]

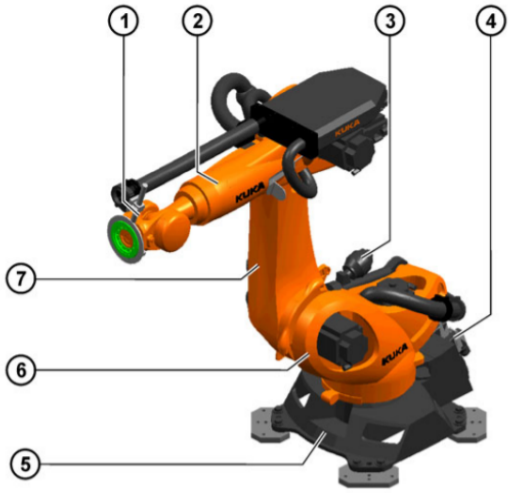

Figure 2-2: Main components of the KUKA KR 210 R2700
1-In-line wrist, 2-Arm, 3-Counterbalancing system, 4-Electrical instalations, 5-Base frame, 6-Rotating column, 7-Link arm. [3]

Figure marker	Component name	Description
1	In-line wrist	It is composed by axes 4, 5 and 6. It is driven directly by the motor of axis 6 which is placed in the wrist itself and via connecting shafts from the arm's rear by motors of axes 4 and 5.
2	Arm	It is composed by the link between the in-line wrist and the link arm. It is driven by the motor of axis 3.
3	Counterbalancing system	It is placed between the rotating column and the link arm. It minimizes the moments generated about axis 2
4	Electrical installations	It contains the motor and data cables of the motors for axes 1 to 6.
5	Base frame	It is the actual robot base that can be screwed to the mounting base. It also holds the motor interface, data cable and energy supply system.
6	Rotating column	It holds the motors of axes 1 and 2. It performs the rotational motion of axis 1
7	Link arm	It is placed between the arm and the rotating column. It consists of the link arm body together with the buffers for axis 2.

Table 2-1: Main components description of KUKA KR 210 R2700 extra. [3]

Axis	Motion range, software-limited	Speed with rated payload
1	+/-185°	123°/s
2	-5° to -140°	115°/s
3	+155° to -120°	112°/s
4	+/-350°	179°/s
5	+/-125°	172°/s
6	+/-350°	219°/s
7 (Linear unit)	0 to 4 m	1.96 m/s

Table 2-2: Axes data for KUKA KR 210 R2700. [3] and [5]

Moreover, to coordinate the manipulator movements, an external controller (relative to the robot, but with no user access) - KUKA KR C4 is used. The main goal of the robot controller is to receive the desired position goal as an external command and to compute the optimal trajectory that will achieve the target position. According to the KUKA documentation, the robot controller is also responsible for applying various safety mechanisms when generating the trajectory. One example of a safety mechanism is represented by the dynamical limitations imposed on acceleration and jerk when generating a trajectory. In addition, when the robot receives external intermediary points but the trajectory fails to respect the limits, the robot controller will enforce a controlled stop, meaning that the robot will abort the execution of a trajectory and will stop in the shortest time possible.

An illustration of the entire robot system (excluding the linear unit), required for actuating the robot is presented in Figure 2-3. This consists of the robot manipulator itself, the robot controller, the cables connecting all the components and a teach pendant. The latter is a valuable tool used for specifying various robot settings and for jogging it. Furthermore, the robot cell used during this work, including the manipulator and the linear unit, made available by SAM|XL with the help of Airborne, is displayed in Figure 2-4.

2-1-2 KUKA communication interfaces

The robot controller was equipped with the KUKA System Software (KSS) which is responsible for the basic control functions of the robot such as path planning, I/O management, data and file management etc. Using KSS, there exist multiple possibilities for robot jogging including manual or automated processes. Using the teach pendant, the robot user can quickly jog the robot given the existing buttons. To increase the repeatability of the jogging procedure, KUKA provides the KUKA Robot Language (KRL) which allows the definition of short scripts that can be run either manually or automatically through the teach pendant.

When writing a simple KRL motion script, the user can provide the position end goal and the motion type. It is important to mention that a script can also contain multiple commands. As a result, the robot will execute every one of them reaching all intermediary points. The motion types that can be programmed are Point-to-point motions (PTP) (fastest path to the end point), Linear motions (LIN) (a straight path to the end point), Circular motions (CIRC) (a circular path to the end point) and Spline motions (complex, curved paths).

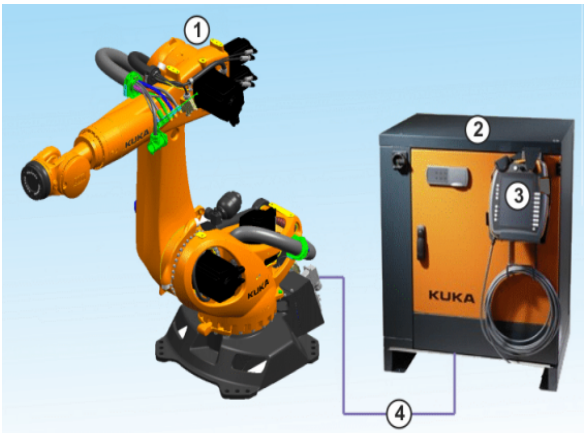

Figure 2-3: Robot system components. 1 - Manipulator, 2 - Robot controller, 3 - Teach pendant, 4 - Connecting cables. [4]

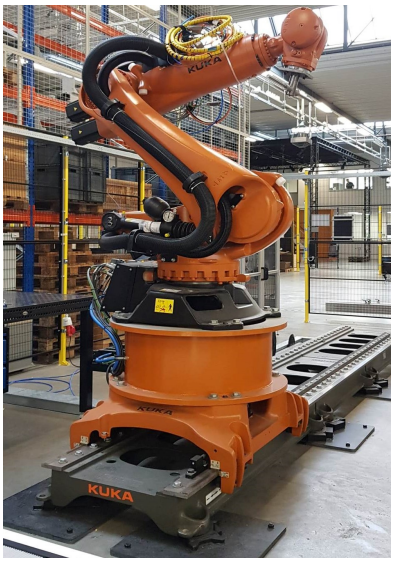

Figure 2-4: Robot cell used for experiments

To allow the communication between the robot controller and external systems or the monitoring of various signals of interest, the KUKA Robot Sensor Interface (RSI) tool was provided. Its main functionality in the current context is the communication between the robot controller and an external computer, defined as a sensor system. In this project, RSI is used both for its data exchange and sensor correction possibilities. In this sense, the data exchange via Ethernet permitted the robot user to collect data regarding the current robot position in real time, with a sampling frequency up to 250 Hz. Furthermore, the correction capabilities provided by RSI allowed an external computer to provide position commands in the form of sensor corrections. This was achieved by using the *RSI_MOVECORR* command that achieves a purely sensor-guided correction of the robot. Moreover, by specifying the argument of the command *RSI_ON* to be *#ABSOLUTE*, a correction mode relative to the initial robot position was chosen.

Robot position recording

As mentioned above, KUKA only permits the collection of position data (and not its derivatives) and this is facilitated by RSI. The procedure for actuating the robot using the robot controller and collecting the position points it reaches during the trajectory is detailed below.

1. The desired motion was defined using a KRL script. The script typically contained two PTP motions to actuate the robot from a start position to an end goal and back.

2. Various tool related parameters were specified in the corresponding teach pendant menu. Moreover, a speed level was chosen.

3. The KRL script was ran using the automated functionality of the teach pendant.

4. Since the KRL script contains the required commands, RSI facilitated the position data collection through an Ethernet connection via the User Datagram Protocol/Internet Protocol (UDP/IP).

5. Data is captured using the Wireshark software, being first saved in the Extensible Markup Language (XML) format that RSI provides and later converted to Comma-Separated values (CSV) for easier data manipulation.

Robot position playback

Additional to the position recording procedure, an important part of the KUKA experiments is represented by the ability to provide position commands to the robot. This should be the last step of the solution in which the generated trajectory is provided to the robot to be executed. In order to implement this functionality, a specific procedure was developed and its steps will be further described.

1. The position commands for all the joints should be stored in a .csv file. Since the used sampling time is 4ms, the change between two consecutive position points should be feasible in 4ms, in order to avoid a controlled stop.

2. A script using Python will convert the information in the CSV file into a format that the robot can interpret as position commands.

3. A solution implemented in Robot Operating System (ROS) will make sure that the position commands can be sent using RSI through UDP to the robot.

4. In order for the robot to receive the position commands sent through UDP, a KRL program will be executed using the Teach Pendant. The main goal of this program is to make the robot wait in the *MOVE_CORR()* command that permits pure sensor-guided corrections.

5. Lastly, when the Python script is ran, the position commands will be fed to the robot and it will start its movement.

After the procedure above is used to provide data to the robot, the same approach presented for position recording can be used to check the exact motion executed by the robot given the specific position inputs.

It is important to mention that when using both communication directions, the Wireshark file obtained will include information about the position commands the robot receives and the motion executed by the robot. This way, the latency between receiving and executing a position command was identified to 7 samples (28 milliseconds) in most of the cases.

Nevertheless, slight differences were observed between the position commands received and the trajectory executed by the robot. Some samples serving as an example in this sense can be observed in the first two columns of Table 2-3. It is important to mention that when the input and output samples were aligned, the minimum error between input and recording was favored.

In addition to the error between input and output, experiments revealed slight differences between the samples of different runs using the same input trajectory. The differences between multiple recordings reveal the non-deterministic behavior of the playback procedure and an example is provided in Table 2-3.

Input trajectory	Recording no.1	Recording no.2	Recording no.3	Recording no.4
-134.506429	-134.502402	-134.503756	-134.502893	-134.501994
-134.456447	-134.454325	-134.455626	-134.454733	-134.453957
-134.403076	-134.402624	-134.403879	-134.402980	-134.402315
-134.346193	-134.347528	-134.348759	-134.347872	-134.347207
-134.285679	-134.289496	-134.290727	-134.289806	-134.289222
-134.221417	-134.227993	-134.229254	-134.228297	-134.227760
-134.153292	-134.163007	-134.164244	-134.163217	-134.162715
-134.081198	-134.094724	-134.095938	-134.094952	-134.094479
-134.005029	-134.022521	-134.023722	-134.022812	-134.022392
-133.924683	-133.946273	-133.947411	-133.946577	-133.946215

Table 2-3: Comparison between input and recordings samples

Moreover, during experiments, the non-determinism was also displayed in the form of failed executions. More specifically, several consecutive experiments were required to obtain a successful execution. Although the robot received the same input trajectory, multiple experiments led to a controlled stop while some obtained a successfully executed motion. This behavior is depicted in Figure 2-5 that shows the robot input trajectory and multiple recordings of the motion executed by the robot. One can observe that seven out of the eight recordings triggered a control stop. Moreover, the controlled stop occurs at different points on the trajectory. The moment a trajectory is interrupted is marked by both a dot along the trajectory and an x symbol, in the corresponding color, at the bottom of the Y-axis. It is important to mention that the Recording no.1 reached the target position specified by the input trajectory and stands for the successful example. Moreover, the order in which the trajectories are provided in the figure, does not match the real experiments order.

It is important to mention that the recordings mentioned in Table 2-3 do not correspond to the recordings displayed in Figure 2-5, so the numbers should not be matched.

Empirically, no conclusion could be drawn in order to determine which experiments fail. Further research on the playback method might reveal reasonable explanation on the controlled stops obtained by some trajectories, but this did not represent the scope of the current work. However, the non-determinism observed in the playback method is worth mentioning because multiple failures of an experiment might indicate an unfeasible trajectory used as position commands, while the experiments provided in Figure 2-5 prove that this was not the case for the given experiment. As a result, one might conclude that multiple experiments are needed to ensure that the controlled stop triggered is not the result of an unfeasible trajectory but of the non-determinism of the playback method. Of course, a reasonable number of trials should be executed to prevent robot damage.

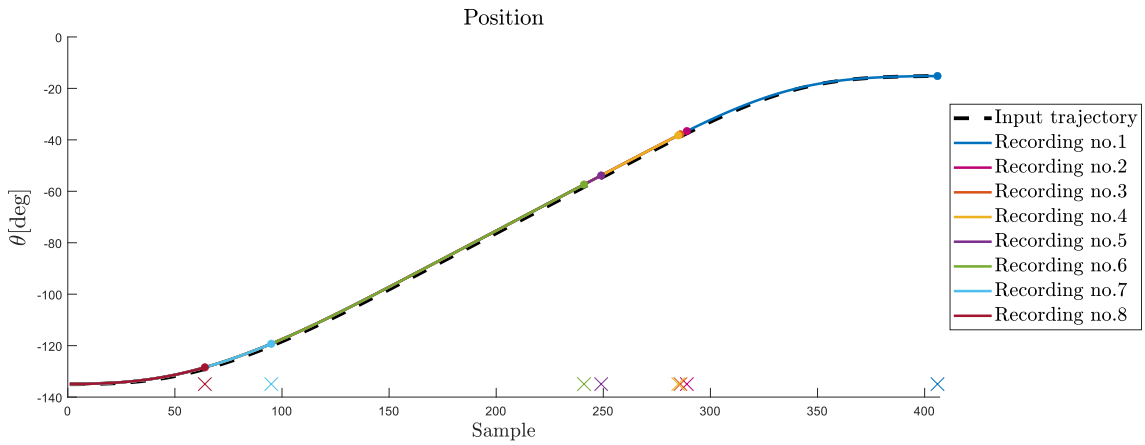

Figure 2-5: Robot joint: A2, Experiment: 70kg, robot pose: 'conf1'
Multiple recordings from the same input trajectory

2-1-3 Experiments design

In order to facilitate the variation of tool parameters that might play a significant role in the generation of the dynamical bounds, a custom tool was attached to the robot. The tool is similar to the barbell systems used in weightlifting competitions, permitting the attachment of metal plates to the main bar. This system allows tool weight adjustment and the modification of the payload Centre of Mass (CoM) relative to the robot flange. The custom tool attached to the KUKA robot is presented in Figure 2-6.

Figure 2-6: Custom barbell tool mounted on the KUKA robot

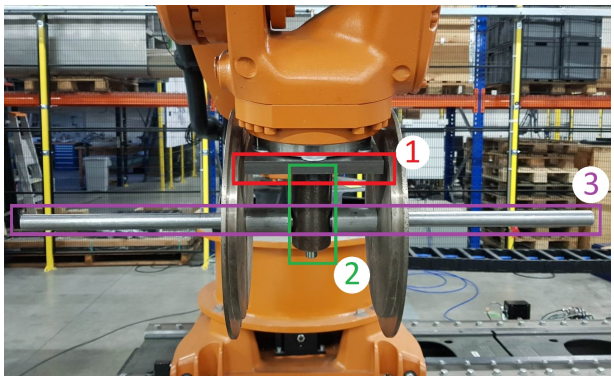

Figure 2-7: Barbell components. 1 - Plate, 2 - Cylinder, 3 - Bar

For a safe actuation of the robot, KUKA requires the robot user to provide information of the tool attached to the robot wrist in a specific menu of the teach pendant. In this sense, information about the payload weight, the position of the payload CoM and the inertia terms of the payload relative to the robot flange are necessary. While the payload weight and its CoM only require measurement, the inertia terms were computed according to (2-1), using

the barbell inertia terms defined in (2-2). The barbell parameters related to the plate, the cylinder and the bar depicted in Figure 2-7 by 1, 2 and respectively, 3 are defined in Table 2-4. Since all composing parameters are constant, the barbell inertia parameters, computed using (2-2), can be considered constant as $I_{xbb} = 0.092$, $I_{ybb} = 0.03$, $I_{zbb} = 0.073$.

Considering symmetrical placing of the metal plates on each side of the barbell, m_w depicts the plates mass on each side and t_w denotes the total width of the disks on each side. Further, $r = 0.08 + \frac{1}{2}t_w$ represents the distance along the barbell on each side, from its center to the center of the weight block. The 0.08m offset in the r parameter is caused by construction related aspects of the barbell. More specifically, the bar requires a gap between its center and the first metal disk. Moreover, the distance r includes $\frac{1}{2}t_w$ since the weight center is considered the center of the weights block, leading to half of the total width of the disks on each side.

$$I_x = I_{xbb} + m_w(0.034 + \frac{1}{6}{t_w}^2 + 2r^2) \tag{2-1a}$$

$$I_y = I_{ybb} + 0.0438m_w \tag{2-1b}$$

$$I_z = I_{zbb} + m_w(0.0098 + \frac{1}{6}{t_w}^2 + 2r^2) \tag{2-1c}$$

$$I_{xbb} = \frac{1}{2}m_b(3a_b^2 + L_b^2) + m_b h^2 + \frac{1}{3}m_c L_c^2 + \frac{1}{12}m_p(c_p^2 + l_p^2) \tag{2-2a}$$

$$I_{ybb} = \frac{1}{2}m_b a_b^2 + m_b h^2 + \frac{1}{3}m_c L_c^2 + \frac{1}{12}m_p(c_p^2 + l_p^2) \tag{2-2b}$$

$$I_{zbb} = \frac{1}{2}m_b(3a_b^2 + L_b^2) + \frac{1}{2}m_c a_c^2 + \frac{1}{12}m_p(2l_p^2) \tag{2-2c}$$

Parameter	Meaning	Value
m_p	Plate mass	3 kg
m_b	Bar mass	1.5 kg
m_c	Cylinder mass	1.5 kg
h	Distance from robot flange CoM to the bar	0.11 m
a_b	Bar radius	0.0 1m
L_b	Bar length	0.7 m
a_c	Cylinder radius	0.02 5m
L_c	Cylinder length	0.12 m
l_p	Plate edge (of the base of the parallelepiped)	0.15m
c_p	Plate height	0.015 m

Table 2-4: Barbell parameters for the components depicted in Figure 2-7 as Plate, Cylinder, Bar

When varying the payload CoM relative to the robot flange, a constant payload of 20 kg was considered, being composed by two disks of 10 kg on each side. This set the parameters m_w = 10 kg and t_w = 0.05 m. On the other hand, the r parameter was varied up to 30 cm by

adding an additional offset ($r = 0.08 + d_o + \frac{1}{2}t_w = 0.105 + d_o$, where d_o depicts the distance offset). The variation of the r parameter was limited by the length of the main bar of the custom barbell setup.

In order to isolate the behavior of each joint, only one joint was actuated during each experiment, while all the others were kept still. This decision was mostly encouraged by the robot property that limits the speed of all the other actuated joints to the slowest one. As a result, when multiple joints are actuated, all joints reach their final destination at the same time.

Since a different configuration of the non-actuated joints will lead to different mechanical properties that could influence the dynamical bounds, several combinations of the non-actuated joints were defined, being called robot poses. Some examples of such poses will be provided further to illustrate the differences between them. In this sense, Table 2-5 and Table 2-6 show different positions of the non-actuated joints composing the robot poses for joints A1 and A2. Moreover, for all other joint from A3 to E1, the robot poses definitions can be found in A-1.

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
A1 "transp"	-184 to 184	-139	125	-180	-120	30	-4000
A1 "shorter"	-90 to 90	-139	125	-180	-120	30	-4000
A1 "conf1"	-180 to 180	-70	110	-180	-50	30	-4000
A1 "conf2"	-90 to 90	75	40	-180	-25	30	-4000

Table 2-5: Robot joint: A1, Position of all joints for multiple robot poses.

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
A2 "transp"	0	-135 to -20	130	-180	-120	30	-4000
A2 "shorter"	0	-80 to -20	130	-180	-120	30	-4000
A2 "conf1"	0	-135 to -20	48	-180	-35	30	-4000
A2 "conf2"	20	-90 to -10	30	-180	-110	30	-4000

Table 2-6: Robot joint: A2, Position of all joints for multiple robot poses.

The name "transp" was given to one robot pose that is similar to the robot transport configuration. This is the pose which robots are typically transported in and has the main benefit of reducing the influence of mechanical properties over the joints by keeping the robot in a configuration close to its center of mass. Moreover, Figure 2-8 and Figure 2-9 show the KUKA robot in two of the robot poses described in Table 2-5, where the red arrows indicate the movement direction.

Figure 2-8: KUKA robot in robot pose A1 "transp"

Figure 2-9: KUKA robot in robot pose A1 "conf1"

2-2 Time-derivation

As the KUKA robots provide only position data and of the KUKA Limiter block impacting higher-order derivatives, the solution required time-derivation of position. This aided in retrieving the velocity, acceleration and jerk profiles resulting from the position measurements. In the search for a suitable time-derivation method that will provide realistic velocity, acceleration and jerk profiles, three approaches were tested. These approaches include Backward Finite Differences (BFD) or Forward Finite Differences (FFD) with a Butterworth filter and Kalman filtering. While the Finite Differences (FD) approaches imply a rather simple implementation, a Kalman filter is more elaborate method and requires the definition of a model.

Backward Finite Differences with Butterworth filter

The velocity, acceleration and jerk profiles are retrieved using the BFD formulas depicted in (2-3). In order to smooth the approximations, a Butterworth filter (2-4) is applied after each derivation operation.

$$v_k = \frac{\theta_k - \theta_{k-1}}{\Delta t} \tag{2-3a}$$

$$a_k = \frac{v_k - v_{k-1}}{\Delta t} \tag{2-3b}$$

$$j_k = \frac{a_k - a_{k-1}}{\Delta t} \tag{2-3c}$$

$$B(f, f_0) = \frac{1}{1 + (\frac{f}{f_0})^{2m}} \tag{2-4}$$

In the formulas above, θ denotes the position, v the velocity, a the acceleration and j the jerk. Moreover, $k-1$ and k denote two consecutive time samples and Δt the sampling time. For the Butterworth filter in (2-4), f denotes the frequency, f_0 represents the cut-off frequency while m is the filter's order.

Forward Finite Differences with Butterworth filter

As anticipated, the FFD formulation is very similar to the BFD formulation. In this sense, the velocity, acceleration and jerk profiles are computed using (2-5) and the Butterworth filter in (2-4) is applied after each derivation to smoothen the approximations.

$$v_k = \frac{\theta_{k+1} - \theta_k}{\Delta t} \tag{2-5a}$$

$$a_k = \frac{v_{k+1} - v_k}{\Delta t} \tag{2-5b}$$

$$j_k = \frac{a_{k+1} - a_k}{\Delta t} \tag{2-5c}$$

Kalman filter

A Kalman filter addresses the problem of state estimation of a system defined by its difference equations. In the current work, an autonomous system dynamics is assumed and the process and measurements noises are ignored. Thus, the system dynamics is defined according to (2-6), where x denotes the state and y is the system output. For the time-derivative relation between position, velocity, acceleration and jerk, a fourth order integrator was chosen as the system dynamics. Since KUKA allows only the measurement of position information, the C matrix will be the unit vector defined below which ensures that $y_k = \theta_k$. In this sense, A and C represent the system matrices.

$$x_{k+1} = Ax_k \tag{2-6a}$$

$$y_k = Cx_k \tag{2-6b}$$

where

$$x_k = \begin{bmatrix} \theta_k \\ v_k \\ a_k \\ j_k \end{bmatrix}, \quad A = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & \frac{\Delta t^3}{6} \\ 0 & 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

Using the system defined in (2-6), the state estimation will be determined using (2-7) where K denotes the Kalman gain and $\hat{x}_h$ represents the state estimation. This procedure ensures the estimation of the position's time-derivatives.

$$\hat{x}_{k+1} = (A - KC)\hat{x}_k + Ky_k \tag{2-7}$$

Comparison of time-derivation methods

When comparing these approaches, no functional differences can be identified between FFD and BFD formulations. On the other hand, it is expected for the Kalman filter to produce distinctive results from the Finite Differences methods. For analyzing the benefits and drawbacks of each method, a simulation test was prepared. More specifically, the three methods were applied on position measurements retrieved from the KUKA robot and the computed velocity, acceleration and jerk profiles were compared.

Figure 2-10 provides a comparison between the three time-derivation methods. In this plot, the position measurements collected while actuating only the joint A1 of the robot was used. One can observe no significant differences between the velocity and acceleration profiles retrieved by the Finite Differences methods. On the other hand, the first samples of the jerk profiles computed using FFD and BFD differ significantly. While FFD provides a higher start, the BFD starts in 0 $^{\circ}/s^3$.

Further, the profiles estimated using the Kalman filter are significantly different from the Finite Differences method. Apart from the maximum values, the Kalman filter provides a noticeable phase-lag especially in the acceleration and jerk profile. In order for the profiles to be aligned, when the position reaches its maximum value all derivatives should be 0 since this position corresponds to an intermediary stop in the motion. As it can be observed in Figure 2-10, the Kalman filter profiles present a misalignment, while the Finite Differences profiles do not.

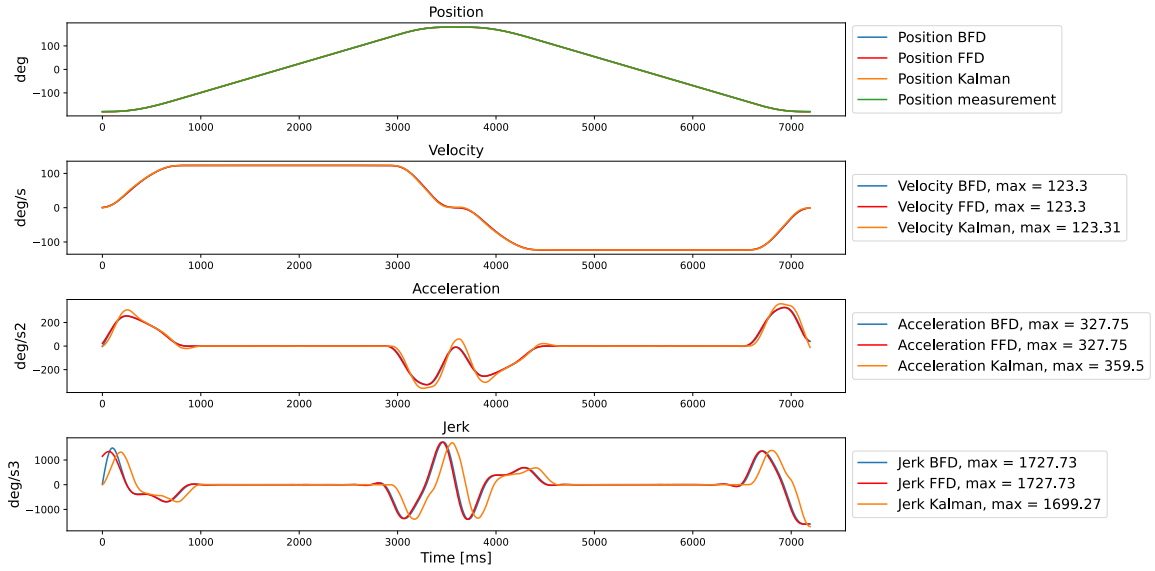

Figure 2-10: Robot joint: A1, Comparative analysis on the three time-derivation methods using position measurements

Time-derivation methods validated on the robot

In order to determine the method yielding the best results, a robot test was undergone. This experiment had two main goals:

- to determine if the high amplitude in the beginning of the jerk profile generated by the FFD will cause a controlled stop
- to identify if the phase-lag of the Kalman filter observed in the jerk profile generated is affecting the position generation.

It is important to mention that the test was developed using the trajectory generation algorithm which will be introduced in Chapter 4. Since this algorithm requires velocity, acceleration and jerk bounds to generate position, the velocity acceleration and jerk profiles provided by the time-derivation methods were used as bounds for the trajectory generation. Moreover, the robot playback method, in which position commands are provided to the robot, will be described in Section 2-1-2.

Effect of high jerk

The experimental test revealed that the high jerk in the beginning of the jerk profile provided by the FFD does not cause any controlled stop. A comparative plot, showing the original position measurement, the trajectory generated using the profiles computed by FFD and the recorded position after the generated trajectory was provided to the robot as position commands is depicted in Figure 2-11. Moreover, the errors are computed using the differences between the three position timeseries. One can observe that although the robot closely follows the generated position, it will not trigger a controlled stop.

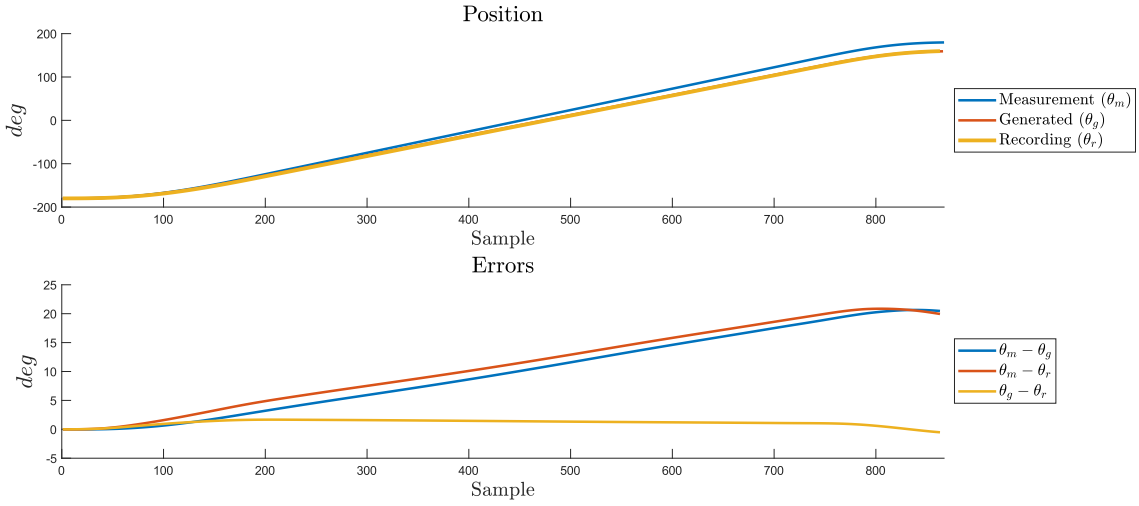

Figure 2-11: Robot joint: A1, Comparison between the position information of a robot experiment using the Forward Finite Differences as time-derivation method

Additionally, during the experiments, the trajectory generator obtained a faster acceleration phase in the case of FFD profiles as a result of the higher jerk in the beginning of the profile. Although the BFD profile was similar, the resulting acceleration phase was slower, leading to the robot to reach its maximum velocity later. Consequently, the faster acceleration phase provided by FFD will favor the minimum execution time requirement.

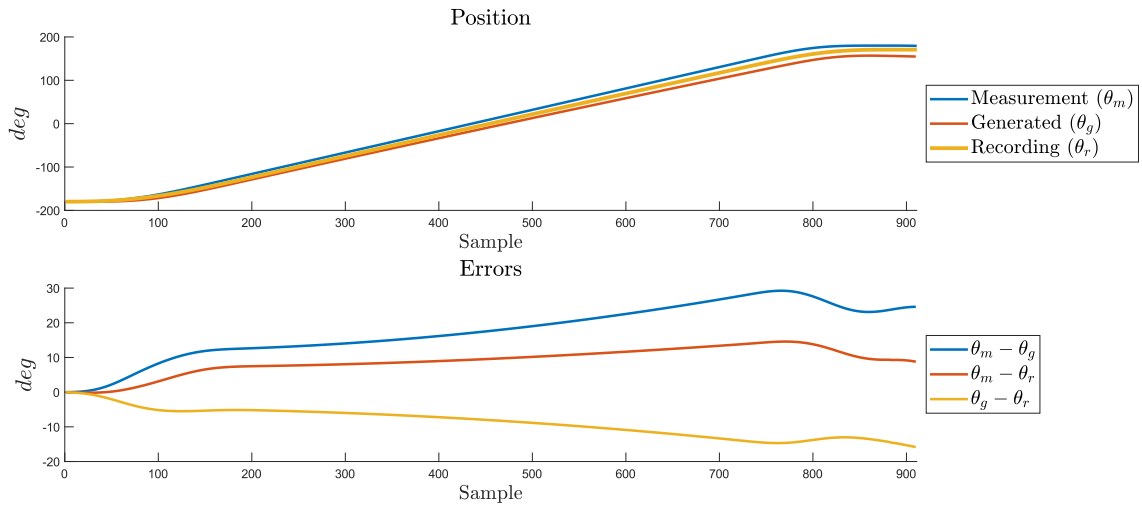

Figure 2-12: Robot joint: A1, Comparison between the position information of a robot experiment using the Kalman filter as time-derivation method

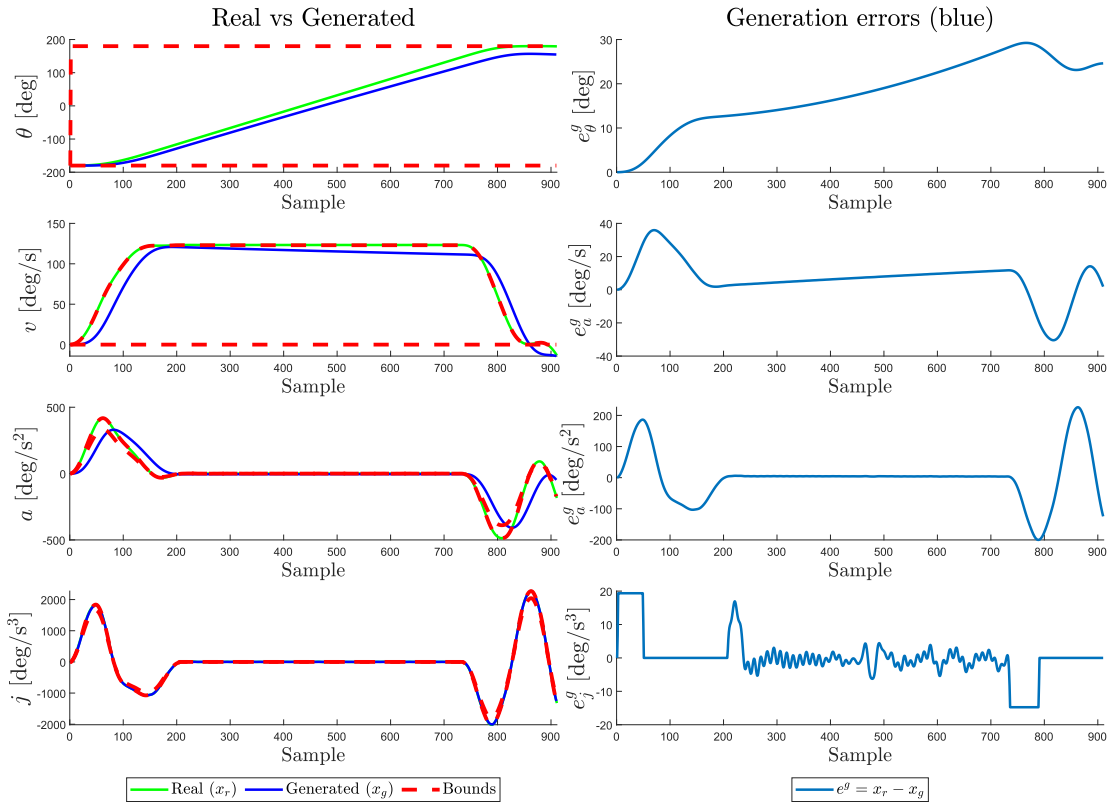

Figure 2-13: Robot joint: A1, Trajectory generation using as bounds the profiles computed using the Kalman filter

Effect of phase-lag

Furthermore, the phase-lag caused by the Kalman filter did not cause any controlled stop either. Providing similar information to Figure 2-11, Figure 2-12 displays the robot trajectory generated using the Kalman filter bounds and the results are very similar to the ones obtained using FFD in terms of accumulated error ($\theta_m - \theta_r$) and trajectory feasibility. On the other hand, Figure 2-13 provides the trajectory generation results which emphasize the original time-derivatives provided by the Kalman filter (from the recorded position) in comparison to the generated velocity, acceleration and jerk profiles, using the trajectory generator introduced in Chapter 4. One can observe that, when following the jerk profile computed using the Kalman method, the trajectory generation will introduce a phase-lag in the resulting velocity and acceleration profiles (compared to the original profiles).

To conclude, considering that the higher amplitude in the beginning of jerk profile computed using FFD seems to favor a reduced execution time and it did not cause any controlled stop on the robot, the Forward Finite Differences method will be used for time-derivation in the following experiments. Moreover, this decision is encouraged by the phase-lag of the Kalman filter that seems to be affecting the trajectory generation.

2-3 Experimental validation of the Limiter effect

2-3-1 Initial analysis

In order to first determine the existence and later the effect of the Limiter on the velocity, acceleration and jerk profiles, an initial analysis has been undergone. In this sense, using position measurements provided by the KUKA robot, the velocity, acceleration and jerk profiles were retrieved and multiple experiments were compared. As mentioned before, during each experiment only one robot joint was actuated to isolate their individual maximum capabilities.

Since multiple motion parameters could accentuate the effect of the Limiter, the influence of five parameters was investigated by varying only one parameter at a time. The methodology and the results of each individual analysis will be presented with the final scope of building a more extensive dataset.

Payload weight

For highlighting the influence of the payload weight on the Limiter's effect, the same motion, defined by the same start and target positions, was executed with different payload weights. Additionally, one specific motion is also defined by a chosen robot pose in which the non-actuated joints were also in the same position, to avoid further effects.

In this sense, Figure 2-14 shows the different profiles obtained when actuating joint A1 in the "transp" robot pose, with the robot carrying no payload and 176kg, respectively. It is important to mention that 176kg is the maximum payload that could be attached to the robot using the barbell setup due to bar length limitations, being the closest value to the robot's rated payload of 210kg.

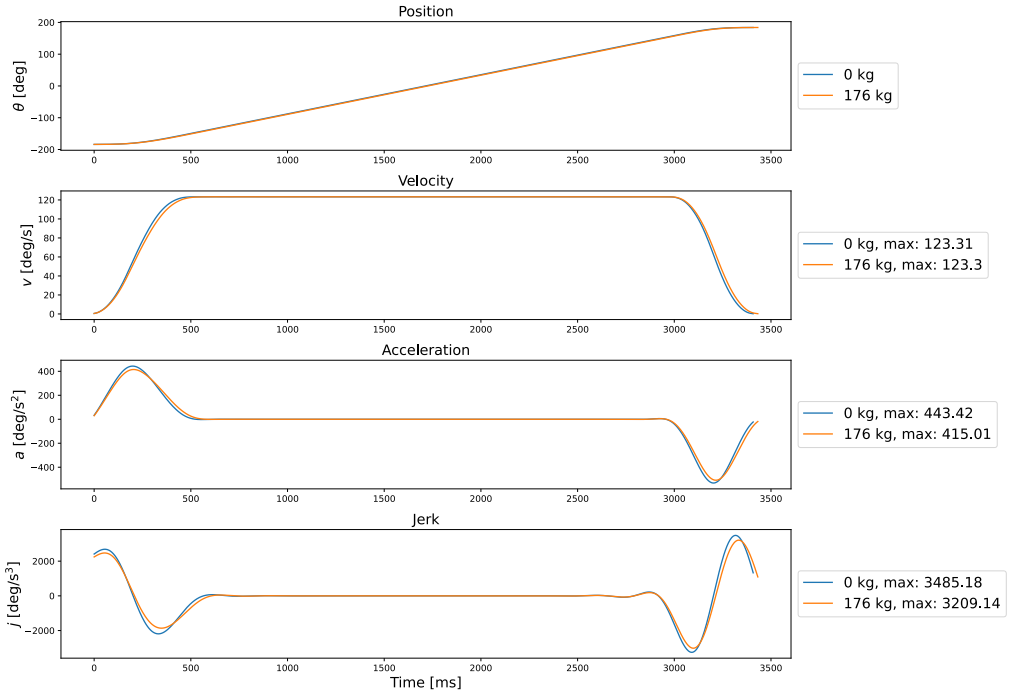

Figure 2-14: Robot joint: A1, Profiles comparison between the same motion executed with the same robot pose 'transp' but different payloads

Moreover, Figure 2-15 provides the same results for the joint A2 carrying the same payloads (0kg and 176kg), but using 'conf1' robot pose. In this case, the effect of the Limiter is even more pronounced and it is highlighted in the figure by the length of the second motion. When the maximum acceleration and jerk are lower, the motion will be slower and the robot will reach its target later in time.

Considering the results highlighted by Figure 2-14 and Figure 2-15, the importance of the payload weight in the limitations imposed on acceleration and jerk profiles is validated.

Positions of the non-actuated joints

Considering that the mechanical properties imposed by the position of the joints could influence the Limiter's effect, an analysis was performed to observe the influence of the robot pose on the limitations enforced by the robot controller on the velocity, acceleration and jerk profiles. The definition of a robot pose, together with the description of all the robot poses used in the experiments are provided in Subsection 2-1-3.

In this sense, Figure 2-16 depicts the comparison between similar motions, executed with the robot in two different robot poses "transp" and "conf1", by keeping the payload weight constant (0kg). Apart from the different profiles, one can observe the difference in the maximum acceleration and jerk values. Moreover, the rated velocity guaranteed by the manufacturer is achieved independent of the acceleration and jerk profiles. Similar results were observed for other joints, but for brevity their associated plots will be detailed in A-2.

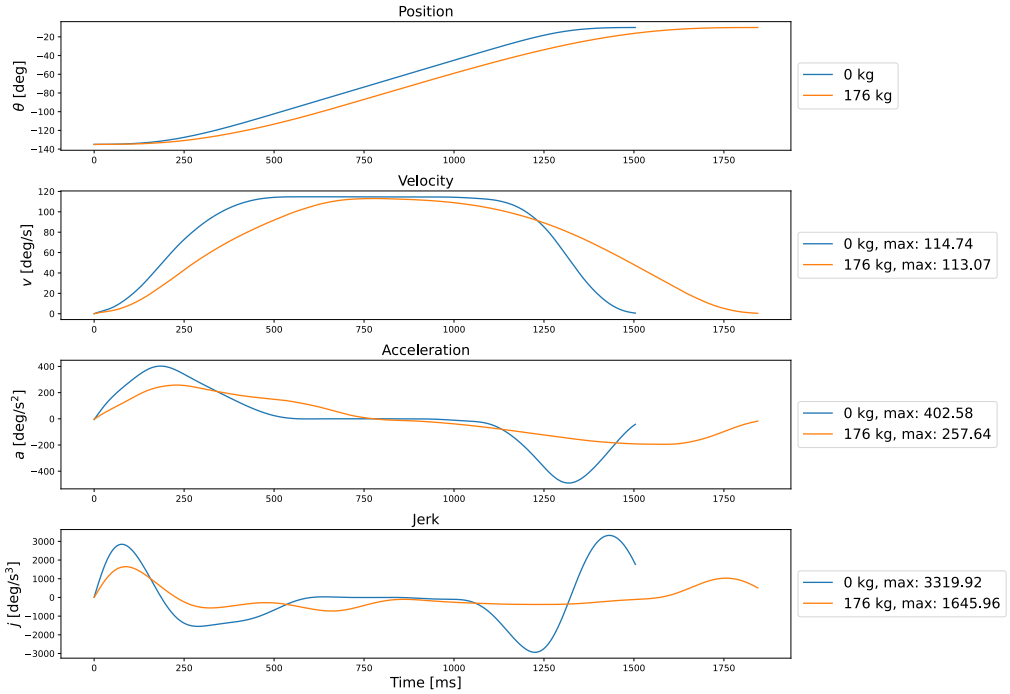

Figure 2-15: Robot joint: A2, Profiles comparison between the same motion executed with the same robot pose 'conf1' but different payloads

Considering the results shown in Figure 2-16, the conclusion is that the positions of the non-actuated joints has a significant impact on the Limiter's effect.

Payload's Centre of Mass relative to the robot flange

Another parameter that could influence the Limiter's effect on the velocity, acceleration and jerk profiles is the position of the payload's CoM relative to the robot flange. This could affect the mechanical properties and cause the robot controller to limit the maximum acceleration and jerk that could be achieved.

During several experiments, the payload's CoM was varied by placing the weights on the bar of the barbell setup further from its center. Unfortunately, due to limitations regarding the bar length, the maximum distance that could be obtained was approximately 30cm.

A comparative plot, showing the profiles obtained by actuating A1 multiple times between the same start and target positions, with a constant payload and different r distances is provided in Figure 2-17. In this figure, one can observe no significant difference between the profiles obtained. As a result, the conclusion of this analysis is that although the payload's CoM might influence the Limiter's effect, the barbell setup could not help in highlighting this. As a result, this parameters will not be varied further in the final dataset.

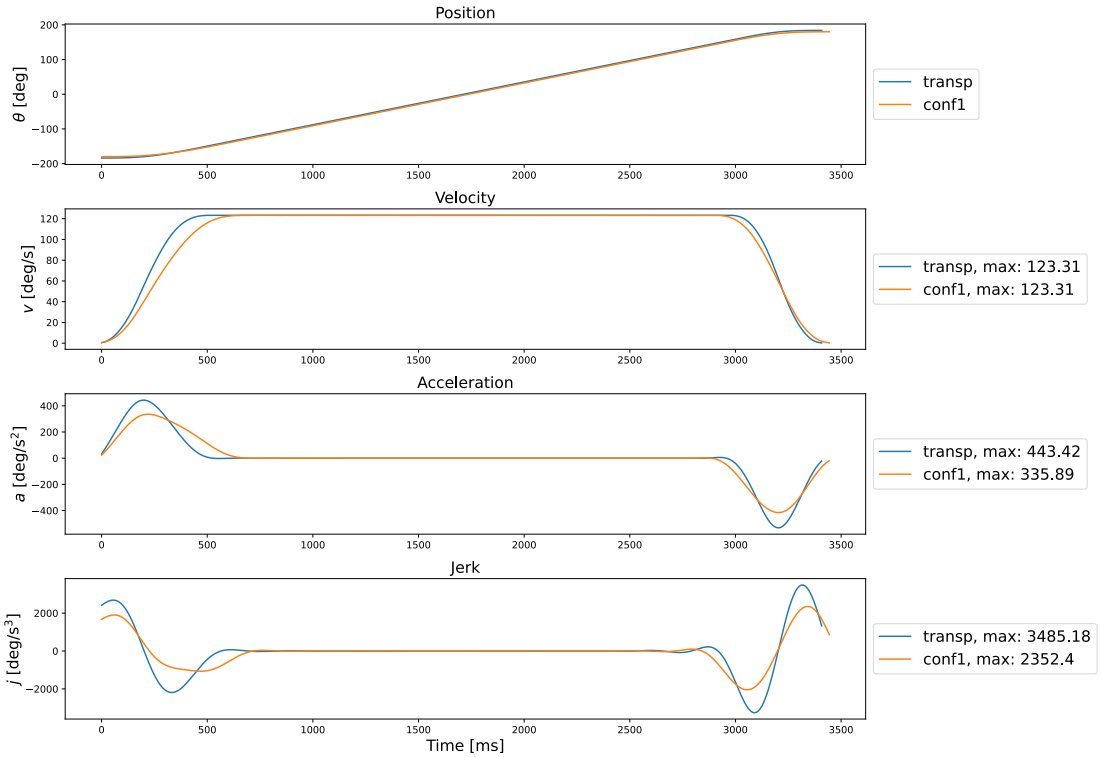

Figure 2-16: Robot joint: A1, Profiles comparison between similar motions executed with no payload but different robot poses

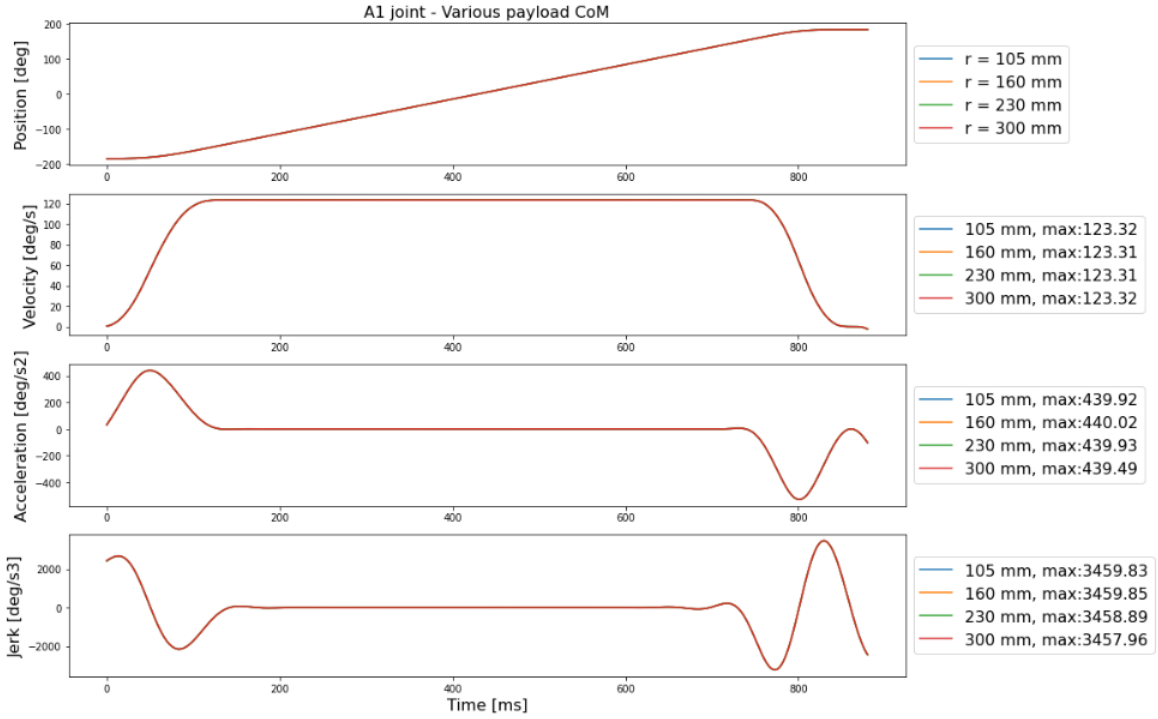

Figure 2-17: Robot joint: A1, Profiles comparison between the same motions executed with the same payload weight, same robot pose, but different payload CoM

Speed level

To investigate the effect of different speed levels, several experiments with constant payload weight and CoM, the same robot pose and the same start and target positions were recorded. As a result, the sole difference between motions was the speed level.

The experiment exemplified in Figure 2-18 showed that the obtained profiles are not connected by any relationship such as proportionality depending on the different speed levels. Moreover, since the maximum speed level corresponds to the minimum execution time of a trajectory, it will emphasize the effects of the Limiter block. As a result, only the 100% speed level will be used in further experiments.

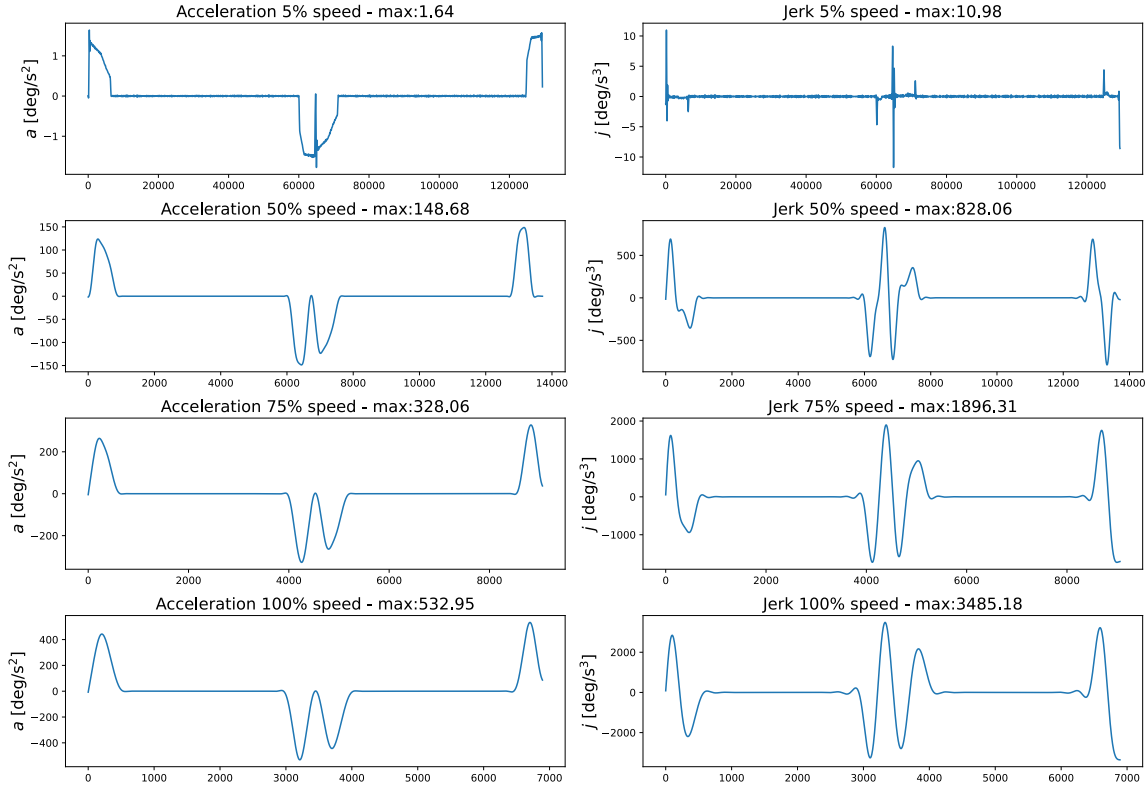

Figure 2-18: Robot joint: A1, Profiles comparison between the same motions executed with the same payload weight and CoM, same robot pose, but different speed levels

Start and target position

Finally, in order to observe if the start and target position play a significant role in the velocity, acceleration and jerk profiles obtained, the payload and the robot pose were kept constant but the trajectory length was reduced. The reduction was achieved by moving the target position closer to the start.

The main conclusion of the experiment was that the robot will reach maximum velocity all the time if the motion is long enough. In the case of joint A1, the lower limit for the trajectory length is set to about 60 degrees. Moreover, Figure 2-19 proves that the motions

that reached maximum velocity present similar acceleration and deceleration times, since the velocity profiles perfectly overlap.

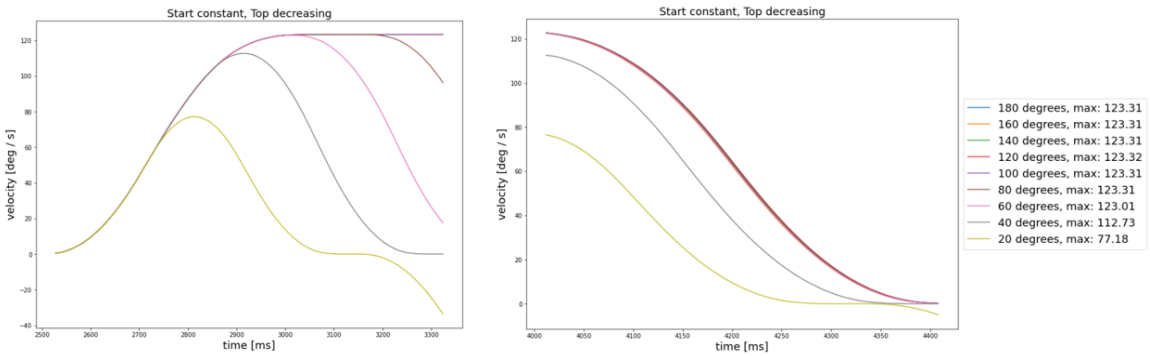

Figure 2-19: Robot joint: A1, Velocity comparison between motions executed with the same payload weight and CoM, same robot pose, varying the motion length.
Left: Acceleration phase, Right: Deceleration phase

2-3-2 Extending the dataset

As the initial analysis proved, out of the motion parameters that could be varied in the current setup, the main parameters that could accentuate the effects of the Limiter block are payload weight and the robot pose. Moreover, if the aim is to obtain trajectories that reach the rated velocity of each joint, the motions should be long enough to allow this performance. As a result, the initial analysis was followed by more experiments that allowed building a more extensive dataset. Its aim is to be used further for identifying the Limiter's behavior.

The four robot poses defined in Table 2-5 and Table 2-6, were extended for all joints. The main reason behind using these robot poses was that "transp" and "shorter" were expected to excite less the Limiter by being similar to the transport configuration defined by KUKA (which should reduce the effect of the mechanical forces). Conversely, "conf1" and "conf2" should accentuate the effect of the Limiter by exercising more extended robot poses. Compared to the transport configuration, in which the robot joints are arranged such that the robot looks closed around its CoM, when exercising more extended configurations, the robot joints are arranged such that the robot's end-effector is placed far from the robot's CoM.

In terms of the payload weight, this was varied between 0kg and 176kg. As mentioned before, 176kg was the maximum payload that could be attached to the robot using the barbell tool. As a result, Table 2-7 provides the dataset grid composed of 34 experiments for each joint. The checkmarks present in the table illustrate the existence of an experiment characterized by the payload weight and robot pose combination.

		Payload weight [kg]										
		0	6	16	36	56	76	106	126	146	166	176
Robot pose	"transp"	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	"shorter"	✓	✓		✓		✓		✓			✓
	"conf1"	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	"conf2"	✓	✓		✓		✓		✓			✓

Table 2-7: Dataset definition depending on payload weight and robot pose

It is important to note that the four robot poses are defined differently for each of the robot joints. For the joints that were mostly used in this work, the definition of the robot poses are presented in Table 2-5 and Table 2-6. Representing important information such that the robot experiments could be replicated, the definition of the four robot poses for the other five robot joints are defined in A-1.

The performed experiments revealed different degrees of impact generated by the varied parameters at each joint level. More specifically, joints A1, A2 and A3 showed considerable differences in the profiles obtained by varying the motion parameters. On the other hand, joints A5 and A6 proved reduced impact, while joints A4 and E1 proved almost no impact at all. For brevity, only joint A1 and A2 will be covered in this section, while figures depicting the other joints' experiments can be found in A-2.

Furthermore, Figure 2-20 and Figure 2-21 show the maximum acceleration and jerk values for both A1 and A2 depending on the payload weight. Each robot pose is represented by an individual color. One can notice the impact of the varied parameters on the maximum values of the acceleration and jerk profiles. As expected, the plots prove reduced differences between either "transp" and "shorter" robot poses or "conf1" and "conf2". However, the difference between the two groups is noticeable.

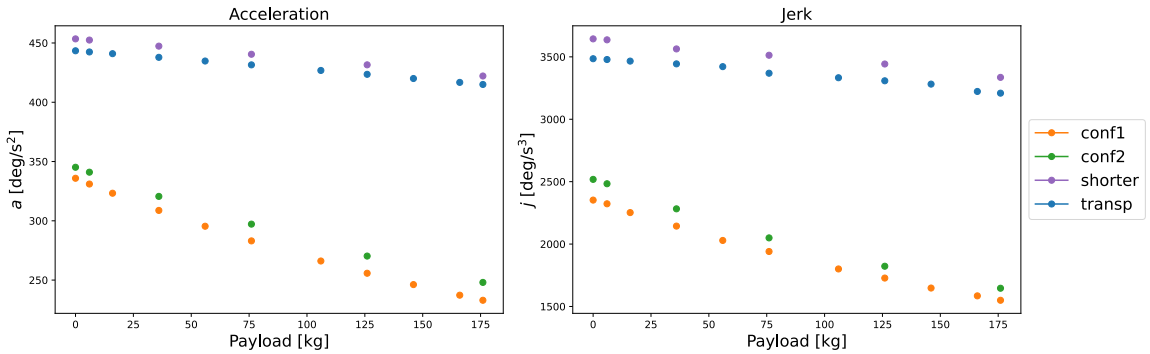

Figure 2-20: Robot joint: A1, Maximum acceleration and jerk values of the motion defined by payload weight and robot pose.

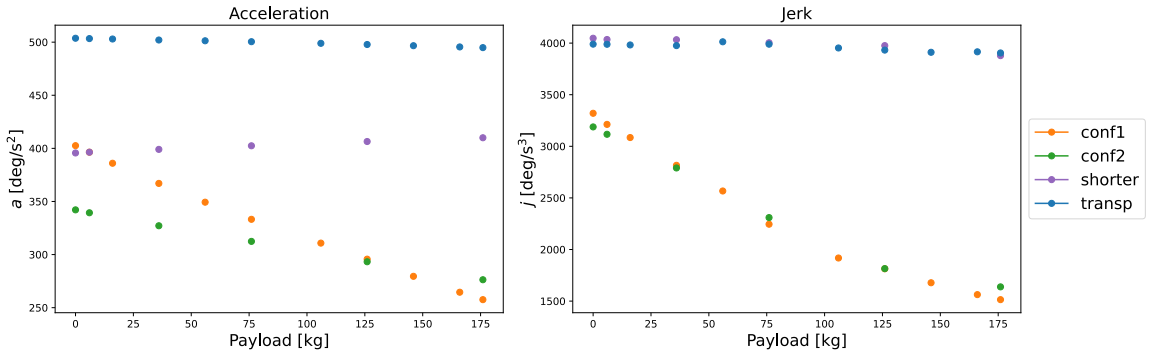

Figure 2-21: Robot joint: A2, Maximum acceleration and jerk values of the motion defined by payload weight and robot pose.

Another empirical observation is that the maximum velocity obtained by time-derivation on the position recordings might slightly differ from the rated velocity specified by the manufacturer. Different values for the maximum velocity can be the result of the time-derivation procedure, as there is no conclusion that can be made on the manufacturer's decision to specify the rated values (the provided rated velocity could be given rounded).

This maximum velocity is particularly important for the trajectory generation procedure that will typically use computed velocity profiles. When specifying a lower rated velocity in the trajectory generator (compared to the computed maximum velocity), the trajectory generator will always violate the velocity constraint and fail to determine a trajectory. When the specified rated velocity is higher than the computed maximum velocity, the trajectory generator might determine trajectories that prove unfeasible when sent to the robot. As this will be further detailed in Chapter 4, the trajectory generation will make use of the maximum velocities empirically determined. Table 2-8 depicts the rated velocity provided by the manufacturer and the maximum velocity obtained through time-derivation for each joint.

Robot joint	Rated velocity	Maximum velocity computed
A1	123 °/s	123.30 °/s
A2	115 °/s	114.74 °/s
A3	112 °/s	112.05 °/s
A4	179 °/s	178.78 °/s
A5	172 °/s	172.50 °/s
A6	219 °/s	219.90 °/s
E1	1.96m/s	1.955m/s

Table 2-8: Joints rated velocity compared to maximum velocity obtained from time-derivation

2-4 Conclusion

In the current chapter the robot setup was introduced, including a short description of the robot system and its communication interfaces. A method aiding the user to retrieve position

data from the robot was presented. Additionally, the procedure used for providing trajectories as position commands to the robot and record the executed motions was defined. In this sense, the playback method revealed errors that occur between the trajectory provided as input and the recorded motion that symbolizes the system's output. Moreover, the non-determinism of this method was demonstrated, proving that multiple experiments should be executed to ensure that a controlled stop is indeed caused by an unfeasible input trajectory.

Further, the experimental setup including a custom tool was presented. The tool is similar to the barbell systems used in weightlifting competitions and permits weight and CoM adjustment.

Since the robot system only provides position data, three time-derivation methods were proposed for retrieving velocity, acceleration and jerk information. Out of these, Forward Finite Differences showed the most promising results. The main benefits of FFD, compared to a Kalman filter and the Backward Finite Differences method, are the lack of a visible lag in the derived profiles and the higher amplitude in the beginning of the jerk profile which might lead to faster trajectories.

Using the velocity, acceleration and jerk profiles obtained from multiple robot motions, an initial analysis was executed to experimentally validate the existence of the Limiter effect. After multiple motion or tool parameters were examined, the analysis showed that the payload's weight and the robot pose have a significant impact on the bounds imposed by the Limiter on velocity, acceleration and jerk. Using these two parameters, an extended dataset was built to reveal the behavior of the Limiter block.

Chapter 3

Bounds prediction

Considering the absence of knowledge about the construction of the Limiter block, its behaviour will be identified in a data-driven approach that does not affect the robot's operation. Moreover, as a result of the initial analysis presented in Chapter 2, the imposed bounds seem to be dependent on various robot and tool parameters. In this sense, in order to determine the Limiter's behaviour, the velocity, acceleration and jerk bounds should be predicted using only previously recorded data, given robot and tool information.

For introducing the proposed solution for bounds prediction, the overall approach will be presented covering the bounds definition, the statement of the modeling problem and several modeling decisions. Further, three modeling approaches for solving the problem will be addressed, presenting their benefits and drawbacks. Finally, the modeling conclusion will be stated and the chosen method will be motivated.

3-1 Strategy

Bounds interpretation

As mentioned in Chapter 2, the manufacturer guarantees that the maximum speed level (speed level 100%) will generate the fastest possible trajectory between the given start and target without triggering a controlled stop. The latter is covered by the Limiter block being a significant part of the robot controller. In this sense, one can infer that the velocity, acceleration and jerk profiles achieved when running the robot in the 100% speed level represent the maximum performances that the robot could achieve during that specific motion, since the robot controller aims for both dynamically-feasible and minimum-time trajectories. As a result, for representing the velocity, acceleration and jerk bounds imposed by the Limiter block, the velocity, acceleration and jerk profiles obtained when executing a motion with maximum speed level will be used.

Modeling idea

Following the reasoning above, the modeling problem aims to determine the velocity, acceleration and jerk bounds that the Limiter block will impose, based on various motion parameters previously known. Since defining an entire motion profile (composed of hundreds of samples) based on several integer motion parameters proves a rather complex task, especially due to the variable duration of the motion, the modeling approach included a more iterative method. Precisely, the solution aims to determine the next maximum velocity, acceleration and jerk value, given the current robot state (defined by position, velocity, acceleration and jerk) and the motion characteristic parameters. In a mathematical formulation, the desired mapping $\hat{f}$ is provided in (3-1).

$$\mathcal{B}(k+1) = \hat{f}(x(k); p_m) \tag{3-1}$$

where $\mathcal{B}(k+1) = [\bar{v}(k+1)\ \bar{a}(k+1)\ \bar{j}(k+1)]^T$ contains the bounds to be determined for the next step, $x(k) = [\theta(k)\ v(k)\ a(k)\ j(k)]^T$ denotes the current robot state defined by position, velocity, acceleration and jerk, and p_m designates all the motion parameters that are constant throughout the motion. This list of motion parameters includes payload weight, start and target positions, robot pose information, inertia terms, tool's CoM relative to the robot flange etc.

It is important to mention that during the modeling phase the current robot state and the bounds will coincide since the profiles obtained when running a motion with the maximum speed level provides also the bounds. On the other hand, in the final solution that combines bounds prediction and trajectory generation, the robot state and the bounds values might differ. More specifically, the trajectory generator will receive the next bounds and will generate the next position sample that fulfills all the bounds. Since the prediction of all three bounds is independent, the most conservative one should always be respected during the trajectory generation, guaranteeing that all others are fulfilled. For example, the fulfillment of a reduced jerk bound might imply achieving a lower velocity than the predicted velocity bound might require.

Nevertheless, the dataset used for modeling the mapping $\hat{f}$ was composed by the extended dataset presented in Chapter 2. For each joint, the set contained the information of 34 motions. In order to also be able to verify the performance of the model, the dataset was split into training and test subsets. The distribution of these are shown in Table 3-1 where the triangles mark train data and the red circles mark test data. Moreover, for the validation set, Cross-validation (CV) was used. According to [11], Cross-validation is a data resampling method used to estimate the true prediction error and to tune model parameters. In this work CV was used to increase the credibility of the validation step without reducing the training set even further.

		Payload weight [kg]										
		0	6	16	36	56	76	106	126	146	166	176
Robot pose	"transp"	△	◯	△	△	△	◯	△	△	△	△	△
	"shorter"	△	◯		△		◯		△			△
	"conf1"	△	◯	△	△	△	◯	△	△	△	△	△
	"conf2"	△	◯		△		◯		△			△

Table 3-1: Train and test split of the dataset used for modeling. △ marks train data and ◯ marks test data.

Feature engineering

Since the mapping $\hat{f}$ will be identified using machine learning algorithms from previously collected data, the selection of features, representative for both the problem and the dataset was an essential step. Using the motion parameters, that are kept constant throughout one robot motion, and the current robot state, the features were defined according to Table 3-2. Further, the motivation behind the selection of each feature is also covered in the table. For causality purposes, all features specified in the table are known before the sample $k+1$ and they will be used to determine the next expected bounds for velocity, acceleration and jerk.

Feature	Motivation
Payload weight	Initial analysis showed its influences on the Limiter's effects.
Start position	Characterizes the traveled distance.
Traveled distance (difference between start and target)	Initial analysis showed that a long enough traveled distance will guarantee the maximum velocity being achieved.
Progress (normalized current position)	Part of the current robot state. The progress regarding its target might be relevant for deciding when the deceleration phase should happen.
Current velocity	Part of the current robot state.
Current acceleration	Part of the current robot state.
Current jerk	Part of the current robot state.
Positions of joints A_i where $i \neq$ to the actuated joint	Represent the robot pose which was proved to influence the Limiter's effect.
Inertia terms x, y, z	They might influence the mechanical properties of the robot tool, from a resistive perspective.
Tool CoM position	They might influence the mechanical properties of the robot tool, from a resistive perspective.

Table 3-2: Modeling features and the motivation behind their usage

In order to emphasize the expert knowledge gained during the initial analysis phase, multiple feature engineering ideas were applied. Although not all improved the outcome, several will be presented further. For brevity, the examples provided regard velocity information, but in the final solution they are also extended to acceleration and jerk:

- **Usage of Δv instead of** v_{k+1}, where $\Delta v = v_{k+1} - v_k$
  During the modeling phase it was observed that when predicting the next bound for velocity, based on the current robot state and the motion parameters, the learning algorithm will use mostly the current velocity v_k to predict the next bound for velocity $\bar{v}_{k+1}$. Moreover, since the next velocity value cannot vary much from the current velocity, only the change in velocity might be relevant.

  After implementing the final solution that combines this prediction approach with the trajectory generation, the conclusion was not to continue with this approach. The main reason was that the repeated additions of Δv to v_k were increasing the offset between real data and prediction.

- **Usage of $v \cdot n_{p_w}$ instead of** v, where n_{p_w} denotes normalized payload weight defined by (3-2), where p_w is the actual payload weight and $p_{\max}$ is the robot rated payload weight of 210kg.

$$n_{p_w} = \frac{-0.9 p_w}{p_{\max}} + 1 \tag{3-2}$$

  According to the initial analysis, the payload weight is one of the dominant factors in the Limiter's effect. As a result, for applying this knowledge to the data, the velocity was multiplied with the normalized payload weight. The payload weight values were normalized to avoid the multiplication with 0 in the case of no payload experiments. This approach adjusted the payload weight range from [0, 176] to [1, 0.25].

  The difference between the prediction using the v and $v \cdot n_{p_w}$ is presented in Figure 3-1. The figure shows in the first plot the predicted velocity multiplied with the normalized payload. In the second plot the velocity values obtained using both the straightforward method (predicting $\bar{v}$) and the proposed approach (predicting $\bar{v} \cdot n_{p_w}$) are compared with the velocity computed using the position measurement (denoted as measured). In order to actually notice the improvement, the third plot shows the prediction error for both methods. One can observe that adopting the $v \cdot n_{p_w}$ significantly reduced the Linear Regression error on the test set. The noisy pattern that can be observed in the prediction errors obtained by the straightforward approach might show that the method obtains higher prediction errors during acceleration and deceleration regions and reduced errors during maximum velocity regions.

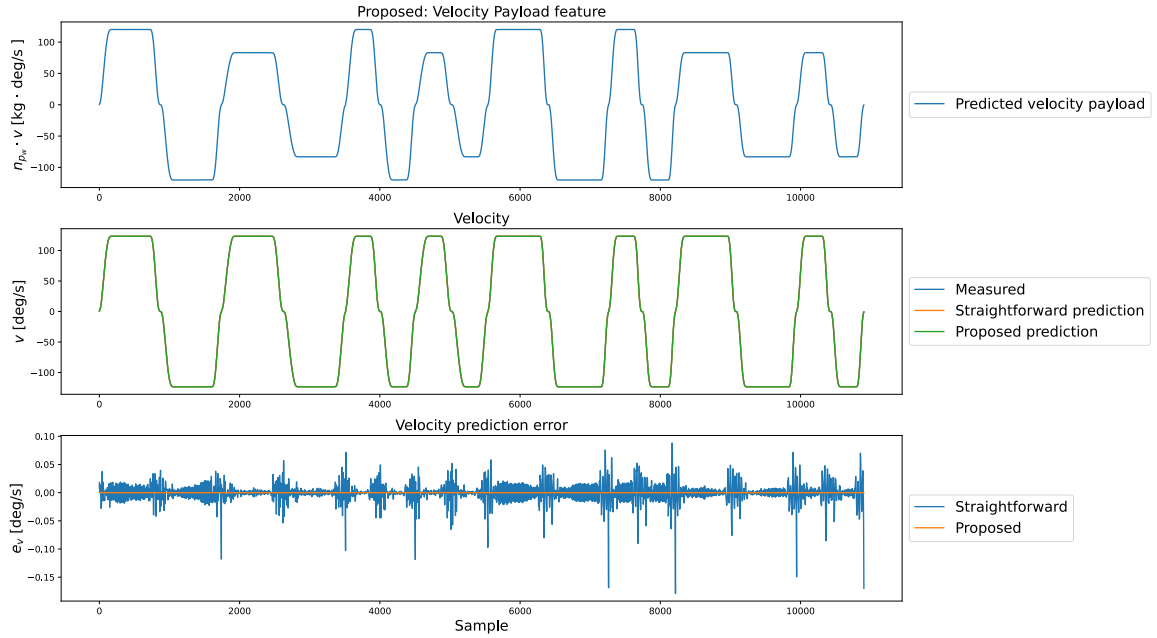

Figure 3-1: Robot joint: A1, Comparison between straightforward approach (predicting $\bar{v}$) and the proposed approach (predicting $\bar{v} \cdot n_{p_w}$). As base line - velocity computed from measured position (denoted as measured)

Conservative predictions

Since the direction of the errors could not be controlled when using typical prediction algorithms, errors could cause the predicted bounds to exceed the real bounds imposed by the robot controller. If the final trajectory executed by the robot was the result of a higher bound caused by prediction errors, the robot would reach a controlled stop.

In order to avoid dynamic bounds violation leading to a controlled stop, a method to enforce the prediction of bounds below the measured bounds was designed. Unfortunately, this method reduced the minimum-time performances, but favored a dynamically feasible trajectory.

For enforcing conservative predictions that do not exceed the measured bounds, a custom objective function for the regressor was introduced in (3-3), where z denotes the real value to be predicted, $\hat{z}$ is the prediction and α and γ are tuning parameters. Moreover, $\Phi(\hat{z}, z)$ depicts the custom objective function.

$$\Phi(\hat{z}, z) = (z - \hat{z})^2 + \alpha e^{\gamma(\hat{z}^2 - z^2)} \tag{3-3}$$

The first term of the objective function minimizes the prediction error, while the second term aims for predictions that are lower than the real value. The parameters α and γ weight the two terms by either reducing the prediction error or introducing more conservative predictions.

In a broader sense, for guaranteeing that no prediction exceeds the actual values, the error $\hat{z}^2 - z^2$ should always stay below 0. The squared terms are motivated by the existence of

negative values in the acceleration and jerk profiles, for which the prediction should be above the real value.

Further plots showing the benefits of the custom objective function will be provided in the modeling approaches section.

3-2 Modeling approaches

For predicting the bounds using previously collected data samples, multiple modeling approaches were tested. To provide a simplistic benchmark for the proposed approaches, Linear Regression was applied thanks to its simplicity. Further a Random Forest Regressor was introduced to adapt to the model nonlinearities and to accommodate the reduced training dataset. Finally, to enforce the custom objective function that favors conservative predictions, a boosting algorithm called XGBoost was used.

3-2-1 Linear Regressor

The linear regression approach is used to explain a variable y using known variables X = [x_1 ... x_p] [12]. This method can be exemplified using (3-4) where the parameters β_1, ... β_p represent model coefficients and e is the model intercept. This expression can be exploited to predict y from input variables x_1, ... x_p. Given a set of observations for y and X, the model coefficient and the intercept can be retrieved by minimizing a cost function. In the current work, the cost function to be minimized in the linear regression is the sum of squared errors, presented in (3-5), leading to a least squares approach.

$$y = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + e = X\beta + e \tag{3-4}$$

$$\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{3-5}$$

, where y stands for the expected model output, $\hat{y}$ for its prediction and N for the number of observations in the set.

The linear regression approach was used for the bounds prediction mostly for its simplicity. Moreover, the parameters are easy to interpret and it provides satisfactory approximations when dealing with a reduced dataset [13]. When training the model on the train dataset, the model coefficients identified for predicting the augmented velocity, acceleration and jerk values are presented in Figure 3-2.

Furthermore, Figure 3-3 provides the comparison between the expected output and the model predictions on a single motion belonging to the test set. The prediction error for the respective comparisons is presented as well, where y_r depicts the real value and y_p depicts the predicted value. For clarity, Table 3-3 shows the mean absolute error and the maximum error for both the train and test dataset. As the test set contains more motions than the one depicted in Figure 3-3 depicts, the results presented in Table 3-3 might not coincide to the ones depicted in Figure 3-3.

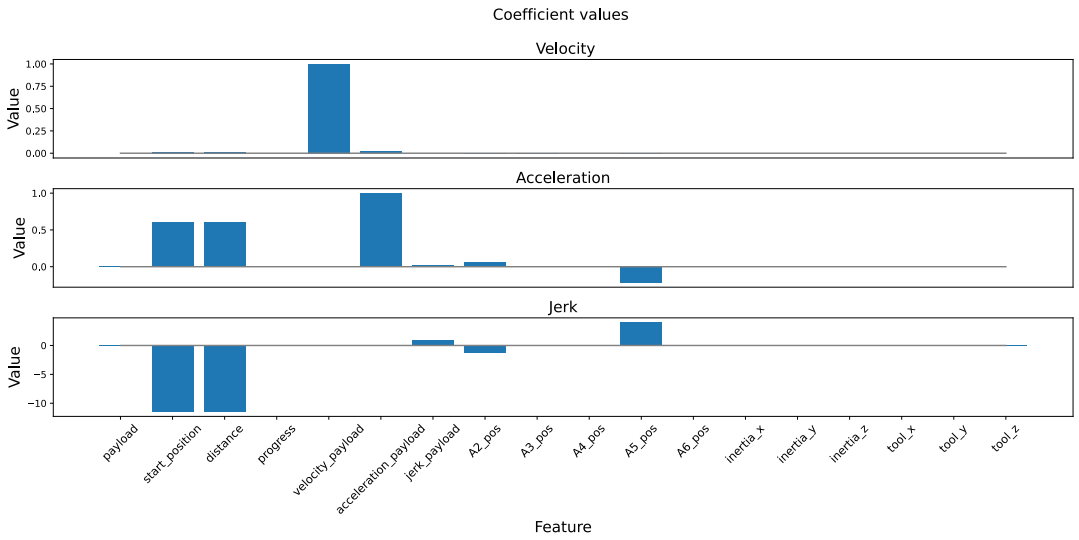

Figure 3-2: Robot joint: A1, Linear Regression model coefficients

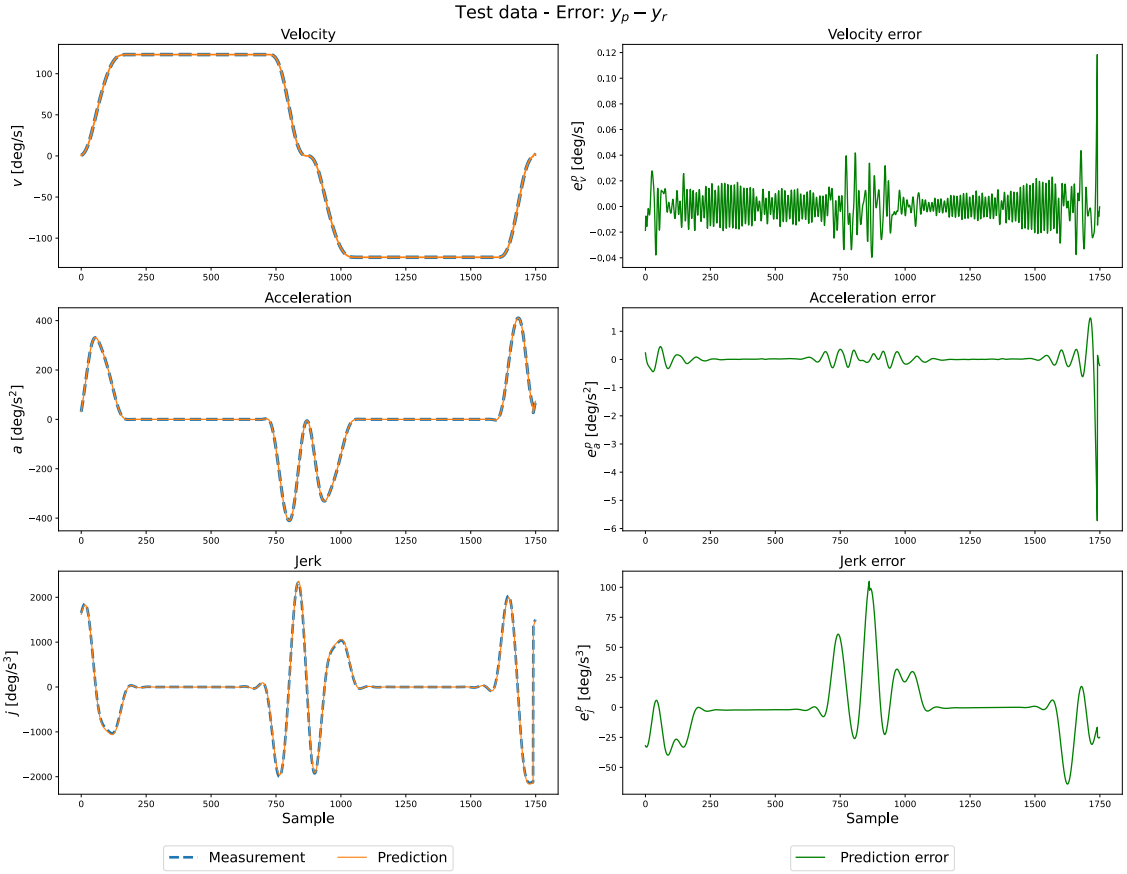

Figure 3-3: Robot joint: A1, Experiment: 6 kg, robot pose: 'conf1'.
Comparison between the expected output and the predicted output using Linear Regression

	Train				Test			
	Max error	Max error %	MAE	MAE %	Max error	Max error %	MAE	MAE %
Velocity	0.18	0.15%	0.01	0.01%	0.18	0.15%	0.01	0.01%
Acceleration	9.82	1.80%	0.20	0.04%	9.77	1.79%	0.20	0.04%
Jerk	172.74	4.74%	18.37	0.50%	173.20	4.76%	20.63	0.57%

Table 3-3: Robot joint: A1, Prediction error for Linear Regression model on train and test dataset

Overall, the performance obtained when using the Linear Regression is noticeable and interpretable. On the other hand, the model complexity is reduced and does not map the nonlinear relations between the model features and the output. As a result, the Linear Regression was kept as benchmark, standing for the minimum performance that could be obtained for the bounds prediction. Further, more complex approaches were introduced.

3-2-2 Random Forest Regressor

In order to adapt to the nonlinear relations between the model features and the output, a Random Forest (RF) Regressor was introduced. Being an ensemble method, RF provides satisfactory results when dealing with rather small training datasets [14]. Random Forest fits a number of decision trees (ensemble methods principle) on various sub-samples of the dataset (the bagging principle) and uses averaging to improve the predictive accuracy and control overfitting [15].

For creating the decision trees, RF chooses from a given number of features, the one that will minimize the given loss function. When using the Sklearn Python library for implementing a Random Forest Regressor, the default loss function to be minimized is the sum of squared errors, just like for the Linear Regression. Moreover, in the current approach, a forest composed of 70 trees was used and the maximum tree depth was set to 25.

For analyzing the prediction results for the Random Forest regressor the bound prediction on a motion belonging to the test data is presented in Figure 3-4. Further, prediction error information for both the train and test datasets are presented in Table 3-4. Similar to the LR results, the results in Table 3-4 might not match the errors depicted in Table 3-4 since the figure contains only one motion and the test set contains multiple motions. Compared to the LR model, the RF regressor seems to predict better the jerk but worse the velocity and acceleration.

	Train				Test			
	Max error	Max error %	MAE	MAE %	Max error	Max error %	MAE	MAE %
Velocity	1.87	1.52%	0.11	0.09%	2.33	1.89%	0.46	0.37%
Acceleration	16.51	3.02%	0.49	0.09%	15.91	2.92%	0.78	0.14%
Jerk	159.94	4.39%	8.97	0.25%	156.70	4.31%	14.26	0.39%

Table 3-4: Robot joint: A1, Prediction error for Random Forest Regressoron train and test dataset

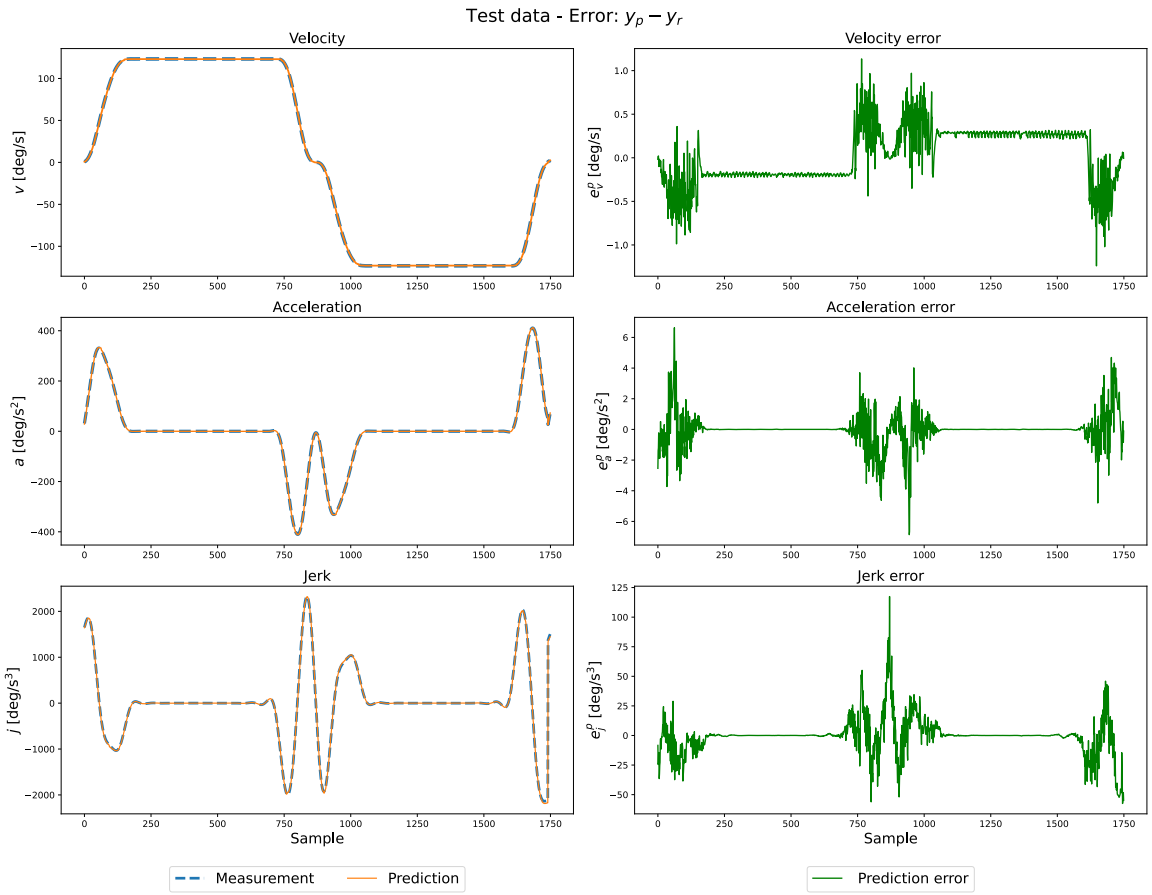

Figure 3-4: Robot joint: A1, Experiment: 6 kg, robot pose: 'conf1'. Comparison between the expected output and the predicted output using Random Forest Regressor

3-2-3 XGBoost Regressor

The Extreme Gradient Boosting (XGBoost) regressor permits the model to correct prediction errors made by previous models. This characteristic is typical for the boosting algorithms. In this sense, XGBoost uses any differentiable loss function and the gradient descent optimization algorithm to determine the model that fits the given data. Its main benefits lay in the execution speed and the model performance as introduced in [16].

In the library used for implementing the method, by default, the XGBoost minimizes the sum of squared errors presented in (3-5). Moreover, since the XGBoost is a rather complex approach that permits the identification of nonlinear dynamics, its performance was compared with the models introduced before.

The prediction results on only one motion included in the test set is presented in Figure 3-5. Furthermore, the prediction error information is presented in Table 3-5 for both the training and test data. One can observe a reduced prediction error on jerk, but a higher prediction error on velocity and acceleration.

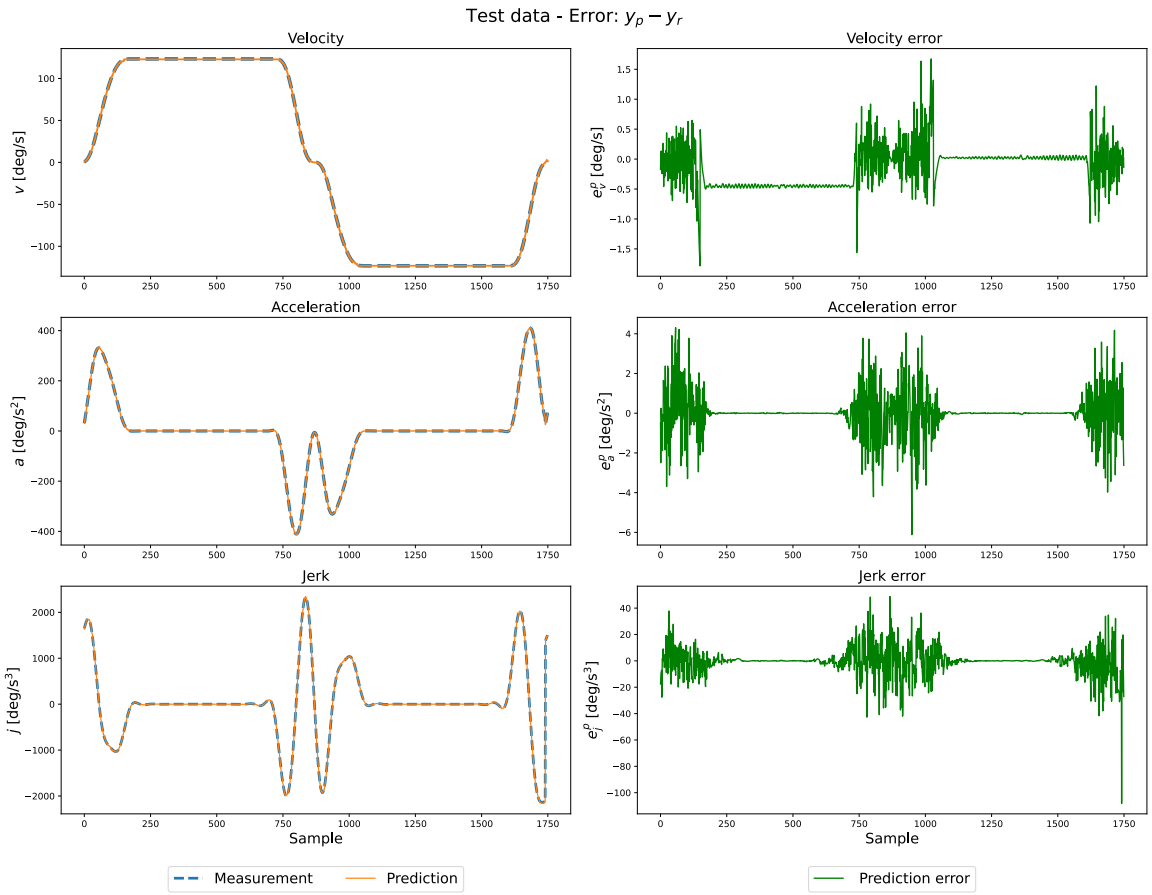

Figure 3-5: Robot joint: A1, Experiment: 6 kg, robot pose: 'conf1'. Comparison between the expected output and the predicted output using Extreme Gradient Boosting Regressor

	Train				**Test**			
	Max error	Max error %	MAE	MAE %	Max error	Max error %	MAE	MAE %
Velocity	3.11	2.52%	0.15	0.12%	2.94	2.39%	0.68	0.55%
Acceleration	12.2	2.31%	0.65	0.12%	14.25	2.61%	0.66	0.12%
Jerk	111.45	3.06%	6.82	0.19%	156.60	4.31%	9.013	0.25%

Table 3-5: Robot joint: A1, Prediction error for Extreme Gradient Boosting Regressor on train and test dataset

Since the XGBoost implementation in Python allows the definition of a custom objective function, this approach was combined with the objective function defined in (3-3) which aims for more conservative predicted values. In this sense, Figure 3-6 provides the comparison between the expected output values for a motion in the test set and the predicted profiles using two different sets of (α, γ) parameters of the objective function in (3-3). More precisely, the first set is defined by $\alpha = 0.8$ and $\gamma = 0.22$, while the second set is defined for $\alpha = 0.75$ and $\gamma = 0.9$. These values were empirically chosen to provide conservative predicted profiles

that look different and highlight the effects of the custom objective function. Moreover, the plots on the right show the prediction error between the expected output and the outputs obtained with the two sets of parameters.

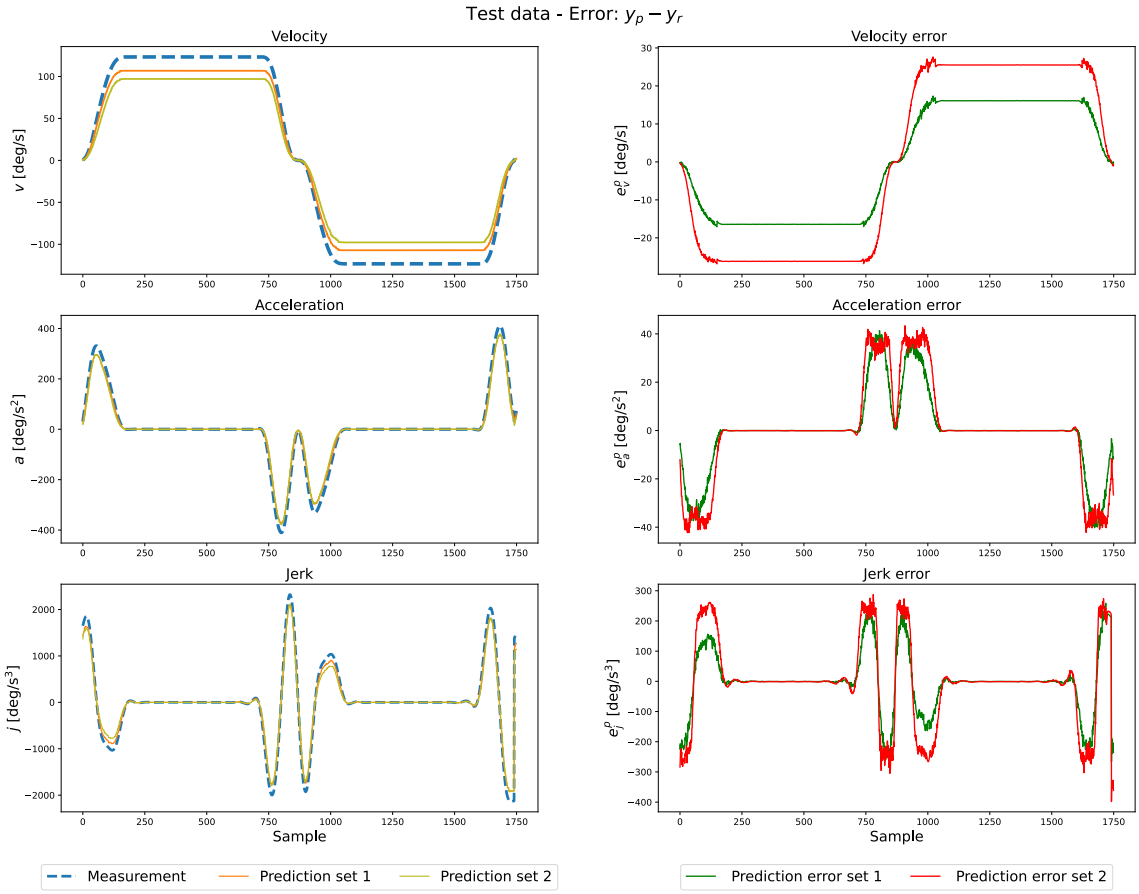

Figure 3-6: Robot joint: A1, Experiment: 6 kg, robot pose: 'conf1'.
Comparison between the expected output and the predicted output using Extreme Gradient Boosting Regressor for two parameter sets of the conservative objective function.
Set 1: ($\alpha = 0.8$, $\gamma = 0.22$), Set 2: ($\alpha = 0.75$, $\gamma = 0.9$)

Additionally, in Figure 3-7 one of the set of parameters used in Figure 3-6 is again utilized. In this case, instead of the plain prediction error $y_p - y_r$, the error $y_p^2 - y_r^2$ is computed. This figure aims to show that suitably chosen parameters for the conservative objective function can lead to the squared error requirement $\hat{y}^2 - y^2 \leq 0$ to be fulfilled. This ensures that the predicted value is always lower than the positive expected value and higher than the negative expected value.

Although the XGBoost did not improve the results of the Random Forest, the capability of using a custom objective function that can influence the direction of the prediction serves as a major benefit.

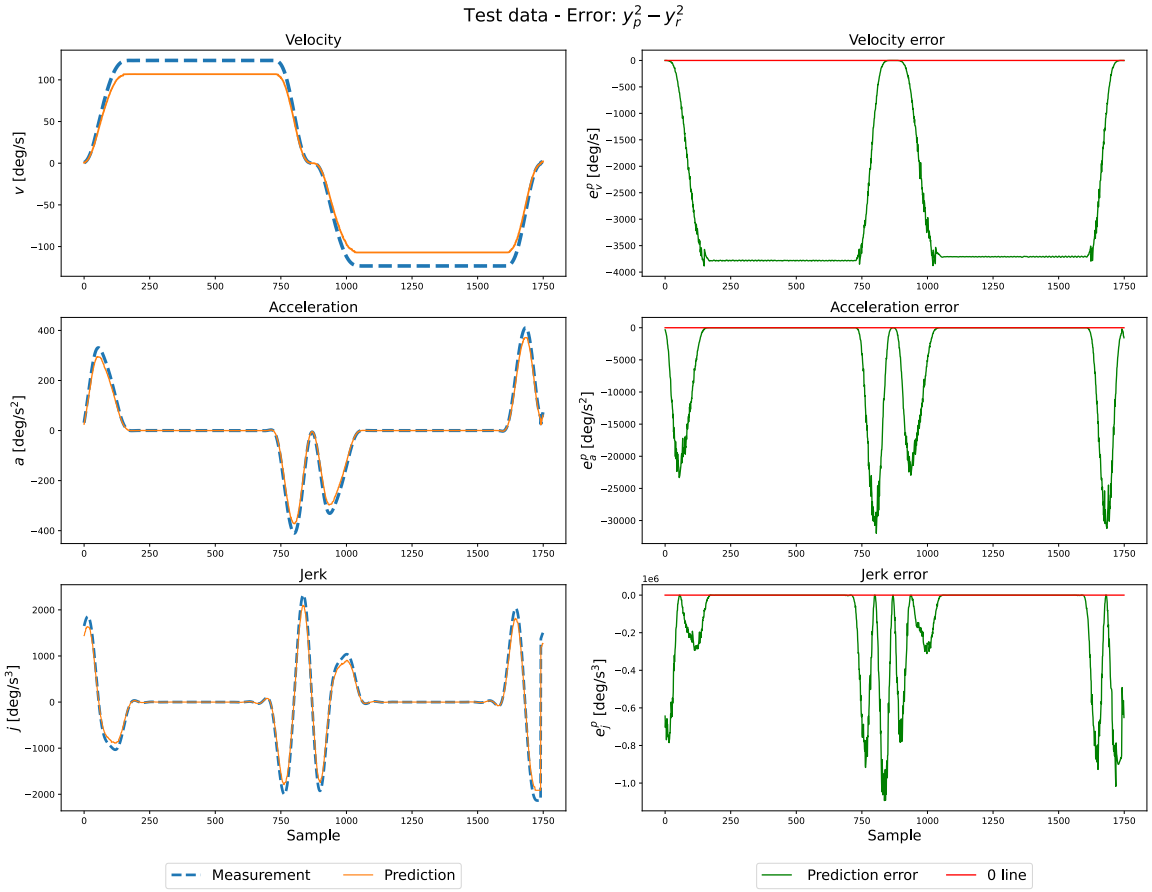

Figure 3-7: Robot joint: A1, Experiment: 6 kg, robot pose: 'conf1'. Comparison between the expected output and the predicted output using Extreme Gradient Boosting Regressor using the conservative objective function and squared prediction error $\hat{y}^2 - y^2$. Set 1: ($\alpha = 0.8$, $\gamma = 0.22$).

3-3 Conclusion

The current chapter introduced the approach for predicting the bounds imposed by the Robot Controller's Limiter block. In this sense, the bounds depicting the maximum performances that the robot can achieve were considered to be represented by the velocity, acceleration and jerk profiles obtained when actuating the robot with the maximum speed level. This supported the identification of a model defining the bounds using previously collected robot data which built the train and test datasets.

Consequently, a mapping between the next bounds and the current robot state and motion parameters had to be identified. For modeling the mapping, multiple approaches were presented. The first method used is the straightforward Linear Regression which benefits of easily interpretable decisions. On the other hand, for obtaining a more complex approach, Random Forest and XGBoost Regressors were introduced.

Lastly, since the approaches typically minimize simplistic objective functions such as squared errors, prediction errors could cause the predicted bounds to exceed the real ones. This aspect

might lead to the robot triggering a controlled stop. In this sense, a custom objective function was defined to ensure more conservative predictions that are more likely to be dynamically feasible.

As a result, since XGBoost proved satisfactory results with both the default and custom objective function, this will be used for further bounds prediction used in the trajectory generation.

Chapter 4

Trajectory generation

The next state velocity, acceleration and jerk bounds, predicted using the approach presented in Chapter 3, are essential for the generation of a dynamically feasible trajectory. As a results, the trajectory generator considers the current robot state and the predicted bounds of the next state and computes the achievable position samples that respects all the bounds, while aiming for minimum execution time. More specifically, the trajectories generated should avoid controlled stops and reach the target position in minimum time. Nonetheless, the resulting trajectories need to be viable positional references for the robot.

To obtain suitable positional references for the robot, several methods were assessed in the context of the current problem. First, an open-source trajectory generator was tested, failing to satisfy the requirements. Further, a trajectory generation approach in the form of an optimization problem was defined. An extensive analysis on the optimization problem results is presented in the current chapter and a longer-horizon optimization problem, meant to solve the shortsightedness of the original solution is evaluated.

4-1 Ruckig

An existing jerk-limited trajectory generation tool, called Ruckig, that provides an open-source library is presented in [17]. This methodology implements an online trajectory generation solution for which the user is allowed to specify the velocity, acceleration and jerk bounds. Based on this, Ruckig will provide a time-optimal profile.

For generating a trajectory between the specified start and target positions, Ruckig had to be adapted. Instead of providing only the maximum and minimum velocity, acceleration and jerk once per experiment, which was limiting the purpose of this work, the bounds were provided for each sample.

Another limitation is that Ruckig requires only strictly positive velocity, acceleration and jerk bounds. Using the given bounds, Ruckig will generate a trajectory assuming symmetrical bounds such as $v_{\min} = -\ v_{\max}$, $a_{\min} = -\ a_{\max}$, $j_{\min} =$ - $j_{\max}$. Considering that the

velocity, acceleration and jerk bounds presented in this work include also negative values, in an attempt to combat this limitation, the absolute values of the computed bounds were provided, although negative bounds were clearly needed for some regions. During these experiments, the assumption was that the combination of the three bounds will enforce the desired behavior. For example, in the deceleration region, characterized by negative acceleration, the prospect was that the decreasing velocity will guarantee the fulfillment of the real acceleration bound.

Unfortunately, the need for negative acceleration and jerk bounds, especially in the deceleration part of the motion, is not enforced by the provided bounds. As a result, the velocity profile starts increasing in the deceleration phase of the motion. This behavior is presented in Figure 4-1 after the 750th sample (provided on the X-axis). This figure provides the comparison between the real position trajectory provided by the robot and the one generated by Ruckig. It is important to mention that in the experiment the bounds were provided according to the robot recording and the number of samples was limited as a result. This motivates the high positional offset at the end of the simulation and the reason for which Ruckig did not aim to continue the simulation until reaching the target position.

For the velocity, acceleration and jerk profiles, Figure 4-1 provides also the bounds that Ruckig had to consider when generating the trajectory. As a result of the constant maximum velocity region, depicted in the plot between samples 150 and 700, the acceleration and jerk bounds provided are almost zero.

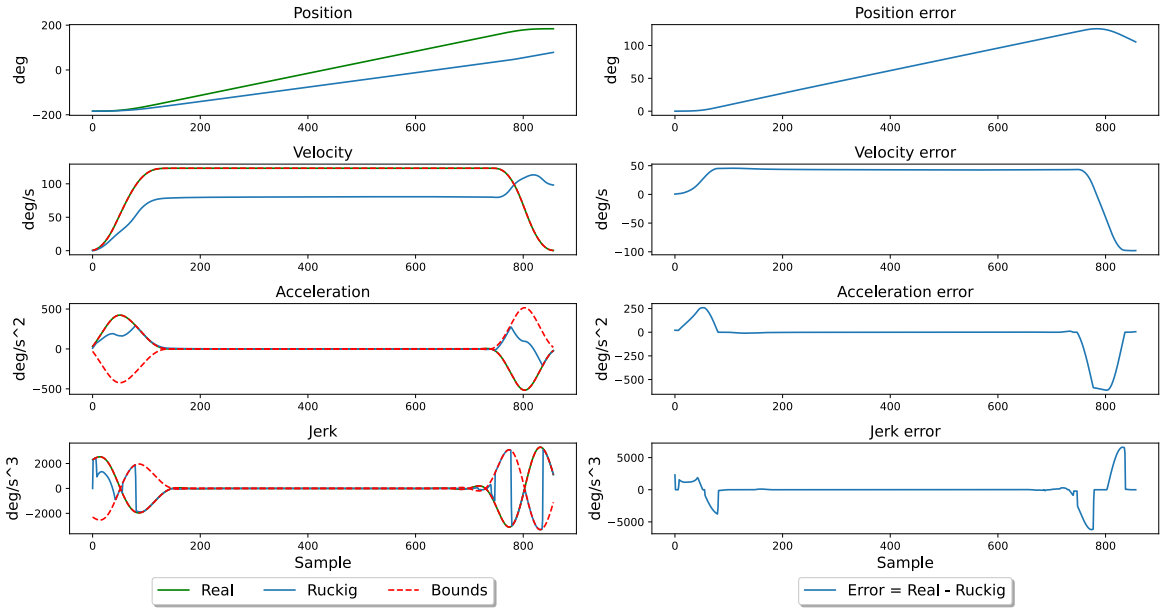

Figure 4-1: Robot joint A1. Experiment: 126kg, robot pose: 'transp'. Trajectory generation using Ruckig

Moreover, the requirements to fulfill all the constraints in the same time, especially the steep velocity increase, lead to an undesired behavior in the acceleration phase of the motion. This is presented in the plot in the first 100 samples of the motion. One can observe that neither the velocity, acceleration or jerk lines computed by Ruckig (blue) overlaps the real robot

recording (green). The limitation on the velocity profile required a low acceleration and jerk. Although the velocity bound increases over time and this change is indicated to Ruckig at every sample, during one iteration Ruckig assumes the velocity bound constant for the entire motion. This will limit the acceleration and jerk causing a lower maximum velocity than in the real scenario.

Considering that Ruckig cannot receive only negative bounds for velocity, acceleration and jerk, adapting Ruckig to the present problem failed to obtain a position profile that could be fed to the robot. As a result, a specialized solution was required.

4-2 Optimization problem with one step horizon

In order to include the velocity, acceleration and jerk bounds in the robot's trajectory, an optimization problem for trajectory generation was proposed. First, the relationship between position and its derivatives is formulated as system dynamics in Subsection 4-2-1. The trajectory generator will use the system dynamics and will consider the velocity, acceleration and jerk bounds as state constraints. These constraints will ensure that none of the bounds are violated, which would lead to a controlled stop otherwise. As presented in Chapter 3, the bounds are predicted independently and satisfying one bound does not guarantee the fulfillment of the other two.

4-2-1 System dynamics

In order to describe the relation between position, velocity, acceleration and jerk as system dynamics, a fourth order integrator system was used. The system input is represented by the fourth derivative of position, called snap ($u = s$) and the state will contain the position, velocity, acceleration and jerk as $x = [\theta \ v \ a \ j]^T$, assuming the typical notation for input as u and state as x. The continuous-time system dynamics in the state-space form is depicted in (4-1). Moreover, the discrete-time system in the state-space form using an arbitrary sampling time Δt is presented in (4-2).

$$\underbrace{\begin{bmatrix}\dot{\theta}\\ \dot{v}\\ \dot{a}\\ \dot{j}\end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\end{bmatrix}}_{A_c}\underbrace{\begin{bmatrix}\theta\\ v\\ a\\ j\end{bmatrix}}_{x} + \underbrace{\begin{bmatrix}0\\ 0\\ 0\\ 1\end{bmatrix}}_{B_c}\underbrace{s}_{u} \tag{4-1a}$$

$$\underbrace{\theta}_{y} = \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}}_{C_c}\underbrace{\begin{bmatrix}\theta\\ v\\ a\\ j\end{bmatrix}}_{x} \tag{4-1b}$$

$$\underbrace{\begin{bmatrix}\theta_{k+1}\\ v_{k+1}\\ a_{k+1}\\ j_{k+1}\end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix}1 & \Delta t & \frac{\Delta t^2}{2} & \frac{\Delta t^3}{6}\\ 0 & 1 & \Delta t & \frac{\Delta t^2}{2}\\ 0 & 0 & 1 & \Delta t\\ 0 & 0 & 0 & 1\end{bmatrix}}_{A} \underbrace{\begin{bmatrix}\theta_k\\ v_k\\ a_k\\ j_k\end{bmatrix}}_{x_k} + \underbrace{\begin{bmatrix}\frac{\Delta t^4}{24}\\ \frac{\Delta t^3}{6}\\ \frac{\Delta t^2}{2}\\ \Delta t\end{bmatrix}}_{B} \underbrace{\frac{\Delta j_k}{\Delta t}}_{u_k} \tag{4-2a}$$

$$\underbrace{\theta_k}_{y_k} = \underbrace{\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}}_{C} \underbrace{\begin{bmatrix}\theta_k\\ v_k\\ a_k\\ j_k\end{bmatrix}}_{x_k} \tag{4-2b}$$

According to the documentation, KUKA RSI provides data with a sampling time of either 4ms or 12ms. Considering that all experiments are executed with a sampling time of 4ms, the trajectory generation will use the same sampling time $\Delta t = 4$ms. In order to avoid confusion in further notation, the system matrices of the continuous dynamics will be denoted as A_c, B_c and C_c, while the matrices for the discrete system will be denoted with A, B and C. As a result, A, A_c denote state matrices and C, C_c depict output matrices. Since the generated trajectories should be provided as input to a physical robot, the discrete system dynamics, (4-2) will be used for trajectory generation.

4-2-2 Optimization problem definition

The optimization problem used in the trajectory generation approach is formulated in (4-3). Here, x_k and u_k depict the state and output at time k.

$$\begin{aligned}
\underset{u_k,\,\epsilon}{\text{minimize}} \quad & Q_1[\theta_{k+1} - \theta_{\text{ref}}]^2 + Q_2[v_{k+1} - v_{\text{ref}}]^2 + S\epsilon + S_u[u_k - u_{k-1}] && \text{(4-3a)}\\
\text{subject to} \quad & x_{k+1} = Ax_k + Bu_k, && \text{(4-3b)}\\
& \underline{x} - \epsilon \leq x_{k+1} \leq \bar{x} + \epsilon, && \text{(4-3c)}\\
& \underline{u} \leq u_k \leq \bar{u}, && \text{(4-3d)}\\
& \epsilon \geq 0, && \text{(4-3e)}\\
& 0 \leq v_{k+1} \leq v_{\text{adm}_{\max}}, && \text{(4-3f)}\\
& 0 \leq A_2^{(i-1)} v_{k+1} \leq v_{\text{adm}_{\max}}, \text{ for } \; i = 2 : N_{\text{steps}} && \text{(4-3g)}
\end{aligned}$$

First, since one of the objectives is to reach the target position in the minimum amount of time, the first term of the cost function ($Q_1[\theta_{k+1} - \theta_{\text{ref}}]^2$) aims to minimize the distance between the next position and the target position.

The second term of the cost function ($Q_2[v_{k+1} - v_{\text{ref}}]^2$) aims to stimulate the acceleration and deceleration phase. More specifically, for the acceleration phase, the v_{ref} parameter will be equal to the rated velocity that depicts the maximum admissible velocity $v_{\text{adm}_{\max}}$ of the actuated joint. According to the KUKA documentation, the objective of the robot controller

is to achieve the rated velocity as well. For the deceleration phase, v_{ref} will become 0 since the focus is for the robot to decelerate as fast as possible.

This switch between the two reference velocities is made after the sample k_{change}, described as the first sample k for which the equations in (4-4) hold. According to these rules, when the velocity and acceleration bounds are decreasing (with 0.5 $\frac{m}{s}$ over the last 5 samples of velocity, and 10 $\frac{m}{s^2}$ over the last 5 samples of acceleration), symbolizing the deceleration region, the reference should not be the rated velocity anymore, but 0, implying that the robot aims to decelerate until stopped. These rules where arbitrarily chosen after extensive experiments and $\bar{v}$ and $\bar{a}$ depict the velocity and acceleration bounds. It is important to mention that this simplification of the velocity change will bot reduce the generality of the solution as k_{change} can be computed online based on the predicted velocity and acceleration bounds.

$$\bar{v}_{k-5} - \bar{v}_k > 0.5 \tag{4-4a}$$

$$\bar{a}_{k-5} - \bar{a}_k > 10 \tag{4-4b}$$

The parameter $v_{\text{adm}_{\max}}$ will be defined according to its joint dependency presented in Table 2-8. According to the KUKA documentation, the manufacturer provides the maximum admissible velocity for each joint. These values are imposed as hard limits for each joint since exceeding the maximum admissible velocity might lead to a controlled stop. However, slight deviations from the provided values were obtained. As a result, this work will use the computed maximum velocity obtained during the time-derivation phase. This will accommodate the profiles computed using Forward Finite Differences.

In addition, the constraint (4-3b) requires the system dynamics to be followed, while (4-3c) depicts the state constraints. The state bounds are provided according to (4-5), where $\bar{v}_{k+1}$, $\bar{a}_{k+1}$, $\bar{j}_{k+1}$ are the predicted next state velocity, acceleration and jerk upper bounds. To guarantee no sudden change in the profiles behaviour, the lower bounds for the next state were set, based on empirical knowledge, as $\underline{v}_{k+1} = 0$, $\underline{a}_{k+1} = 0.8\bar{a}_{k+1}$, $\underline{j}_{k+1} = 0.9\bar{j}_{k+1}$.

Since hard state constraints lead to infeasibility in most of the situations, these were set as soft constraint with the help of the slack variable ϵ. To permit the velocity, acceleration and jerk bounds to slightly exceed the predicted bounds, the positiveness requirement in (4-3e) was applied and high ϵ values were penalized by the third term in the cost function. Additionally, θ_{start} and θ_{ref}, in (4-5), depict the starting and reference position, respectively. In order to aim for reaching the motion's goal, the reference position will be set to the target position of the motion.

$$\underline{x} = [\theta_{\text{start}},\ \underline{v}_{k+1},\ \underline{a}_{k+1},\ \underline{j}_{k+1}]^T \tag{4-5a}$$

$$\bar{x} = [\theta_{\text{ref}},\ \bar{v}_{k+1},\ \bar{a}_{k+1},\ \bar{j}_{k+1}]^T \tag{4-5b}$$

Further, (4-3d) provides the input constraints. Using the jerk information, the bounds for the input are set as in (4-6) since the goal is not only to limit the jerk, but also the change in jerk. As for the state bounds presented above, $\underline{u}$ and $\bar{u}$ depict the lower and upper input bound, respectively.

$$\underline{u} = -\frac{\bar{j}_{k+1} - \bar{j}_k}{\Delta t} \tag{4-6a}$$

$$\bar{u} = \frac{\bar{\dot{j}}_{k+1} - \bar{\dot{j}}_k}{\Delta t} \tag{4-6b}$$

Finally, the last two constraints depicted by (4-3f) and (4-3g) aim to limit the velocity between 0 and the maximum admissible velocity. While (4-3f) provides the constraint for the next velocity only, the constraint in (4-3g) aims to limit the velocity values in each of the following $N_{\text{steps}} - 1$. The lower bound to 0 is imposed since the robot should keep the motion direction. Nonetheless, the velocity for the next N_{steps} steps is replaced by an approximation of velocity in the absence of inputs. This explains the multiplication with $A_2^{(i-1)}$, the second row of the matrix A to the power of $(i-1)$.

4-2-3 Simulation results

For the initial tests of the optimization problem, the time-derivatives computed for the position measurements collected during several experiments were used as velocity, acceleration and jerk bounds. In a further step, predicted bounds for velocity, acceleration and jerk should be used in the constraints instead. The decision to use the computed position derivatives instead of the predictions in the beginning aims to limit the possible error sources. Although this approach was briefly presented in Section 2-2 for analyzing the time-derivation methods, the results presented in this chapter aim to asses the quality of the trajectory generation approach.

Successful simulations

A successful simulation of the trajectory generation, displaying joint A1 aiming to determine the motion between the start $\theta_{\text{start}} = -180°$ and target $\theta_{\text{ref}} = 180°$, is depicted in Figure 4-2. The figure presents a comparison between the real position recorded using KUKA RSI and the position generated using the optimization problem defined in (4-3a) - (4-3g). Moreover, the plot shows the relation between the time-derivatives of the recorded position and the velocity, acceleration and jerk determined by the optimization problem. In addition, the figure presents the evolution of the system input.

To facilitate the comparative analysis, the figure also provides the errors depicting the difference between the recorded position and its time-derivatives (denoted as "Real" in the plot and marked with green) and the evolution of the state x_g used in the optimization problem (denoted as "Generated" and marked with dark blue). The input rate is also provided and marked with orange.

Lastly, the figure provides the state and input constraints denoted as "Bounds" and marked in red. An important note is that the state constraints (presented in the position, velocity, acceleration and jerk plots) are provided as soft constraints. In this sense, state constraints violation is permitted but greatly penalized. On the other hand, the input constraints are formulated as hard constraints and no violation is allowed.

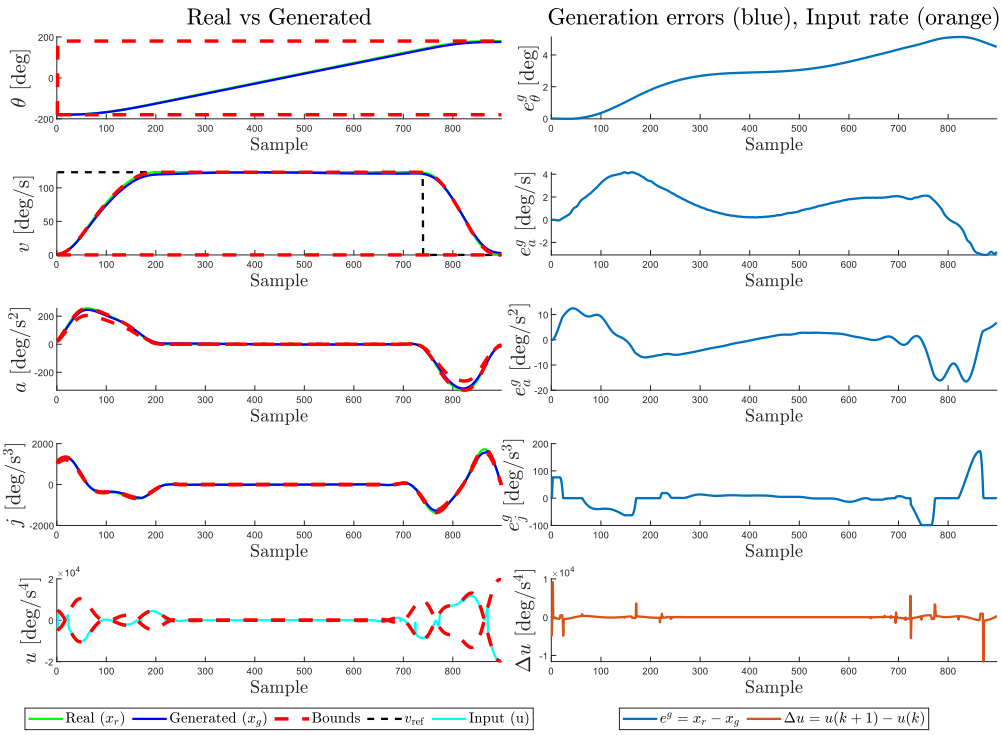

Figure 4-2: Robot joint: A1, Robot configuration: conf1, Payload: 126kg
Trajectory generation using the one-step optimization problem

For consistency, the plot structure used for Figure 4-2 will be used to depict all simulations presented in the current section.

In Figure 4-2, it can be seen from the plot that the position error accumulates throughout the motion, but the final offset is minor. This offset will define the final point reached by the robot to be $\theta_{\text{final}} = \theta_{\text{ref}} - \theta_{\text{offset}}$. Moreover, in the region where the robot achieves maximum velocity, zero acceleration and zero jerk, the velocity, acceleration and jerk errors will gravitate around zero.

Another successful simulation displaying joint A2 is presented in Figure 4-3. The figure shows the results for the one-step optimization algorithm that generates a trajectory for joint A2 between $\theta_{\text{start}} = -135°$ and $\theta_{\text{ref}} = -10\ °$. In this case, the motion is shorter than the one depicted above for A1, leading to a shorter period of constant velocity. The shorter simulation time might be the reason for a smaller positional offset at the end of the simulation compared to the A1 simulation presented in Figure 4-2.

To demonstrate the functionality of the trajectory generation approach for less typical motion profiles, another experiment depicting joint A2 is presented in Figure 4-4. For this motion, the robot does not even reach the maximum admissible velocity anymore and the profiles are composed only by acceleration and deceleration phases, lacking the constant velocity region presented in both simulations above in Figure 4-2 and Figure 4-3. The positional offset in case of this simulation is reduced, being also explained by the short simulation time and by the similarity between the generated velocity, acceleration and jerk profiles and their correspondents computed using the time-derivation approach.

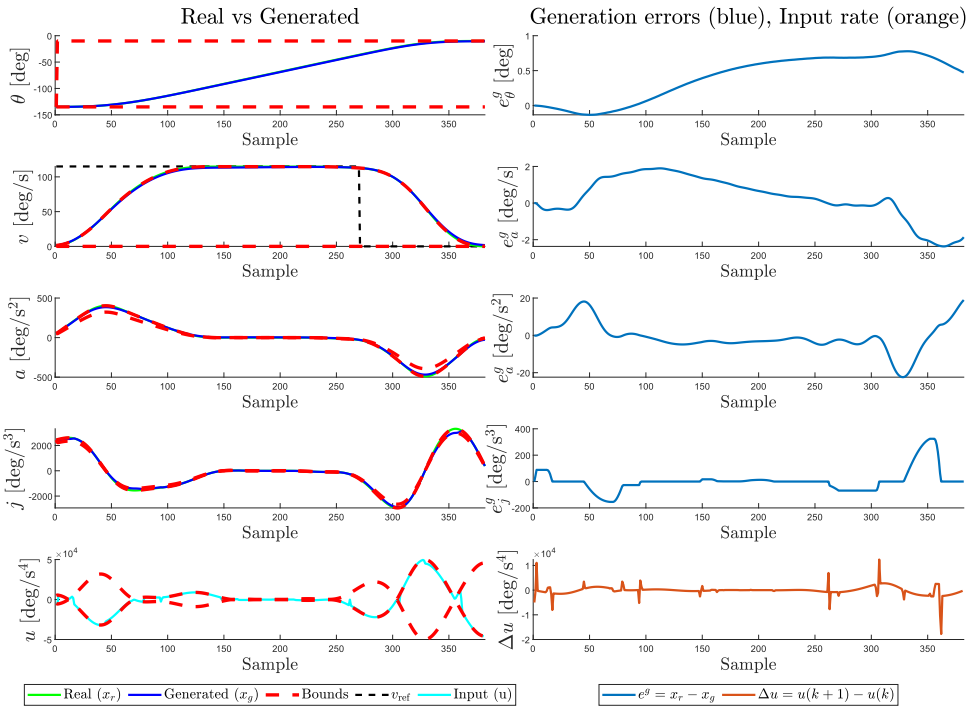

Figure 4-3: Robot joint: A2, Robot configuration: conf1, Payload: 0kg
Trajectory generation using the one-step optimization problem

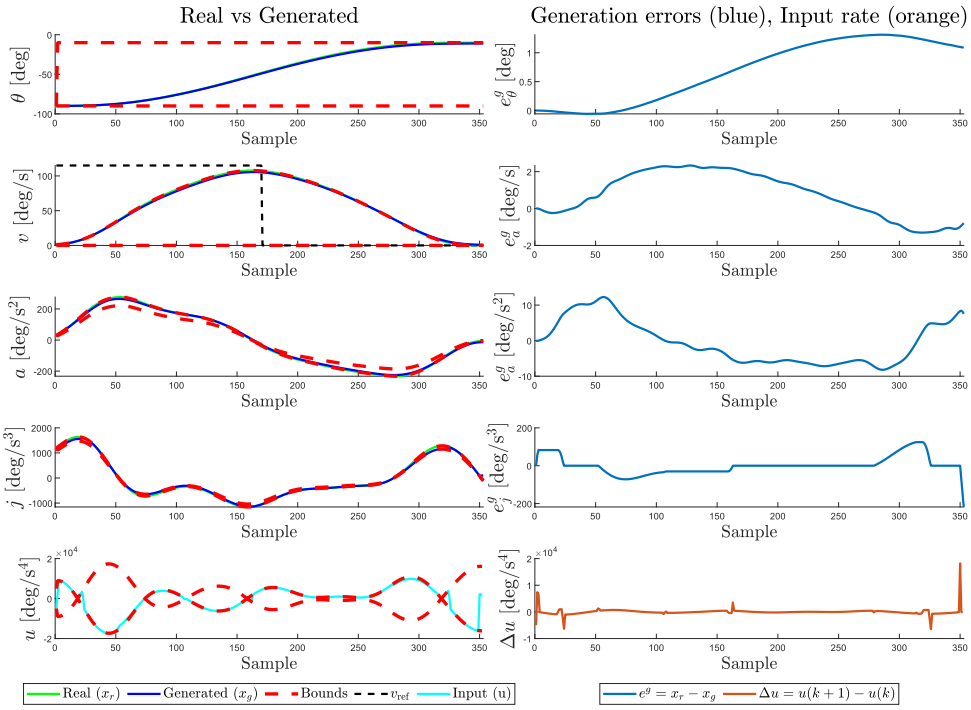

Figure 4-4: Robot joint: A2, Robot configuration: conf2, Payload: 176kg
Trajectory generation using the one-step optimization problem

Identified problems

Although the simulations presented above depict satisfactory results, several simulations prove the contrary, suggesting the lack of algorithm robustness. Motivated by this, the current subsection will present the issues of the optimization problem along with illustrative scenarios.

The first issue exemplified in Figure 4-5 arises from the non-negativity constraint on velocity, depicted by the left side of (4-3f) and (4-3g). These constraints impose positive velocities for the next N_{steps} steps. However, notice that the latter is based on a velocity approximation in the absence of inputs. In order to ensure that the hard constraint is not violated, the built-in planner might need to apply a sudden change in snap.

In the simulation presented in Figure 4-5, the corrective action, meant to respect all the imposed constrains, takes place in the last samples of the simulation. The sudden jump introduced in the input will lead to an unusual behavior in jerk and acceleration. This is exemplified by a high deviation between the generated (dark blue) and the real (green line) jerk. Additionally, the deviation is highlighted by the abrupt drops in the acceleration and jerk error plots that reach significant negative values. Although this did not lead to an unsuccessful simulation, the sudden change in input might lead to a controlled stop when applied to the robot.

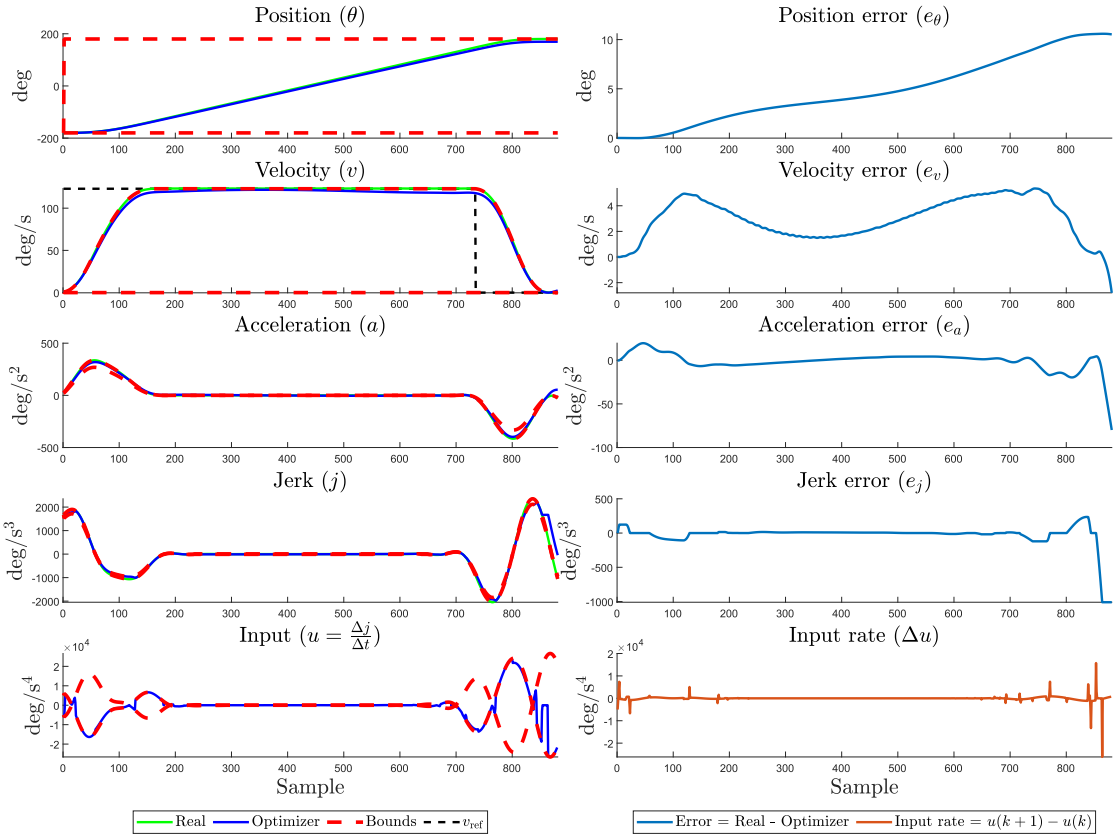

Figure 4-5: Robot joint: A1, Robot configuration: conf1, Payload: 0kg
Trajectory generation using the one-step optimization problem

In some less fortunate cases than the simulation above, the non-negativity constraint led

to numerical problems. Such example is provided in Figure 4-6. Experiments showed that when the non-negativity constraint is removed, the simulation presented in Figure 4-6 will become feasible, with the velocity line going below zero. This is not desired for the real robot experiments since it might lead to controlled stop or unexpected behaviour.

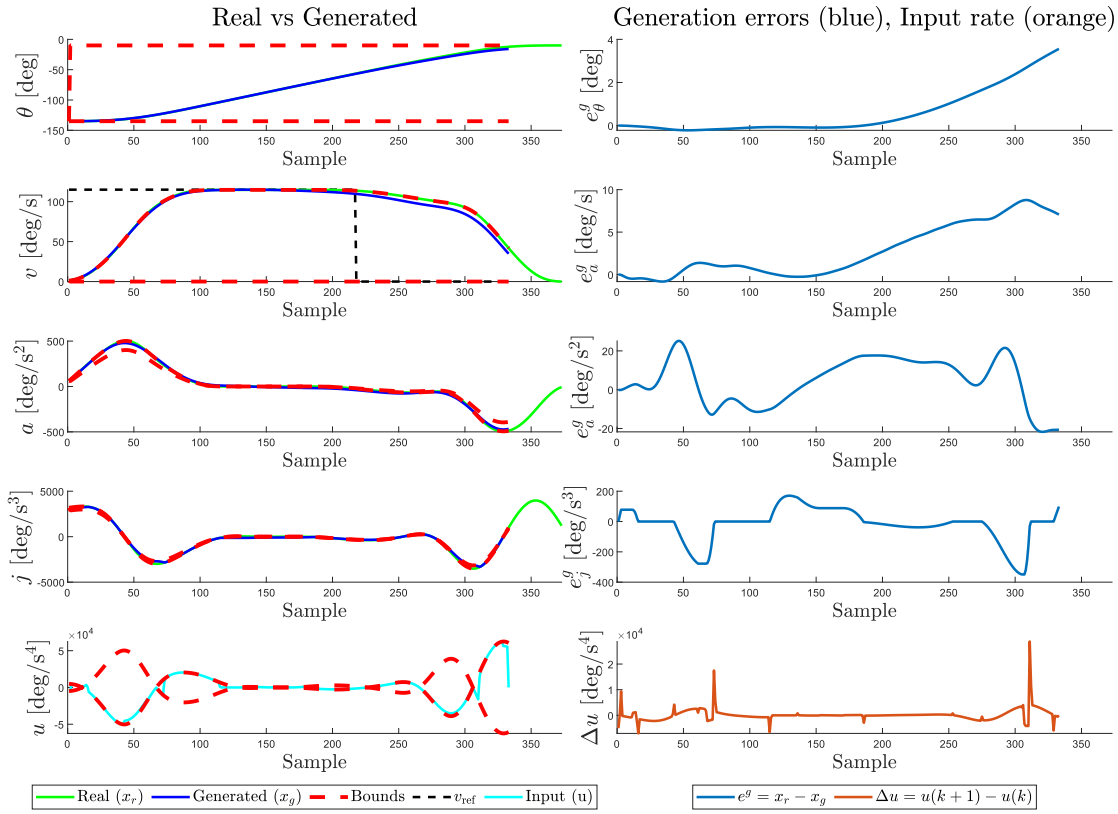

Figure 4-6: Robot joint: A2, Robot configuration: transp, Payload: 36kg
Trajectory generation using the one-step optimization problem

The next two scenarios illustrated in Figure 4-7 and Figure 4-8 occur due to the imposed velocity upper bound, depicted in right side of (4-3f) and (4-3g). The main characteristic leading to a difference between the two scenarios is represented by the input bounds. Depending on the motion region in which the velocity threatens to violate the upper velocity bound in a N_{steps} horizon, the input bounds might permit a corrective action or not. More specifically, if the input constraints allow an absolute snap different than 0, an input can be applied to reach a position that ensures velocity, acceleration and jerk values that do not violate the bounds. In the case in which the hard velocity constraint threatens to be violated, the input constraints would ideally permit a negative snap that will move the velocity away from the maximum admissible bound. In this sense, while Figure 4-7 presents a simulation in which a corrective action is possible, in Figure 4-8 the input bounds do not permit a correction.

In Figure 4-7 the upper velocity bounds is violated in the constant velocity region which implies tight input bounds. In this motion region, the input bounds are too tight, aiming for negligible snap meaning no change in jerk. This requirement is understandable since negligible snap will imply almost no change in jerk. When the jerk is also negligible (as in

this case) this will imply almost no change in acceleration. Subsequently, no change in the negligible acceleration will also imply almost no change in the velocity, leading to a constant maximum velocity (since this region is marked by maximum velocity). When the velocity constraint tends to be violated in the negligible snap region, any corrective action that the trajectory generator can apply will be too small to avoid bound violation. This results in problem infeasibility.

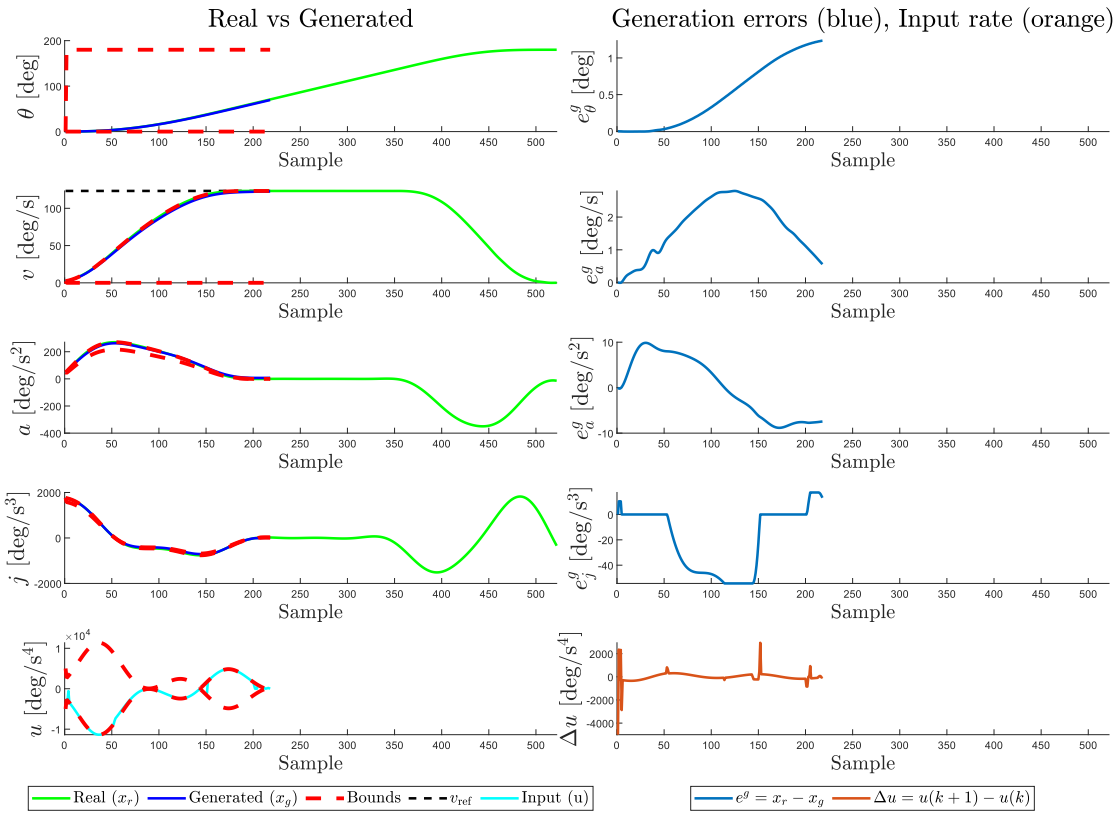

Figure 4-7: Robot joint: A1, Robot configuration: conf2, Payload: 120kg
Trajectory generation using the one-step optimization problem

On the other hand, Figure 4-8 presents a scenario in which the input bounds permit a sudden change in the input in order to avoid violating the velocity bound. Although it is desired to fulfill all the imposed constraints, the corrective action will lead to a decrease in jerk and acceleration that will lead to a decreasing negative acceleration. Considering the system dynamics, the negative acceleration will maintain a decreasing velocity that will hit the lower velocity bound after several time samples. Since the profiles presented in Figure 4-8 characterize a deceleration behavior, the position offset will grow. This could be considered an unsuccessful simulation in which the built-in controller is never able to compensate for the corrective action applied around the 200^{th} sample. It is important to mention that, although in multiple simulations the built-in trajectory generator avoids upper velocity bound violation, not all of them present the behavior showed in Figure 4-8, but lead to a successful simulation.

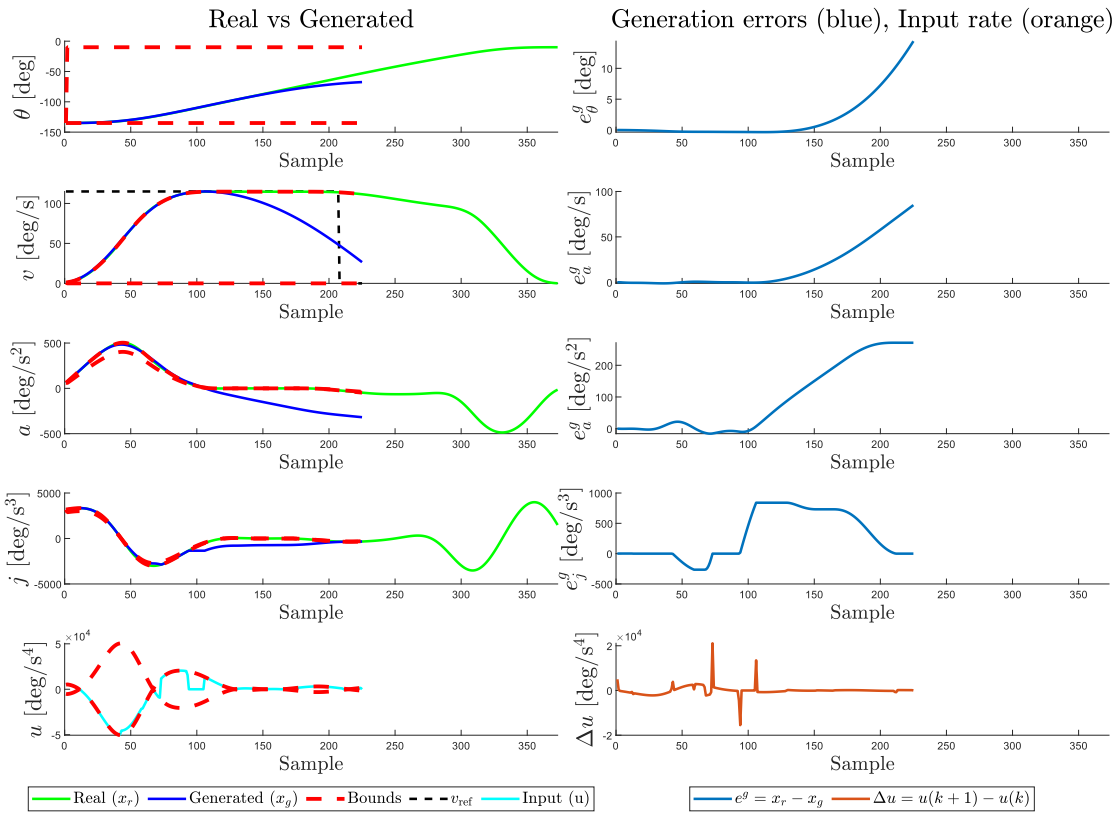

Figure 4-8: Robot joint: A2, Robot configuration: transp, Payload: 0kg, $Q_2 = 10^3$
Trajectory generation using the one-step optimization problem

Results overview

In order to examine the capabilities of the optimization problem with a one-step horizon, a thorough analysis was made. This included all the experiments for which position data was collected from the robot. Using the Forward Finite Differences time-derivation method, the velocity, acceleration and jerk profiles were retrieved. Further, these profiles were used as the velocity, acceleration and jerk bounds in the optimization problem. For both A1 and A2, simulation overviews will be presented in Tables 4-1 and 4-2, respectively. Each table considers multiple simulations for each motion, varying one parameter of the cost function. Since Q_2 enforces the rate for following the velocity reference, various values lead to different performances of the built-in trajectory generator. As a result, for each motion, five simulations were executed for Q_2 in $[1, 10, 10^2, 10^3, 10^4]$.

As described before, multiple motion parameters were varied during the robot experiments, but the most relevant ones are still the payload and configuration. As a result, the data presented in the tables below will be described by these two motion parameters. Moreover, each configuration is defined by the $\Delta\theta$ parameter representing the difference between the start and target position, $\Delta\theta = \theta_{\text{ref}} - \theta_{\text{start}}$.

From a qualitative perspective, the tables show the feasibility of the simulations. The red x symbols mark the set of experiments in which all simulations were unsuccessful, reaching

problem infeasibility in one of the scenarios described above. The numbers mark successful simulations, while the blue dashes mark missing experiments from the payload and configuration grid.

From the quantitative perspective, the values included in the tables describe the average positional offset at the end of the simulation. Out of the five simulations considered for each experiment, using different values of the parameter Q_2, only the successful simulations were considered in the computation of the average offset of each motion. In the table, the ratio of successful simulations for each experiment is included between parentheses. The position offset at the end of the simulation represents a relevant metric since any difference between the real and the simulated profiles lead to discrepancies in the position evolution. Consequently, the ideal value of this parameter should be 0, showing that the target position was exactly reached and, eventually, the velocity, acceleration and jerk profiles were perfectly followed.

Table 4-1 depicts the average position offsets at the end of each simulation for joint A1. One can observe that all simulations corresponding to configuration "conf2" reached infeasibility. Moreover, the position offset for configuration "shorter" seem to be lower than configurations "transp" and "conf1". The main cause for this difference could be the shorter trajectories executed using the robot pose "conf1". The rather reduced motion distance did not allow the position offset to accumulate throughout an extended period of time. Furthermore, the data presented in the table does not suggest any influence of the payload weight parameter on the position offset.

		Configuration			
		transp	**shorter**	**conf1**	**conf2**
		$\Delta\theta = 368°$	$\Delta\theta = 180°$	$\Delta\theta = 360°$	$\Delta\theta = 180°$
Payload [kg]	0	11.73 (3/5)	3.83 (4/5)	10.57 (3/5)	x
	6	11.84 (4/5)	3.80 (4/5)	7.11 (3/5)	x
	16	11.38 (3/5)	-	x	-
	36	8.42 (4/5)	3.55 (4/5)	x	x
	56	10.53 (3/5)	-	x	-
	76	10.20 (3/5)	3.66 (4/5)	x	x
	106	x	-	x	-
	126	11.47 (3/5)	3.88 (3/5)	4.50 (3/5)	x
	146	9.90 (3/5)	-	5.23 (3/5)	-
	166	11.70 (3/5)	-	5.23 (3/5)	-
	176	10.03 (3/5)	4.05 (3/5)	6.95 (3/5)	x

Table 4-1: Robot joint: A1, Mean absolute position offset at the end of the simulation and number of succesful simulations. Simulations for Q_2 in $[1, 10, 10^2, 10^3, 10^4]$

Table 4-2 depicts similar experiments overview for joint A2. Compared to the results for joint A1, it is harder to determine a relation between the motion distance $\Delta\theta$ and the average positional offset since the motion range of this joint is rather limited.

		Configuration			
		transp	**shorter**	**conf1**	**conf2**
		$\Delta\theta = 125°$	$\Delta\theta = 60°$	$\Delta\theta = 125°$	$\Delta\theta = 80°$
Payload [kg]	0	0.48 (3/5)	x	0.53 (3/5)	2.21 (4/5)
	6	0.67 (3/5)	x	0.49 (3/5)	2.07 (4/5)
	16	0.98 (3/5)	-	0.32 (4/5)	-
	36	x	x	0.46 (1/5)	0.81 (3/5)
	56	x	-	x	-
	76	x	x	1.61 (2/5)	0.25 (3/5)
	106	x	-	1.72 (4/5)	-
	126	x	x	2.74 (3/5)	1.29 (4/5)
	146	x	-	1.91 (4/5)	-
	166	x	-	1.62 (4/5)	-
	176	x	x	1.93 (4/5)	2.53 (5/5)

Table 4-2: Robot joint: A2, Mean absolute position offset at the end of the simulation and number of succesful simulations. Simulations for Q_2 in $[1, 10, 10^2, 10^3, 10^4]$

4-3 Optimization problem with a longer receding horizon

Considering that the one-step optimization problem presented above will suffer from short-sightedness, a receding horizon optimization problem was developed. The option to choose the length of the receding horizon might allow the generator to better avoid violating the velocity bound. Specifically, using a sufficiently long horizon, the optimization problem might provide improved input actions by covering multiple samples in advance. In the proposed approach, only the first input will be applied each sample in the simulation, in a receding horizon manner.

The receding horizon approach is presented in (4-7a) - (4-7f). Being similar to the optimization problem defined for a one step horizon, the same system dynamics introduced in (4-2) will be used. In terms of cost function, the only change brought by the longer horizon is the sum over N steps introduced in (4-7a).

$$\underset{u(k),\,\epsilon}{\text{minimize}} \quad \sum_{k=1}^{N} Q_1[\theta_{k+1} - \theta_{\text{ref}}]^2 + Q_2[v_{k+1} - v_{\text{ref}}]^2 + S\epsilon(k) + S_u[u(k) - u_{k-1}] \tag{4-7a}$$

$$\text{subject to} \quad x_{k+1} = Ax(k) + Bu(k), \qquad k = 1..N, \tag{4-7b}$$

$$\underline{x}_{k+1} - \epsilon(k) \leq x_{k+1} \leq \bar{x}_{k+1} + \epsilon(k), \qquad k = 1..N, \tag{4-7c}$$

$$\epsilon(k) \geq 0, \qquad k = 1..N, \tag{4-7d}$$

$$\underline{u}(k) \leq u(k) \leq \bar{u}(k), \qquad k = 1..N, \tag{4-7e}$$

$$0 \leq v_{k+1} \leq v_{\text{adm}_{\max}}, \qquad k = 1..N \tag{4-7f}$$

Furthermore, (4-7c) presents the state constraints imposed. In this case, the constraints are defined for all N steps and the state bounds defined above as $\bar{x}_{k+1}$ and $\underline{x}_{k+1}$ will be considered constant for the entire horizon. As before, the ϵ terms added both in the constraint and the

cost function (in the term $S\epsilon(k)$) support the definition of this requirement as a soft constraint. As a result, the violation of the state bounds will be heavily penalized by the S cost but will not lead to infeasibility.

The last two constraints are defined as before. Specifically, both the input and the velocity limitations will be considered constant for the N-steps horizon. Since a longer horizon is used, the N steps velocity constraint will not be applied in the absence of input anymore.

Results overview

The optimization problem using a longer receding horizon fails to obtain a solution in most of the cases. Specifically, the optimization problem becomes unfeasible due to various limitations similar to the ones presented for the one step solution. For brevity, plots exemplifying the simulation results of the longer receding horizon approach were avoided.

On the other hand, one of the main limitations of the receding horizon approach is represented by the constant bounds for velocity, acceleration and jerk in the acceleration and deceleration regions. When the problem has to compute N inputs and also ensure that none of the constraints are violated for the entire horizon of N steps, the built-in controller was able to compute rather conservative input values that did not favor minimum-time trajectories.

4-4 Trajectory generation validated on the robot

To validate the trajectory generation approach developed in the current section, some tests on the KUKA robot were required. In this sense, several trajectories generated using the one step horizon approach were provided as input to the KUKA robot. More specifically, the generated position information was sent to the robot using the approach introduced in Section 2-1-2 and the actual position values reached by the robot were collected using KUKA RSI as in the procedure mentioned in the same section.

The dynamic bounds were defined using velocity, acceleration and jerk profiles computed from position measurements collected from the KUKA robot. The definition of bounds in this manner, allowed the validation of the trajectory generation itself, without introducing additional error sources.

The general validation experiments executed on the KUKA robot aimed to answer to the following questions:

- Are all velocity, acceleration and jerk bounds imposed by the robot controller fulfilled?
- Can the trajectory generation provide position samples achievable by the robot in the 4ms sampling time?
- Do the observed spikes in the input rate cause a controlled stop?

During the experiments, no controlled stop was reached while executing an entire trajectory. The lack of controlled stop actions indicated that all robot controller limits were satisfied and

that the position samples could be tracked by the robot. Moreover, no issue was caused by the rather varying snap symbolizing the system's input in the trajectory generation problem.

A successful robot test for joint A1 is presented in Figure 4-9. The upper plot provides information regarding the original recorded position (blue), the position obtained using the trajectory generation approach (red) and the recording of the trajectory executed by the robot when it receive the generated trajectory as input (yellow).

For a clearer comparison, the errors between the three trajectories are provided. Relative to the original trajectory measurement, the blue line in the lower plot depicts the difference in the generated trajectory, while the red line depicts the error obtained by the robot recording following the generated trajectory as input. Finally, the yellow line depicts the difference between the input (generated trajectory) and output (robot position recording) of the robot during the experiment. As described in Chapter 2, the input and output are not aligned in the recording file due to a lag of typically 28 ms. As a result, for presenting the errors in Figure 4-9, the trajectories were aligned in the plot to minimize the recording error ($\theta_g - \theta_r$).

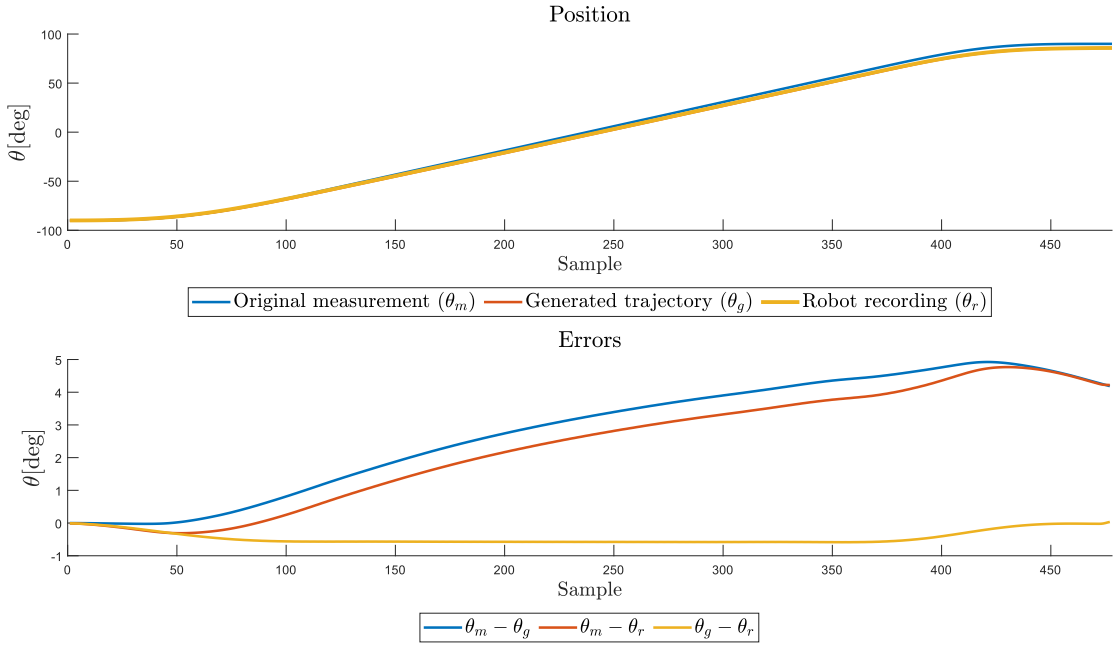

Figure 4-9: Robot joint: A1, Robot configuration: shorter, Payload: 0kg
Recording of generated trajectory executed on the KUKA robot.

Figure 4-10 provides another similar robot test, but actuating joint A2 with the robot carrying no payload. Moreover, Figure 4-11 depicts a similar motion with a payload of 70kg. One can observe that the robot obtains similar performances independent of the robot payload.

However, as observed from simulations as well, the generation error at the end of the simulation in case of joint A2 is smaller than for joint A1 due to the shorter motions leading to a lower accumulated error. Moreover, the yellow line in the lower plot depicting the recording error shows the the robot closely followed the input trajectory.

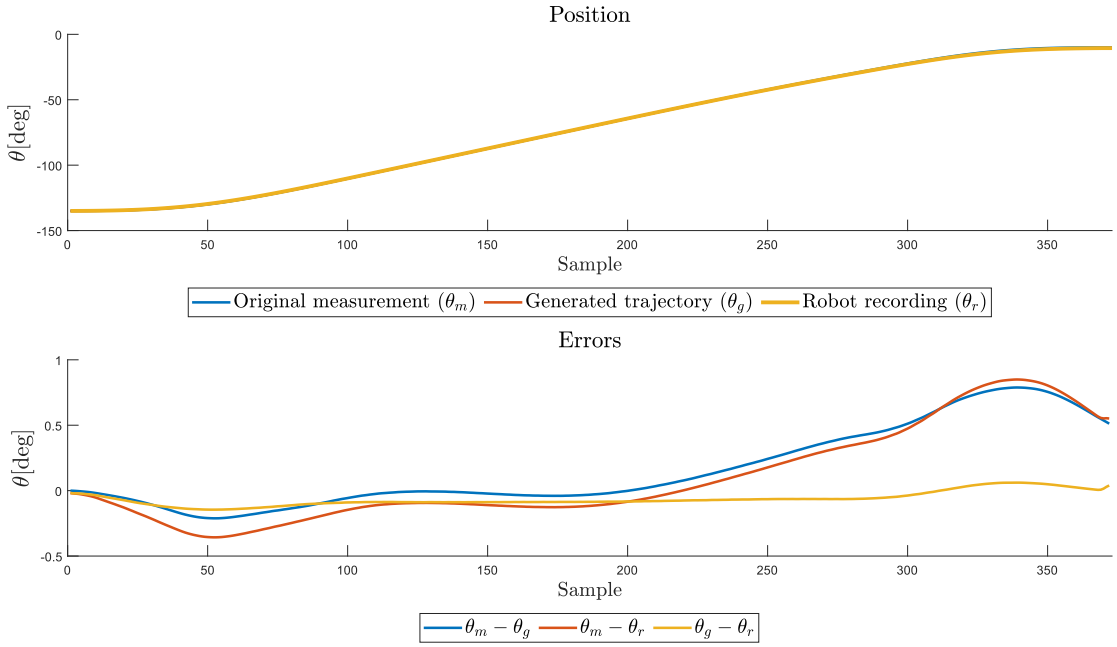

Figure 4-10: Robot joint: A2, Robot configuration: transp, Payload: 0kg
Recording of generated trajectory executed on the KUKA robot.

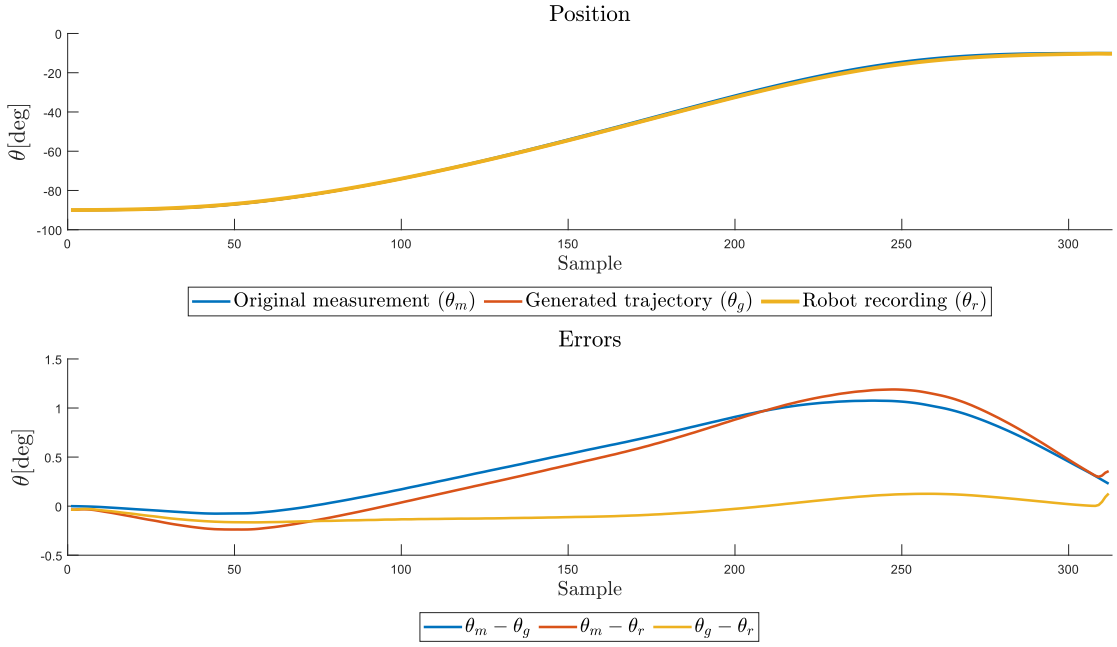

Figure 4-11: Robot joint: A2, Robot configuration: conf2, Payload: 76kg
Recording of generated trajectory executed on the KUKA robot.

Using the robot experiments provided above, one can conclude that the robot is able to follow the trajectory generated using the one step horizon approach. Although the generated

trajectories present differences from the original ones, when the trajectory is provided as robot input, the KUKA robot closely follows the generated trajectory. Moreover, the generated trajectories do not violate any of the limitations imposed by the robot controller and do not lead to a controlled stop.

4-5 Conclusion

The current chapter covered several methods for trajectory generation given a set of velocity, acceleration and jerk bounds. The first approach considered for trajectory generation is Ruckig, an open source tool for generating jerk-limited trajectories. Unfortunately, Ruckig is not a viable option for the current work since it does not allow negative acceleration or jerk bounds.

For tackling the specific trajectory generation problem of this work, a tailored trajectory generation approach was proposed. This uses given velocity, acceleration and jerk bounds and a forth-order integrator system dynamics that simulate the relationships between the position and its time-derivatives, to compute the next trajectory point (with a sampling time of 4ms). Although the trajectory generation approach lacks robustness and several scenarios proved unfeasible, the proposed method was able to generate trajectories for specific motions.

To avoid short-shortsightedness problems caused by the one step horizon approach, a longer receding horizon method was tested. However, this method did not improve the trajectory generation suffering from the lack of velocity, acceleration and jerk information for an N-step horizon.

Finally, the trajectory generation with one step horizon was tested on the real robot and the recorded positions were compared to the original motions. The robot tests avoiding controlled stops aided to conclude that the trajectory generation was able to provide feasible robot trajectories. The main significant drawback of the trajectory generation approach is still represented by the positional offset at the end of the experiment. This might imply that the trajectory generation is slightly more conservative than the robot controller and it will require more time to reach the final point in the trajectory.

Chapter 5

Trajectory generation with predicted bounds

In Chapter 3, the framework for predicting the velocity, acceleration and jerk was described. Further, the optimization algorithm for generating minimum-time, dynamically feasible trajectories given the velocity, acceleration and jerk bounds was introduced in Chapter 4. Merging these two topics, the goal is to generate minimum-time dynamically feasible robot trajectories using the predicted bounds for the next state's velocity, acceleration and jerk. This matches the final goal of the current work, enclosing a method to provide dynamically feasible trajectories that encompass the robot controller behaviour that limits the acceleration and jerk based on multiple motion and tool parameters.

In terms of trajectory generation, the chapter covers two main strategies for the state update. This split was made to facilitate the identification of error sources. Initially, the bounds are predicted based on real robot measurements for each sample in the artificial state update approach. This way, possible prediction or trajectory generation errors could be identified and error drift minimized. Further, a real state update is introduced and the bounds are predicted based on the states computed by the trajectory generator. This represents the final solution that only receives motion related information and the initial robot state and aims to reach the target position in minimum time, without reaching a controlled stop.

5-1 Artificial state update

As mentioned above, before introducing the end-to-end solution, one intermediary step was taken. Its purpose was to isolate the possible error sources. For this scenario, the bounds are predicted based on robot's position measurements and its time-derivatives, defining the current robot state x_k. This method is graphically represented by the diagram in Figure 5-1 and will be named artificial state update.

Based on Figure 5-1, one can notice the distinction between the measured states, the state bounds and the generated trajectory. More specifically, θ_r denotes the recorded robot position,

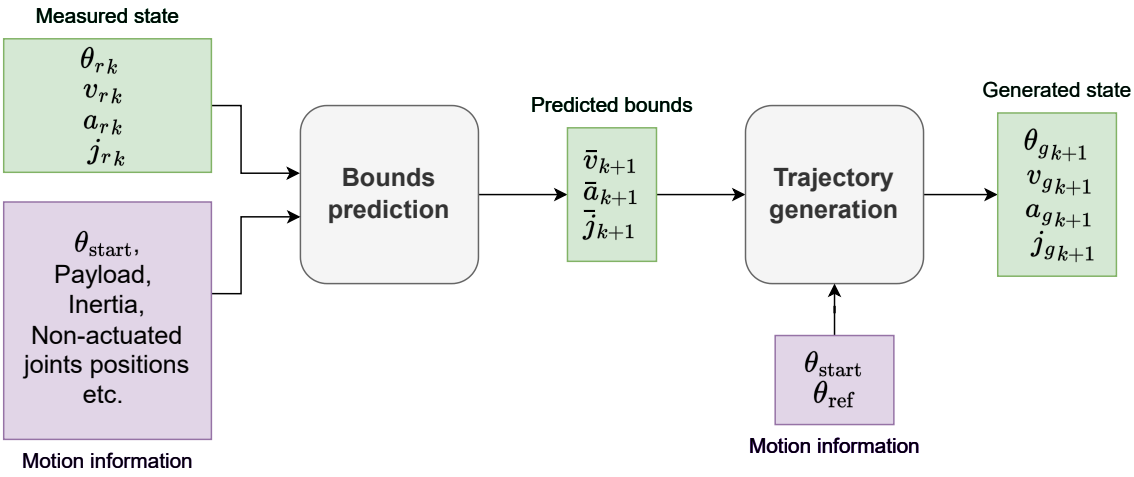

Figure 5-1: Diagram of Trajectory Generation with predicted bounds using artificial state update

while v_r, a_r and j_r designate its time derivatives. Further, the output of the built-in trajectory generator is denoted as $x_g = [\theta_g \;\; v_g \;\; a_g \;\; j_g \;]^T$ and the velocity, acceleration and jerk bounds are represented by $\bar{v}$, $\bar{a}$ and $\bar{j}$.

It it important to mention that in Figure 5-1, the green blocks provide data that is updated every sample, while the purple blocks define information that is constant and motion specific. In this sense, one can observe that the prediction block receives as input the current robot state that is updated every time sample and motion specific data such as the starting position or payload weight, that is provided as constant input to the predictor. The output of the prediction block is represented by the forecasted velocity, acceleration and jerk bounds of the next time sample.

As stated above, the main difference between this approach and the final solution is represented by the update of the current robot state. As shown in the diagram, the current robot state in the artificial state update approach is actually read from previous robot measurements. More specifically, the state x_r is composed of position measurements θ_r collected during previous robot experiments and its time-derivatives v_r, a_r and j_r.

Using the predicted bounds and the starting and target positions (also constant motion specific information and marked by a purple block), the built-in trajectory generator will compute the next point of the trajectory. In the diagram, the generated state is marked as $x_{g_{k+1}} = [\theta_{g_{k+1}} \; v_{g_{k+1}} \; a_{g_{k+1}} \; j_{g_{k+1}}]^T$. The trajectory generator will aim for both maximum performances and dynamical feasibility.

Simulation results

Using the approach presented in Figure 5-1, several simulations are displayed. It is important to remind that for each simulation, the reference motion was not included in the training set of the regressors, and only used as test set.

Figure 5-2 presents the trajectory generation result using the recorded position and its time

derivatives as input to the bounds predictor. In this case, the α and γ parameters used for the prediction cost function (defined in Chapter 3) were set to 0 and 1.5, respectively. This means that the conservativeness was omitted and the most accurate prediction was preferred.

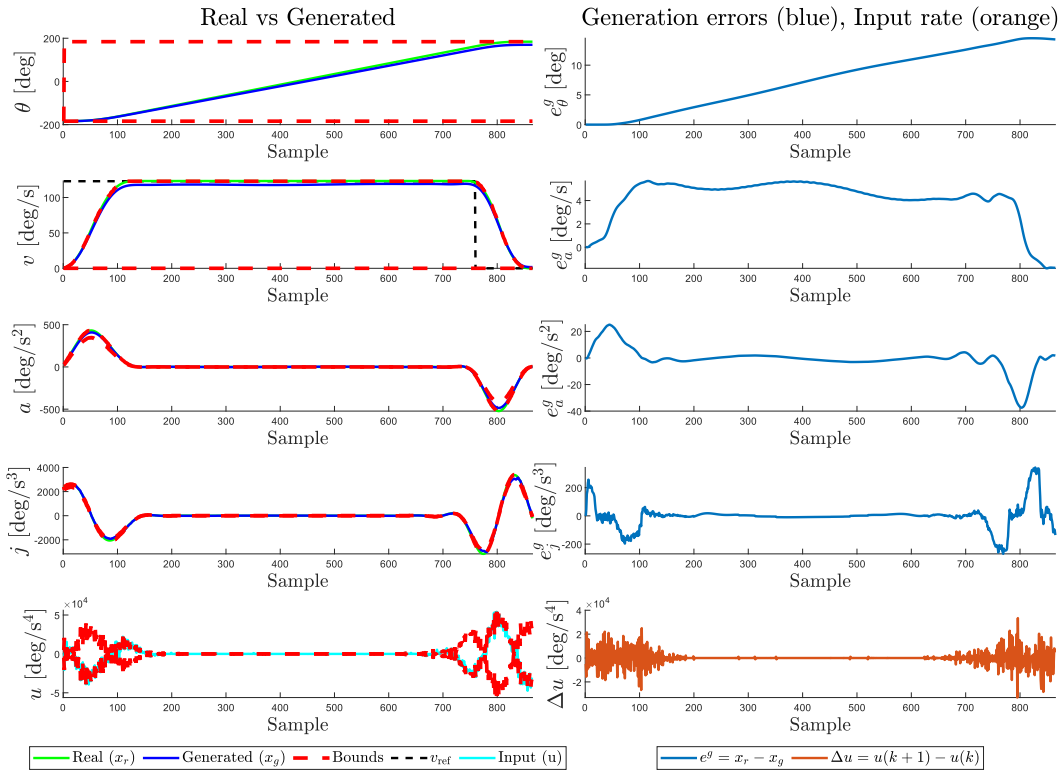

Figure 5-2: Robot joint: A1, Robot configuration: transp, Payload: 76kg - $\alpha = 0$ and $\gamma = 1.5$ Position generation with artificial state update

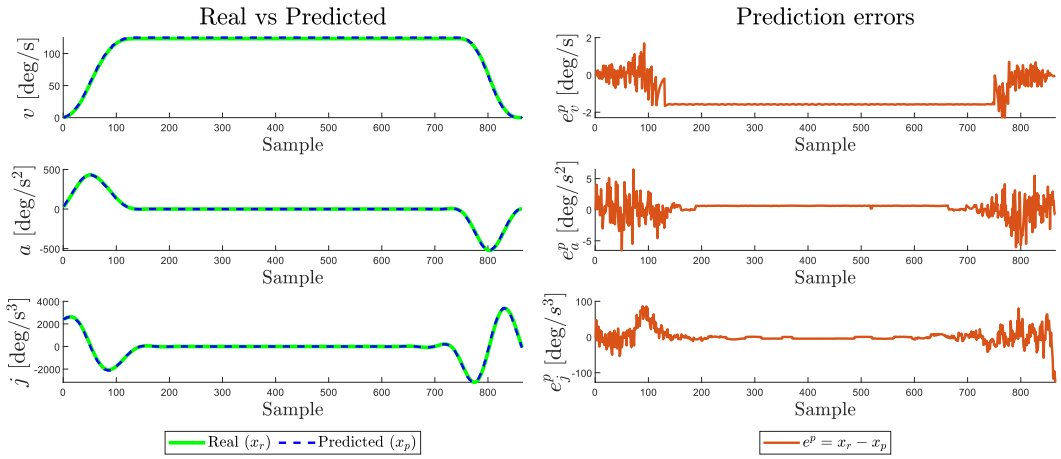

Figure 5-3: Robot joint: A1, Robot configuration: transp, Payload: 76kg - $\alpha = 0$ and $\gamma = 1.5$ Bounds prediction, using measured position and its time derivatives as input

Figure 5-2 and Figure 5-3 belong to the same experiment and aim to distinguish between the prediction and generation errors. The prediction results are more visible in Figure 5-3, where one can observe the low prediction errors obtained. Moreover, Figure 5-2 shows the generation results and the generation errors that prove satisfactory performance with a position offset at the end of the simulation of 15 degrees. A positional offset at the end of the simulation was expected since the generated velocity, acceleration and jerk profiles are more conservative than the real ones. This aspect is highlighted by the generation errors plots. When the generated profiles are more conservative and the simulation duration is constant, this will result in a positional offset at the end of the simulation.

In order to prove the effects of the parameters involved in the prediction cost function, α and γ, two comparative plots are presented in Figure 5-4 and Figure 5-5, for each of the two joints. The plots demonstrate the conservativeness of the predictions by setting $\alpha \neq 0$. In this sense, using three different parameter pairs, three sets of bounds were predicted. Using these as bounds in the trajectory optimization problem, three sets of profiles were obtained for each motion. The simulation using $\alpha = 0$ and $\gamma = 1.5$ does not enforce conservativeness and aims to predict the bounds similar to the real ones. On the other hand, the last two sets of parameters will determine conservative predictions. When comparing the simulation using $\alpha = 0.1$ and $\gamma = 0.4$ and the one using $\alpha = 0.2$ and $\gamma = 1.5$, the latter will prove more conservative. This is the result of the α parameter that will give a higher weight to the exponential function in the conservativeness cost function.

Figure 5-4 shows that, when keeping the number of samples constant in the simulation, a more conservative prediction will increase the positional error at the end of the simulation. Also, since the built-in trajectory generator aims to satisfy the bounds provided, the velocity, acceleration and jerk errors will also increase with the conservativeness. Similarly, Figure 5-5 displays the generation of a trajectory with less typical velocity, acceleration and jerk profiles, actuating joint A2.

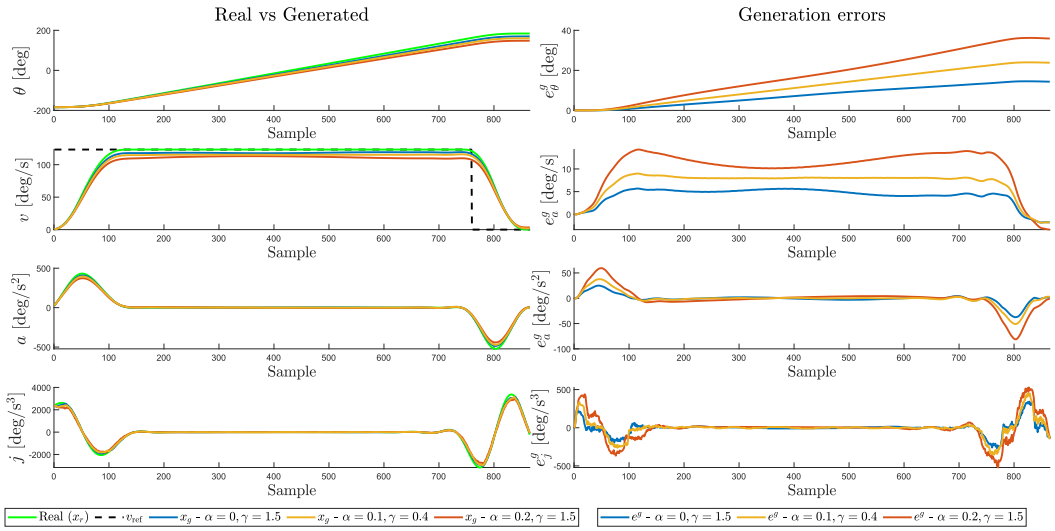

Figure 5-4: Robot joint: A1, Robot configuration: transp, Payload: 76kg. Trajectory generation with artificial state update - comparison of multiple prediction cost function parameters (α, γ)

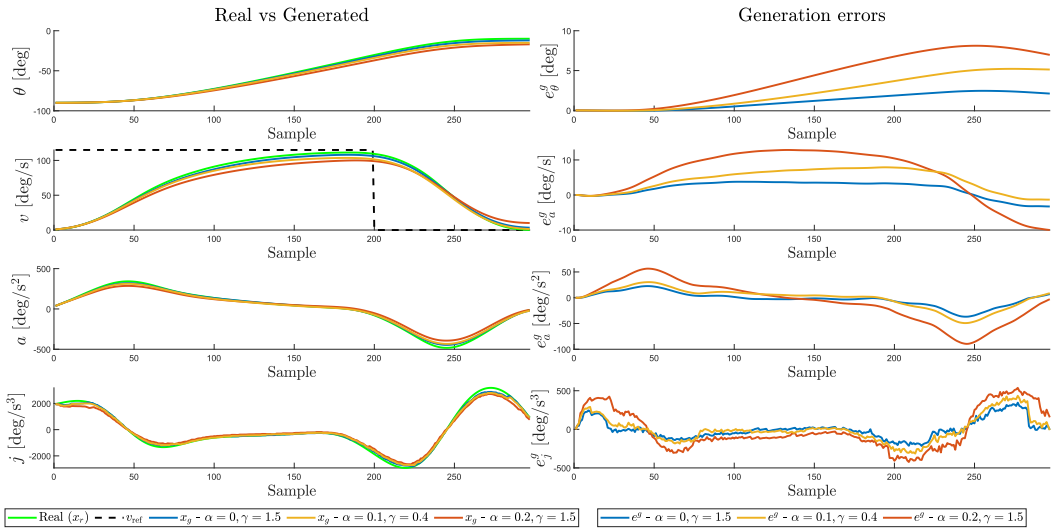

Figure 5-5: Robot joint: A2, Robot configuration: conf1, Payload: 0kg. Trajectory generation with artificial state update - comparison of multiple prediction cost function parameters (α, γ)

Furthermore, Figure 5-6 proves one of the advantages of the conservative bounds. Here, one can observe that the simulation aiming for accurate predictions (defined by $\alpha = 0$ and $\gamma = 1.5$) fails to retrieve a solution, by reaching problem infeasibilty. This is marked by the blue lines that stop around the 200th sample. On the other hand, when choosing a conservative scenario for the bound prediction, the problem becomes feasible. More specifically, one of the conservative approaches (having $\alpha = 0.1$ and $\gamma = 0.4$) yields satisfactory results and proves the benefits of the conservative predictions that favor dynamic feasibility. This favorable case is marked by the yellow lines and yields a position offset of about 20 degrees.

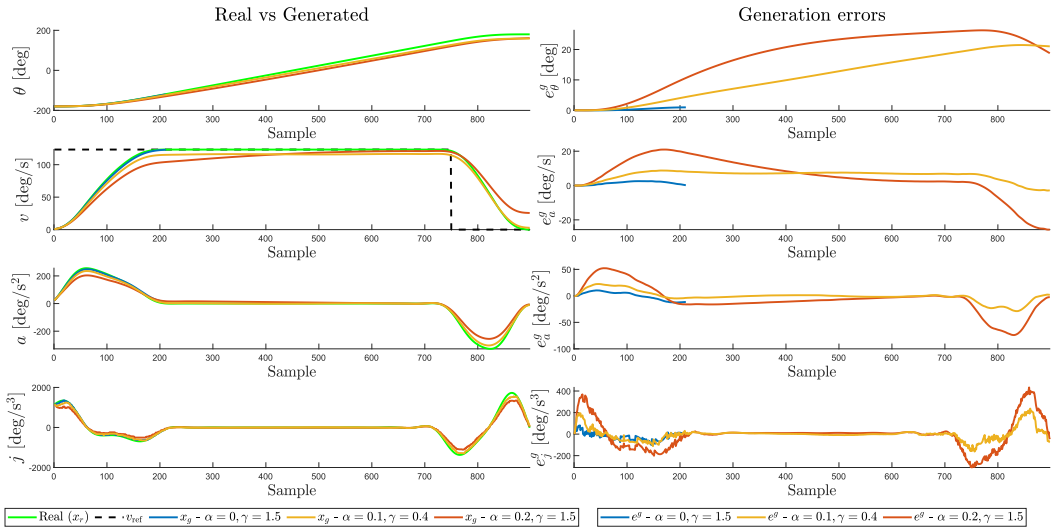

Figure 5-6: Robot joint: A1, Robot configuration: conf1, Payload: 120kg. Trajectory generation with predicted bounds - comparison between multiple prediction cost function parameters (α, γ)

On the other hand, in Figure 5-6 a more conservative scenario (e.g. the one marked by red lines) solves the infeasbility problem but leads to higher velocity offset at the end of the simulation (which also means that the final velocity is far from 0) and higher position offset during the simulation. This strengthens the idea that the cost function parameters enforcing conservative predictions need to be tuned for yielding the best results, depending on each motion's characteristics.

5-2 Real state update

For the final solution to be complete, the proposed approach should be able to generate minimum-time dynamically feasible trajectories between the given start and target positions with knowledge of only robot parameters provided by the user.

In this sense, the bounds prediction and the trajectory generation should work in a recursive fashion and the output of the trajectory generation block will be the input of the bounds prediction block and vice-versa. More specifically, the bounds predictor will consider as the current robot state the latest point computed by the trajectory generator. Further, the trajectory generation block will use the latest bounds predicted to generate a new position point in the trajectory. A graphical representation of this approach is presented in Figure 5-7.

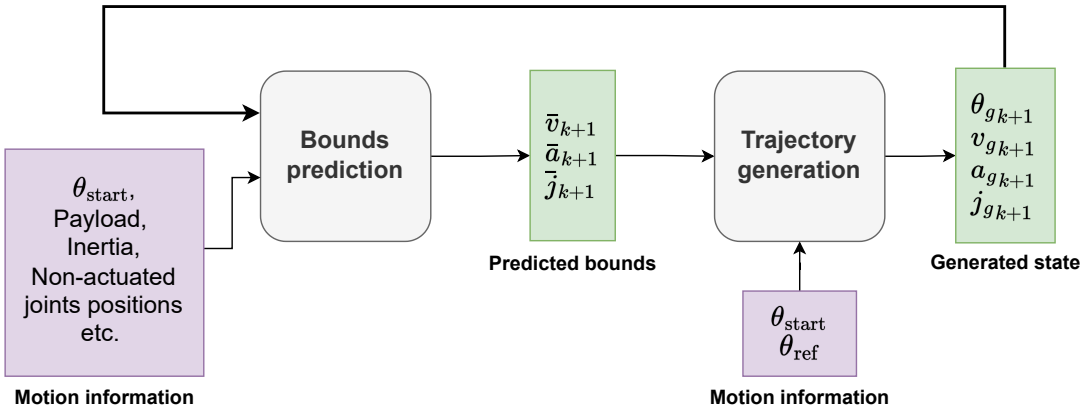

Figure 5-7: Diagram of Trajectory Generation with predicted bounds using real state update

Simulation results

Several simulation trials are introduced further. Unfortunately, none of the simulations executed using the current approach resulted in a successful experiment. When the predicted bounds deviate too much from the original ones, the trajectory generation classifies the problem as unfeasible. An example of this situation is represented by the jerk predictions around the 70th sample of the simulation presented in Figure 5-8. In this case, the jerk predictions do not resemble the original jerk values and, since the problem fails from retrieving further solutions, it will cause the simulation to stop.

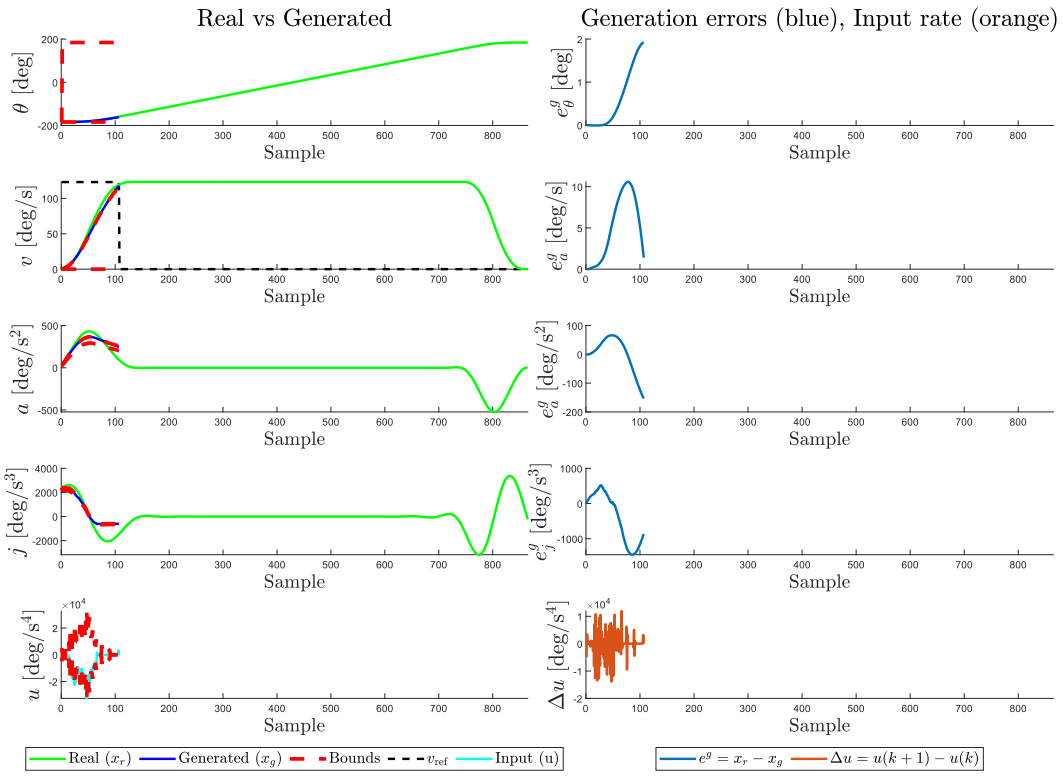

Figure 5-8: Robot joint: A1, Robot configuration: transp, Payload: 76kg - $\alpha = 0$ and $\gamma = 1.5$. Trajectory Generation with predicted bounds using real state update

One motivation behind this might be the lack of feedback either from the robot system or the original trajectory. In the absence of feedback, the prediction and trajectory generation errors will accumulate leading to a deterioration in the prediction quality.

Another possible reason for the unsatisfactory outcome of the final solution might be represented by the rather reduced dataset used for regressors' training. This might limit the generalization capabilities of the prediction algorithm since the robot behavior might not be entirely represented by the reduced dataset.

One last aspect that might lead to an unsuccessful outcome is the reduced ability of the prediction algorithm to distinguish well-enough between the steady regions and the deceleration regions. This might be improved by an extensive dataset as well.

To check whether more conservative predictions of the velocity, acceleration or jerk bounds will improve the outcome, two comparative plots are provided further. For both plots, three sets of parameters defining the prediction cost function were applied. In Figure 5-9 one can observe that a more conservative prediction did not improve the outcome. On the contrary, the higher errors presented by the red line show that the most conservative prediction presented actually lead to increased deviations from the expected behavior.

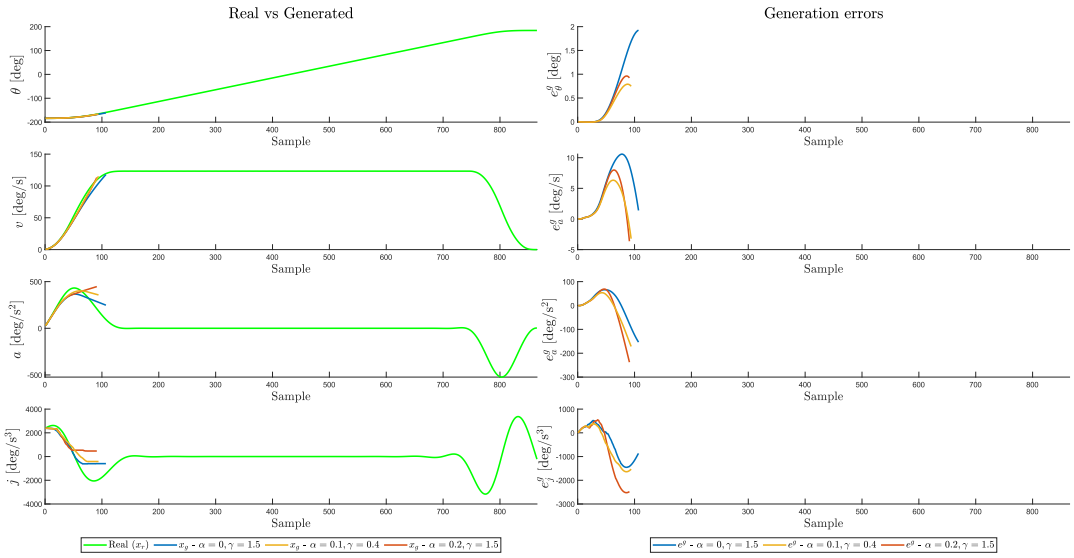

Figure 5-9: Robot joint: A1, Robot configuration: transp, Payload: 76kg. Trajectory Generation with predicted bounds using real state update - comparison between multiple prediction cost function parameters (α, γ)

On the other hand, Figure 5-10 shows that one of the conservative set of profiles, marked by the yellow lines, did not result in an unfeasible problem. Although the simulation was considered successful, the velocity, acceleration and jerk profiles significantly deviate from the real profiles. This will still lead to reaching the target position, but the obtained trajectory does not resemble the motion regions observed in the recorded profiles (acceleration, constant velocity and deceleration).

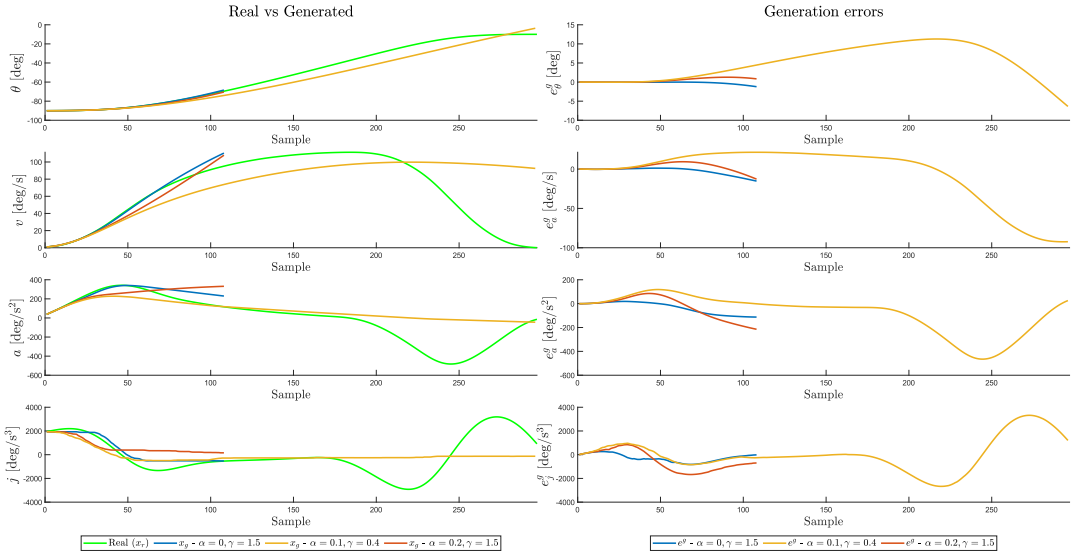

Figure 5-10: Robot joint: A2, Robot configuration: conf1, Payload: 0kg. Trajectory Generation with predicted bounds using real state update - comparison between multiple prediction cost function parameters (α, γ)

When conservative bounds are involved in the trajectory generation, one more explanation could be found for the unsatisfactory behavior. In the development phase, the regressors are trained on the real robot profiles, that are not lowered by any conservative approach. On the other hand, when conservative bounds are used for trajectory generation, the entire profile will be reduced and the prediction block might receive as input different data from its training set. In this sense, an improved training method or an increased dataset including more reduced profiles might enhance the performance of the solution.

5-3 Trajectory generation with bounds prediction validated on the robot

To validate the trajectory generation including bounds prediction, several robot tests were conducted. Since the real state update approach did not yield successful simulations, only the artificial state update method was tested on the real robot and the results are presented in the figures below. These were considered useful to conclude the feasibility of the solution aggregating the bounds prediction and trajectory generation.

The execution of the KUKA experiments in these scenarios was motivated by the following questions:

- Will the prediction integrated in the trajectory generation cause a controlled stop?
- Will the bouncing input rate affect the trajectory execution on the robot?
- Will the conservative prediction introduce any drawback in the robot's experiments?

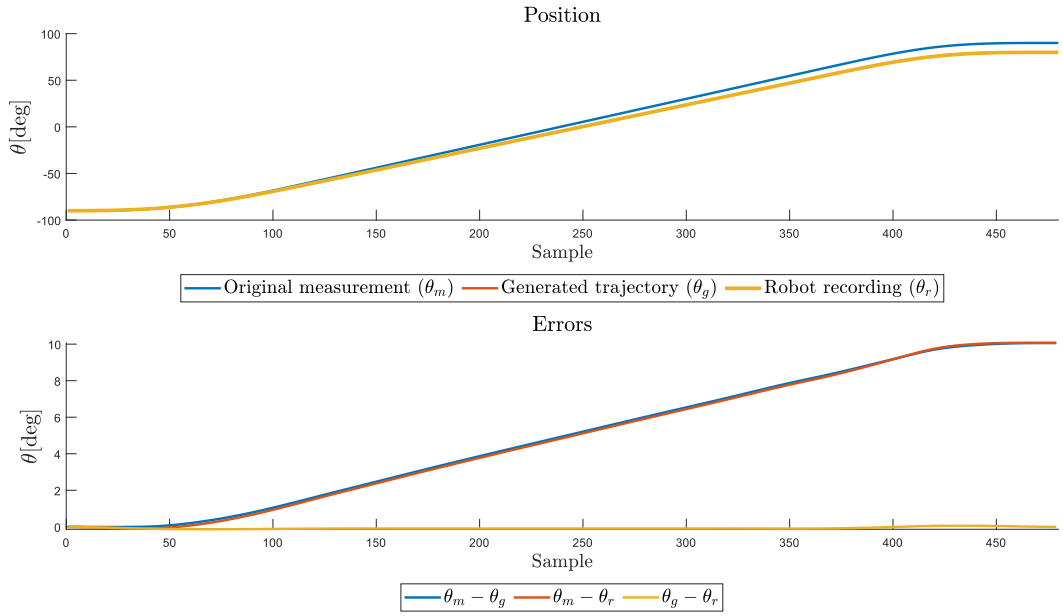

Figure 5-11: Robot joint: A1, Robot configuration: shorter, Payload: 76kg.
Recording of generated trajectory with artificial state update executed on the KUKA robot

The experiments on the KUKA robots proved that the prediction integrated in the trajectory generation will not lead to a controlled stop. Precisely, the robot could closely follow the generated trajectory. This robot behavior also indicate no problem induced by the bouncing input rate (such as the one presented in Figure 5-8).

Figure 5-11 and Figure 5-12 are similar to the ones provided in Chapter 4 for testing the trajectory generation approach alone. In this sense, the two figures show the real position recorded from the KUKA robot using the robot controller, the generated trajectory using the artificial state update approach and the robot position collected with the generated trajectory used as input.

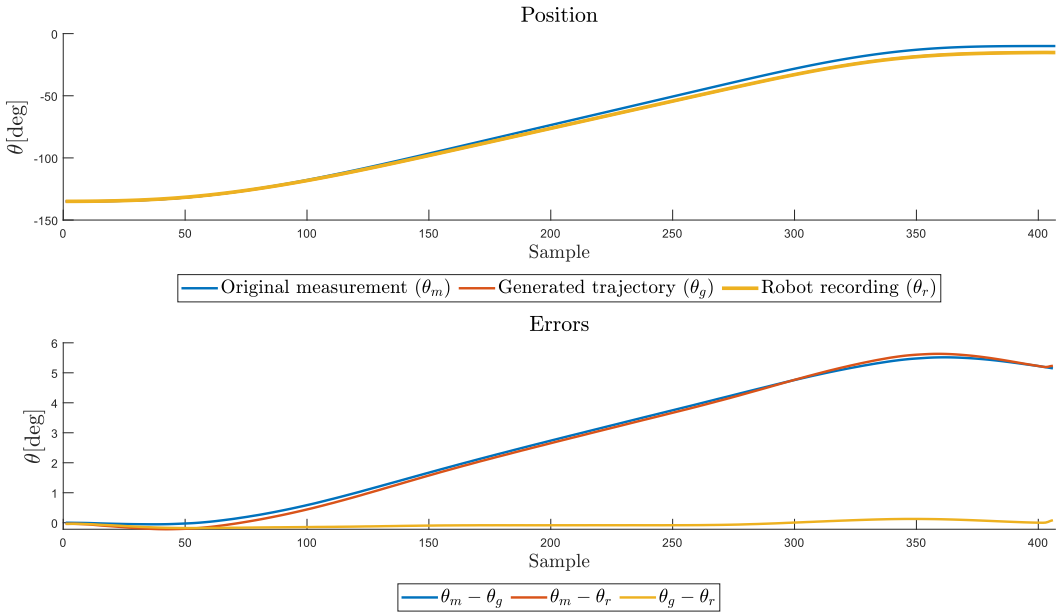

Figure 5-12: Robot joint: A2, Robot configuration: conf1, Payload: 76kg.
Recording of generated trajectory with artificial state update executed on the KUKA robot

In both plots one can observe a rather increased generation error (marked by the blue line in the lower plot), but a reduced error between the input and the output of the robot experiment (marked by the yellow line in the lower plot). As expected, this behavior led to the blue and red lines in the lower plot to coincide since the recorded trajectory followed closely the generated trajectory.

5-4 Trajectory generation with conservative bounds validated on the robot

In order to test the effect of the conservative bounds, trajectories generated using the artificial state update approach were provided to the KUKA robot. In this sense, Figure 5-14 and Figure 5-16 show the robot recordings of the trajectories generated with three different sets of parameters for the prediction objective function. The trajectory generation is presented in Figure 5-13 and respectively, Figure 5-15.

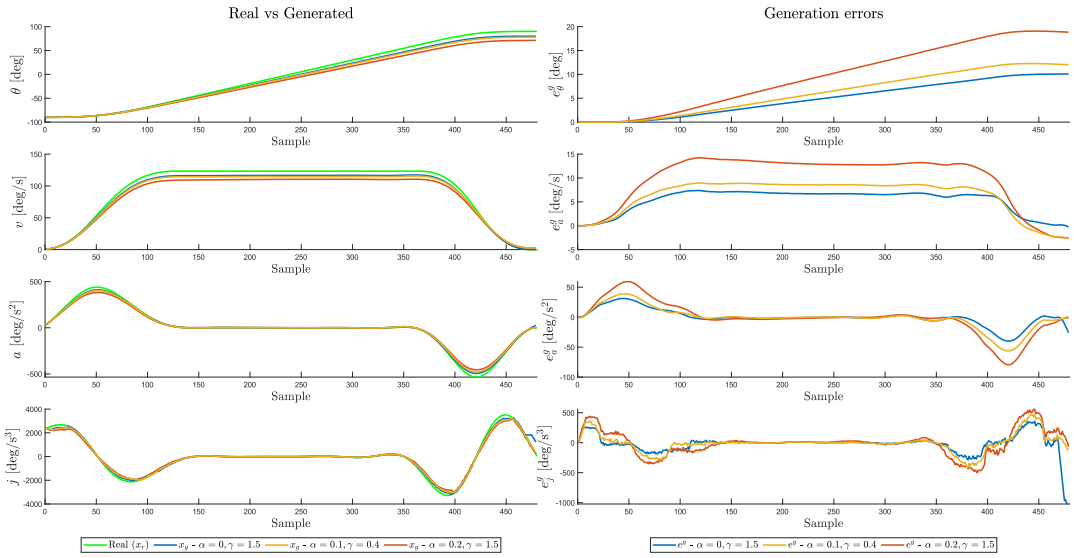

Figure 5-13: Robot joint: A1, Robot configuration: shorter, Payload: 76kg.
Trajectory generation with artificial state update - comparison with multiple prediction cost function parameters (α, γ)

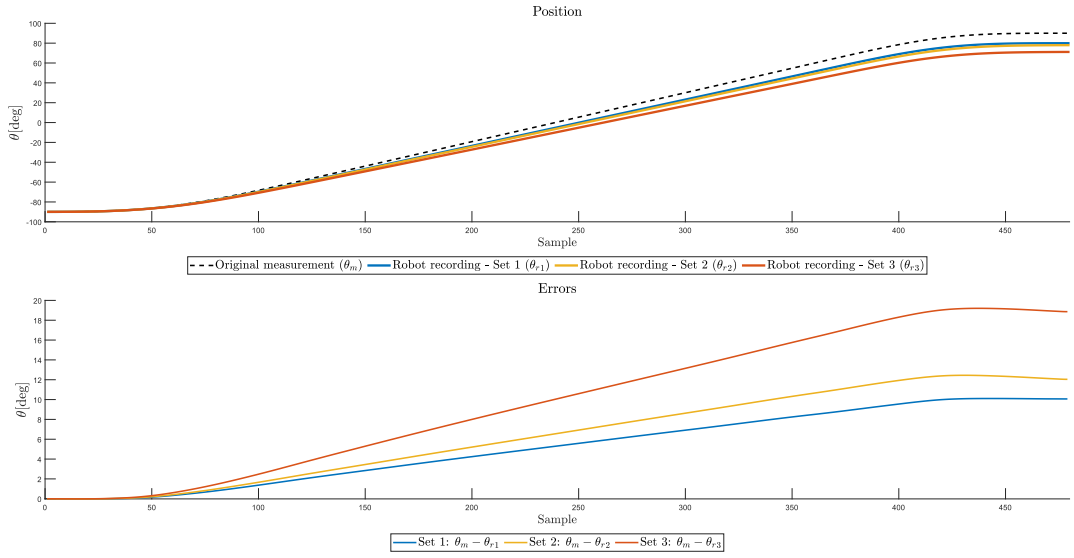

Figure 5-14: Robot joint: A1, Robot configuration: shorter, Payload: 76kg.
Recording of generated trajectory with artificial state update and conservative bounds executed on the KUKA robot
Set 1: $\alpha = 0, \gamma = 1.5$, Set 2: $\alpha = 0.1, \gamma = 0.4$, Set 3: $\alpha = 0.2, \gamma = 1.5$

As the results in Subsection 5-1 proved, in the artificial state update approach which uses original robot measurements, the number of samples will be preserved. Moreover, this also implies that the conservative bounds still follow the original profiles form but lower the predicted values. As a result of the conservativeness, when predicting lower bounds, the position offset at the end of the simulation is expected to increase. In an ideal situation, the opti-

mization problem should run until the target position is achieved, allowing the trajectory generation to follow slightly different velocity, acceleration and jerk profiles. This might be obtained in an improved version of the real stated update approach.

On a good note, Figure 5-14 and Figure 5-16 show that the robot was indeed able to follow the generated trajectory independent of the parameters used in the objective function for bounds prediction. As expected, the position offset at the end of the simulation increased. Additionally, the bottom plot of both figures shows the error between the original recording and the robot recordings following the generated trajectories. In Figure 5-14, the position error accumulates throughout the experiment samples and stabilizes at the end of motion. This led to increased position offset at the end of the simulation when the conservativeness is more pronounced.

In comparison to Figure 5-14 in which the positional offset accumulates throughout the experiment samples and stabilizes at the end of motion, in Figure 5-16 the positional offset at the end of the two conservative experiments (marked by red and yellow lines) seem to start decreasing. This behavior might be caused by a rather improper trajectory generation that presents an increased velocity at the end of the simulation. This behavior can be observed in Figure 5-15 after the 350th sample. As a result, the robot will get closer to the target position but the velocity profile will slightly differ in shape from the original one.

It is also interesting to mention that during the actual experiments, the robot was moving significantly slower. This was observed not only visually, but also by the sound the robot made when executing the trajectory. The sound also suggested that the motions executed are rather smooth.

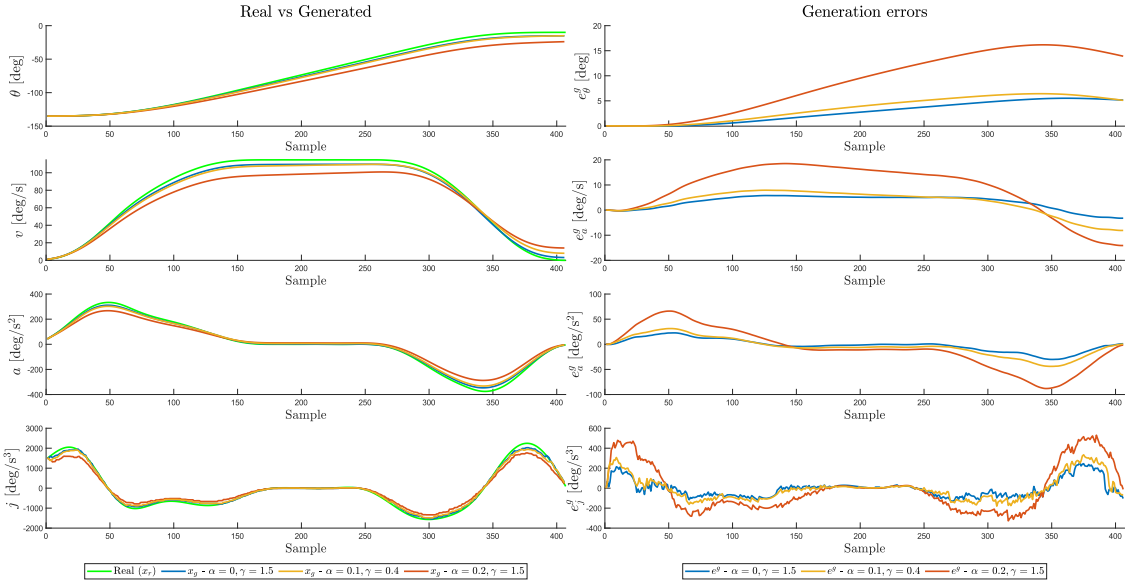

Figure 5-15: Robot joint: A2, Robot configuration: conf1, Payload: 76kg.
Trajectory generation with artificial state update - comparison with multiple prediction cost function parameters (α, γ)

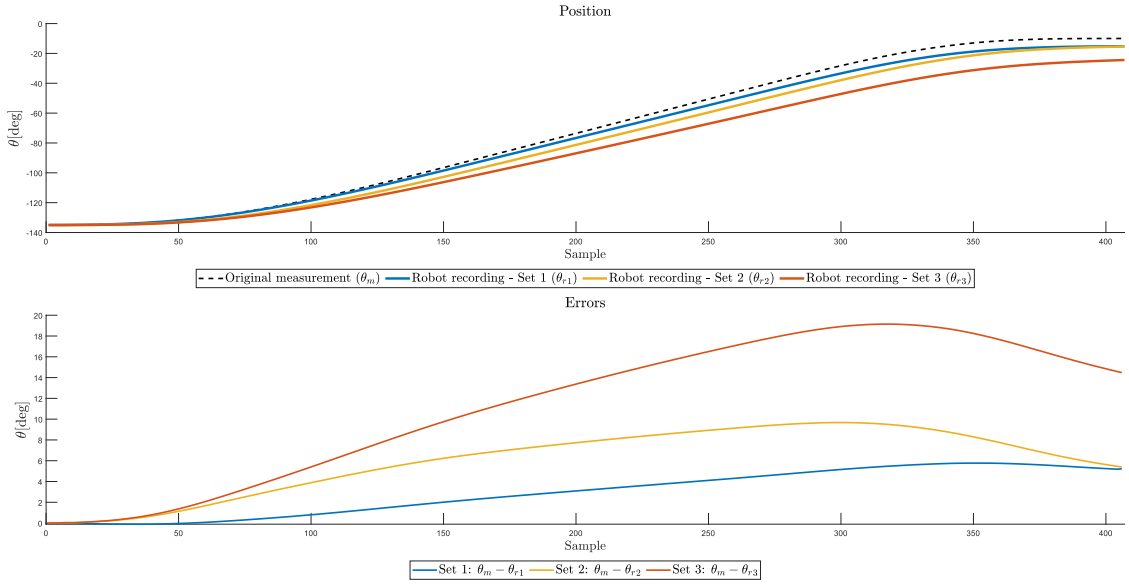

Figure 5-16: Robot joint: A2, Robot configuration: conf1, Payload: 76kg.
Recording of generated trajectory with artificial state update and conservative bounds executed on the KUKA robot.
Set 1: $\alpha = 0, \gamma = 1.5$, Set 2: $\alpha = 0.1, \gamma = 0.4$, Set 3: $\alpha = 0.2, \gamma = 1.5$

5-5 Conclusion

The current chapter introduces the complete approach combining bounds prediction and trajectory generation. To isolate the possible errors sources, two approaches were proposed using either an artificial or a real state update. In this sense, the main distinction arises from the update of the current robot state used as input for the bounds prediction block.

In the artificial state update approach, the current robot state is obtained from the robot position measurements collected during KUKA experiments. This method achieved successful simulation results and proved the capability of the trajectory generation with bounds prediction. Furthermore, simulations including conservative bounds were included. The conservativeness proved effective to transform an unfeasible simulation into a feasible one.

Further, the end-to-end solution was presented including a recursive approach in which the current state, used as input of the bounds prediction block, is represented by the next state computed by the trajectory generator. The results unfortunately showed no successful simulation and several possible causes were presented. One potential reason is the lack of feedback which led to accumulated errors affecting the predicted bounds. In terms of bounds prediction, another possible motivation for the unsuccessful solution is the rather reduced dataset used for regressors' training. The quality of the dataset might also affect the capability of the regressors to distinguish between the motions regions.

Finally, robot experiments results were provided for the artificial state update method to prove the approach's effectiveness on the KUKA robot. The generated trajectories were closely followed by the robot and no controlled stop was triggered during the experiments.

Chapter 6

Conclusion

6-1 Summary

This work is focused on generating dynamically feasible trajectories using a data-driven approach in order to achieve high performance control of robot manipulators. Since robot systems are typically concealed, with no information provided about their operation, this thesis aimed to discover a part of the robot controller and generate more transparent trajectories that include the robot's limitations.

First, the robot setup was described as the basis of the testing environment. This is composed of a KUKA KR 210 R2700 robot mounted on a linear unit, the required robot controller KUKA KR C4, the teach pendant and connecting cables. Further, to set the context for the experiments using the KUKA robot, the procedure for data collection was introduced. This requires the usage of the Teach Pendant to specify robot parameters such as tool information, a KRL script to specify the motion to be executed and KUKA RSI to transmit the position information to an external computer with a sampling time of 4ms.

Since KUKA RSI only permits position data to be recorded, three time-derivation methods were introduced and compared. Out of the methods presented, Forward Finite Differences proved the most promising results and was used further to determine the velocity, acceleration and jerk profiles. FFD provided smooth approximations of the position's time-derivatives that could be used in the trajectory generator. When compared to the Kalman filter, FFD presented no lag in between the profiles, which facilitated the trajectory generation. Moreover, since jerk profiles computed using the FFD benefit of a higher amplitude in the first samples, this will favor a reduced execution time by reaching the maximum velocity faster.

After establishing the procedure to retrieve the velocity, acceleration and jerk profiles from position measurements, an initial analysis on several robot motions was performed to investigate valuable features in predicting dynamic limitations. The evaluation on multiple motion parameters confirmed the existence of the Limiter block and and its effect. As far as the collected evidence suggests, the payload's weight and the robot pose show the biggest influence on the velocity, acceleration and jerk profiles. As a result, by varying these two parameters,

an extended dataset was created to describe the limits imposed by the robot controller as a function of the robot parameters. This initial analysis and the resulting dataset is particularly important for determining the behavior of the Limiter block.

Since the current work aims not only to identify the Limiter block, but also to provide the robot with dynamically feasible trajectories, a method to supply position commands to the robot was presented. This involves a Python script and KUKA RSI to define the position samples and communicate through UDP with a sampling time of 4ms. During experiments, this method proved rather nondeterministic since multiple identical input trajectories led to different robot behavior. The worst case encountered during experiments was when providing the same input trajectory to the robot, nine out of ten experiments caused a controlled stop in different regions of the robot motion and only one finished the motion execution successfully.

Considering that the manufacturer does not provide any information on how the robot controller produces the limits imposed on velocity, acceleration and jerk, the profiles enforced during motions running with maximum speed level were considered to demonstrate the Limiter's effect. This assumption is motivated by manufacturer's guarantee that maximum speed level motions will reach the target position in minimum-time, suggesting maximum robot performances that do not violate any of the imposed bounds. Further, this information was used to determine the function defining the Limiter block, using multiple motion relevant parameters, in order to predict the velocity, acceleration and jerk bounds for a specific motion.

The bounds prediction method is iterative and aims to determine the next state's bounds from the current robot state. In simple words, the bounds prediction aims to determine which is the most distant point that the robot can achieve in a sample of 4 ms starting from the current position, without causing a controlled stop.

Multiple methods were investigated for predicting the bounds such as Linear Regression, for its simplicity, Random Forest, to increase the model complexity and therefore prediction accuracy and Extreme Gradient Boosting, for its complexity and the capability to use a custom objective function. The objective function was changed in the XGBoost formulation to favor more conservative predictions, ensuring that the predicted bounds never exceed the real bounds. As a result, although both Linear Regression and RF provided promising results, XGBoost was used in the final solution. Its accuracy, together with the custom objective function provided adequate predictions to the current problem.

Since the bounds are determined independently for velocity, acceleration and jerk, a method for trajectory generation that obtains attainable position values was required. First, an open-source approach was tested. However, because it was determined to be unsuitable for the current problem, a custom trajectory generation method was proposed.

Knowing the current robot position, the trajectory generator computes, in an iterative fashion, the most distant viable position sample that can be reached in 4 ms without violating any of the bounds imposed by the Limiter block. This is obtained by solving an optimization problem in which the velocity, acceleration and jerk bounds are used as state constraints. Input constraints are also included in the form of snap constraints and are computed using the predicted jerk bound.

Although multiple successful simulations resulted from the trajectory generator, some experiments still fail to obtain a feasible solution. The main reasons behind the unfeasible problem result are the non-negativity and the maximum admissible velocity constraints. On the other

hand, an analysis showing both the feasibility of the experiments and the average positional offset at the end of the simulation (computed only for the successful ones) was provided. The capabilities of the trajectory generation method were also validated in robot experiments, the generated trajectory being followed with negligible error.

Finally, to include the bounds prediction in the trajectory generation solution, a definitive solution was proposed. In the final approach, the bounds prediction and trajectory generation work in a recursive approach using real state update. This method provides the current robot state, required for bounds prediction, from the latest generated state. To isolate the possible error sources, an artificial state update was proposed. In comparison to the real state update, the main distinction of this approach is that the current robot state is obtained from position measurements.

Using predicted bounds for the trajectory generation in the artificial state update manner, the solution yielded satisfactory results both in simulation and robot experiments. This method proved that the prediction error does not affect the robot's behavior and did not trigger a controlled stop. Moreover, several simulations proved that the conservative bounds prediction could solve the unfeasibility problem. Furthermore, the final solution represented by the real state update method did not yield any successful simulation so no robot experiments could be executed.

6-2 Closing remarks

This section will provide details on the important findings in this work.

The initial analysis empirically proved that the payload weight and the robot pose considerably influence the velocity, acceleration and jerk bounds imposed by the KUKA robot controller. Although this analysis was limited by the tool used in the robot experiments, the variation of several parameters was sufficient to prove the existence of the Limiter block and to explain its effects by the variation of motion or tool parameters. This serves as valuable groundwork for future experiments to include other relevant motion parameters.

Regarding the trajectory generation, although intermediary solutions obtained successful simulations, the final solution still presents two bottlenecks. First, the trajectory generation approach is not robust on its own and fails to retrieve a solution in several cases. Secondly, the trajectory generation using predicted bounds in the real state update fashion fails to generated attainable robot trajectories. These two bottlenecks will be further detailed.

When using velocity, acceleration and jerk bounds computed by time-derivation from previously collected position measurements, the trajectory generation obtained a positional offset at the end of the simulation relative to the motion distance smaller than 3.2% for both analyzed joints. This metric shows a limited performance deterioration caused by the trajectory generation. In real scenarios, the offset at the end of the simulation could be acknowledged by the user who should provide a higher target position. Since this is not possible for target positions close to the end of a joint motion range, further improvements on the solution performance might be desired. Altogether, the robot experiments showed that when the provided bounds are similar to the real bounds imposed by the Limiter block, the trajectory generator will compute feasible trajectories that can be exploited in real robot scenarios.

On the other hand, the trajectory generation method still fails to retrieve a feasible solution in several scenarios. The main cause for this is the violation of maximum or minimum admissible velocity, which led to an unfeasible optimization problem. This demonstrated the need for an improved robustness of the trajectory generator method. During experiments several aspects affecting the robustness were identified and suggestions on how to improve will be further indicated.

The velocity bounds violation could be caused by the short horizon considered in the optimization problem. This causes shortsightedness in the optimization approach by not considering further decisions. To solve the shortsightedness problem, a longer horizon approach was proposed. Unfortunately, this could not resolve the problem because, considering only the knowledge of next step bounds, constant limits for velocity, acceleration and jerk were imposed for the next N samples. This only limited the performance of the trajectory generator by not accounting for varying velocity, acceleration or jerk bounds. This suggests the requirement for a method to recursively predict future N bounds to combat this limitation. This will likely increase the robustness and the performance in the acceleration and deceleration regions of the trajectory.

In comparison to the trajectory generation method using the originally computed bounds (using previously recorded position measurements), the artificial state update, with predicted bounds, obtained insignificant growth in the positional offset at the end of the simulation relative to the motion distance. This result showed that a more accurate prediction will not greatly impact the trajectory generation. Moreover, the robot experiments showed that a less smooth input, caused by the prediction error, would not lead to a controlled stop. This already proves that in a real scenario, with sufficiently accurate predictions, the generated trajectory could be applied to robots. Furthermore, as far as the collected evidence suggests, the higher order derivatives (relative to acceleration) will not play a major role in causing a controlled stop.

Additionally, simulations showed that several scenarios originally classified as infeasible could be solved by the usage of the conservative bounds predictions. This proves that, at the cost of increased execution time or positional offset at the end of the simulation, transparent trajectories can still be obtained. This observation matches the initial intuition that less violent trajectories, in terms of acceleration and jerk profiles, are less likely to cause a controlled stop.

The real state update approach did not yield any successful simulations. This is caused by three potential reasons. First, the relatively reduced dataset that does not represent a sufficiently broad region of the robot's behavior, might affect the prediction capabilities. More specifically, due to consecutive prediction and generation errors, the generated trajectory samples might reach uncovered regions of the robot's behavior leading to erroneous predictions.

Second, the model seems to not distinguish well enough between profiles' regions, e.g., between acceleration and constant velocity regions. Similarly, accumulated errors cause the generated trajectory samples to be part of a constant acceleration region sooner than the experiment would indicate in reality. This will lead to constant predicted bounds since the regressor expects no change in the current state, leading to increasing errors between the predicted and real profiles. Empirically, this led to an unfeasible simulation.

Third, the high discrepancy between the predictions and the real profile values might also be the result of accumulated errors due to the lack of feedback, such as from an ideal path. A

correction should be made in this sense because long trajectories will always be prone to error drift. As manufacturing processes typically require longer trajectories for efficiency purposes, this would represent a major cause of concern and should be tackled further.

Moreover, it was also proved in simulation that in the real state update scenario, the conservative bounds prediction did not significantly improve the simulation results. The main reason behind this is that having conservativeness involved, the bounds prediction block will receive the adjusted current state values (lowered as a result of more conservative bounds) which might not be covered by the training set. This is aligned with previous observations as conservative prediction might lead to uncovered regions of the robot dynamics, requiring a more comprehensive dataset in order to be effective. This will be further detailed in the next section.

All in all, the successful simulation results showed the suitability of transparent trajectory generation that uses knowledge about the KUKA Limiter block. Although not entirely successful, the proposed solution represents a step in the direction of improved predictability of robot manipulators. Conversely, there are two fundamental drawbacks that might be valuable for further work. First, the need for an improved prediction model that can distinguish between trajectory regions and that can better represent a broader area of the dynamics space. Second, a more robust trajectory generation method that is less sensitive to velocity constraints.

Further, several prospects on solving the current solution's shortcomings will be presented.

6-3 Future work

Considering the major drawbacks of the proposed solution, various potential directions to improve the current work were identified. Further, the possible next steps are described and motivated.

- Since the current solution seems to lack knowledge about the Limiter block, a broader, more comprehensive dataset that more broadly and accurately represents the robot's behavior needs to be built. This can be easily achieved by first, including more robot poses in the datatset by varying the position of the non-actuated joints and second, by varying more tool parameters.
- Given the functional limitation of the current solution, different motion related parameters might be varied and considered in the bound prediction approach. One example of possibly relevant motion information is represented by joint torques that can be easily measured using KUKA RSI.
- Assuming the availability of a more generous dataset, a more complex prediction method can be applied to better recognize the steady regions. Eventually, a stateful method might be desired.
- Considering the lack of feedback in the current method that permits the accumulation of errors during the motion simulation, a form of feedback from a suitably-designed reference path could be introduced in the Trajectory Generation.

- As one long term goal of this work was to be easily applicable to all robots, the proposed solution could be extended to cover also other robots' manufacturers. For time related reasoning, this work had to be limited to KUKA robots.

Appendix A

Joints A3 to E1

A-1 Robot poses definition

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
A3 "transp"	0	-130	-110 to 145	-180	-120	30	-4000
A3 "shorter"	0	-90	-50 to 60	-180	-65	30	-4000
A3 "conf1"	0	-32	120 to -115	-180	-60	30	-4000
A3 "conf2"	20	-10	115 to -60	-180	110	30	-4000

Table A-1: Robot joint: A3, Position of all joints for multiple robot poses.

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
A4 "transp"	0	-90	90	-340 to 340	-90	30	-4000
A4 "shorter"	0	-90	90	-100 to 100	-90	30	-4000
A4 "conf1"	0	-25	65	-345 to 345	-65	30	-4000
A4 "conf2"	20	-50	110	-180 to 180	-55	30	-4000

Table A-2: Robot joint: A4, Position of all joints for multiple robot poses.

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
A5 "transp"	0	-90	110	0	-120 to 120	30	-4000
A5 "shorter"	0	-90	110	0	-60 to 60	30	-4000
A5 "conf1"	0	-15	25	-180	110 to -120	30	-4000
A5 "conf2"	20	-50	60	-5	-90 to 90	30	-4000

Table A-3: Robot joint: A5, Position of all joints for multiple robot poses.

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
A6 "transp"	0	-90	90	-180	-90	-340 to 340	-4000
A6 "shorter"	0	-90	90	-180	-90	-100 to 100	-4000
A6 "conf1"	0	-25	63	-180	-66	-340 to 340	-4000
A6 "conf2"	20	-20	40	-180	70	-180 to 180	-4000

Table A-4: Robot joint: A6, Position of all joints for multiple robot poses.

Robot pose	A1 position [deg]	A2 position [deg]	A3 position [deg]	A4 position [deg]	A5 position [deg]	A6 position [deg]	E1 position [mm]
E1 "transp"	0	-139	125	-180	-120	30	-4000 to 0
E1 "shorter"	0	-139	125	-180	-120	30	-4000 to -3200
E1 "conf1"	0	-45	38	-180	-42	30	-4000 to 0
E1 "conf2"	45	-50	100	-180	60	30	-4000 to -2000

Table A-5: Robot joint: E1, Position of all joints for multiple robot poses.

A-2 Payload weight and robot pose effects on limits

A-2-1 Joint A3

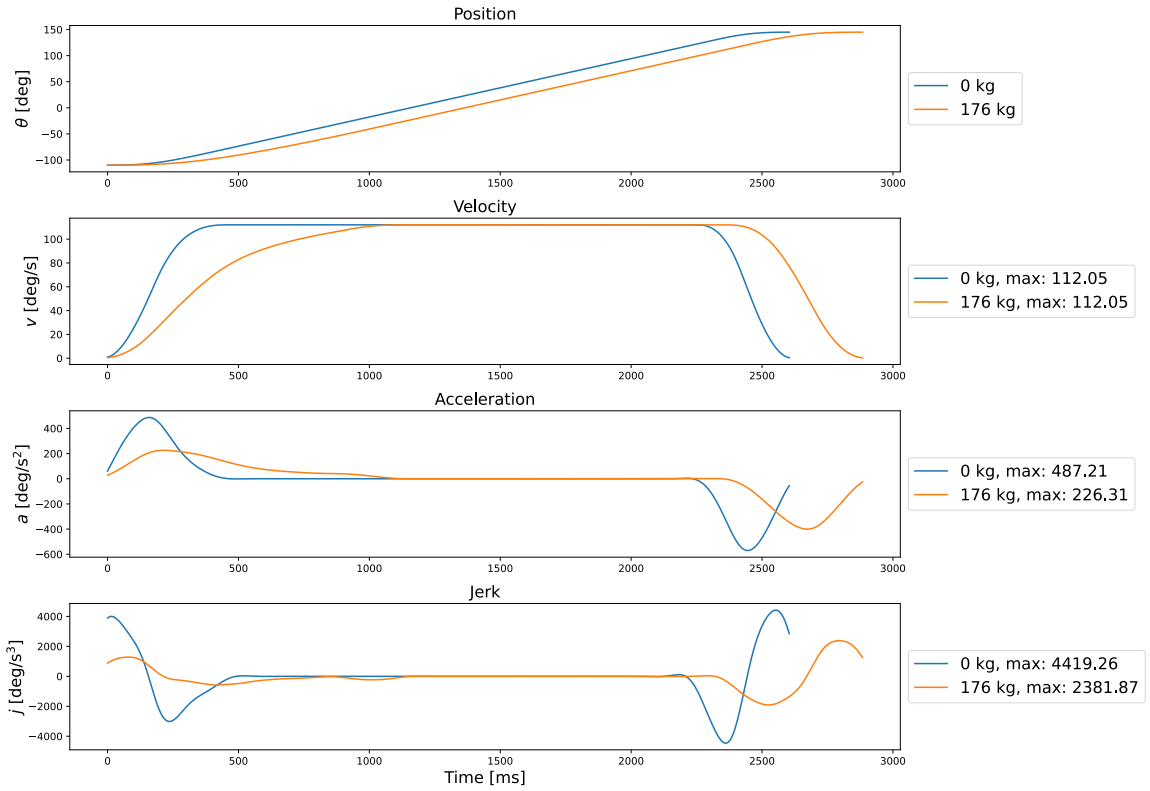

Figure A-1: Robot joint: A3, Profiles comparison between the same motion executed with the same robot pose 'transp' but different payloads.

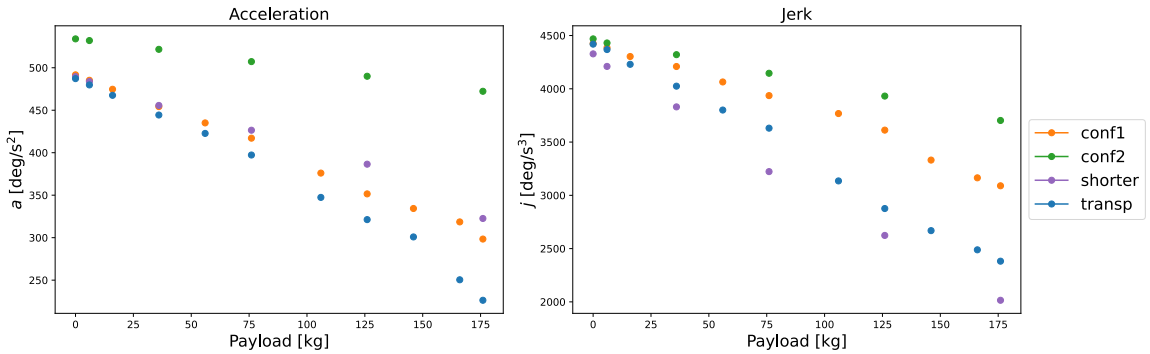

Figure A-2: Robot joint: A3, Maximum acceleration and jerk values spread depending on the payload weight and robot pose.

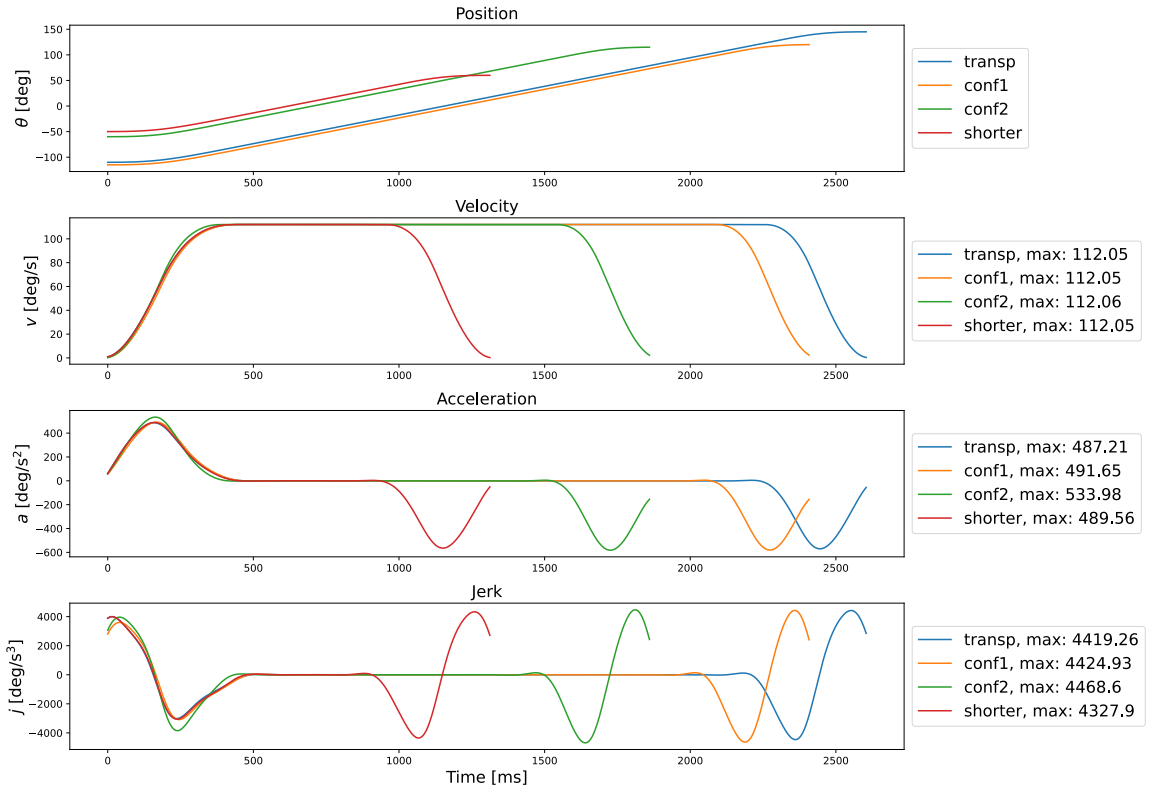

Figure A-3: Robot joint: A3, Profiles comparison between similar motions executed with no payload but different robot poses.

A-2-2 Joint A4

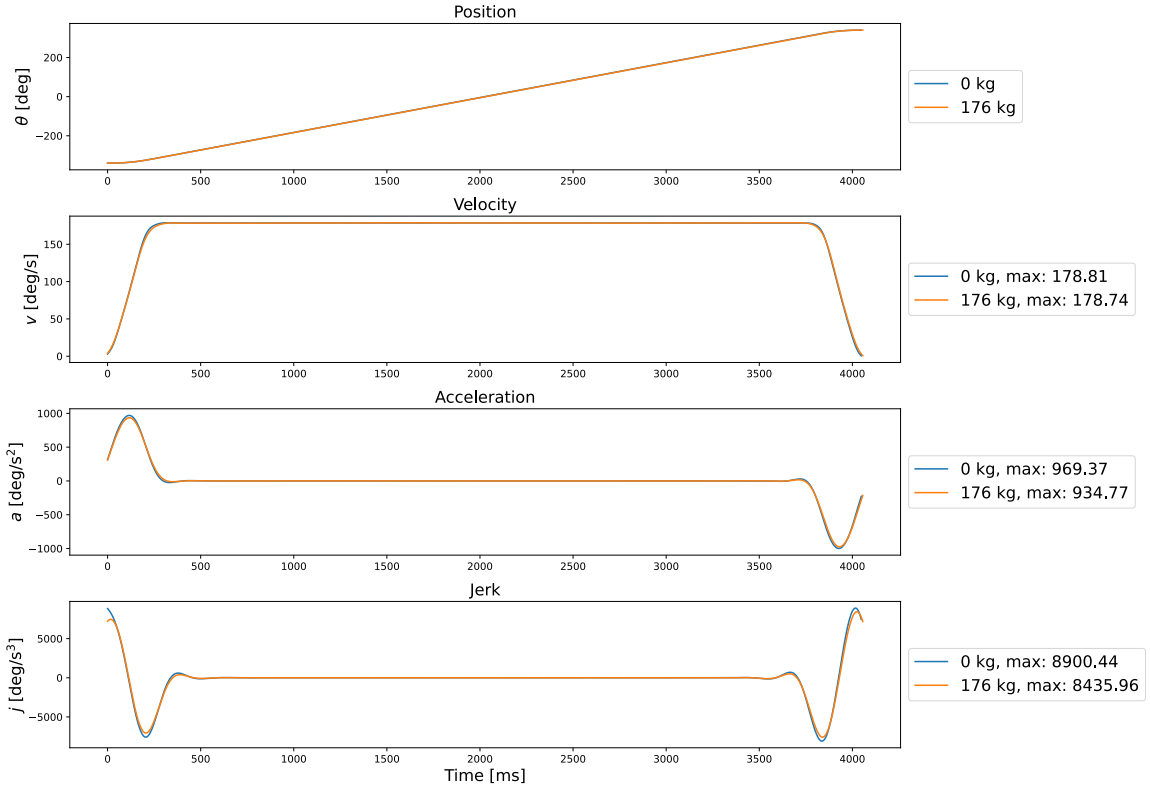

Figure A-4: Robot joint: A4, Profiles comparison between the same motion executed with the same robot pose 'transp' but different payloads.

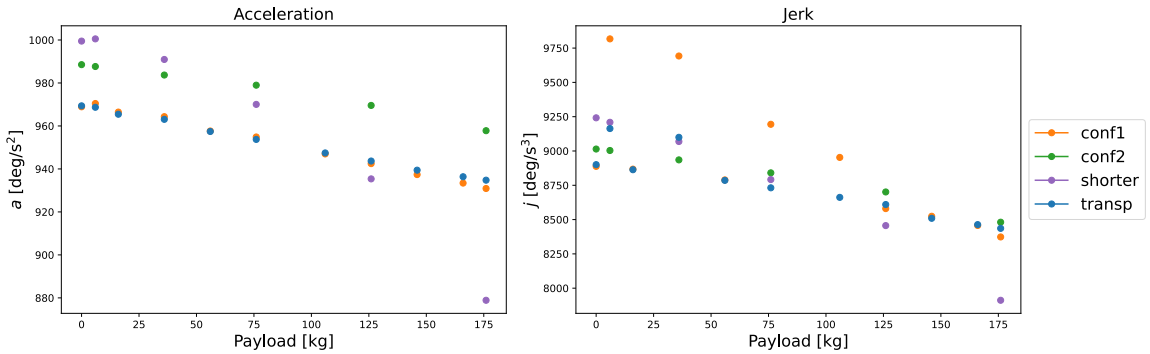

Figure A-5: Robot joint: A4, Maximum acceleration and jerk values spread depending on the payload weight and robot pose.

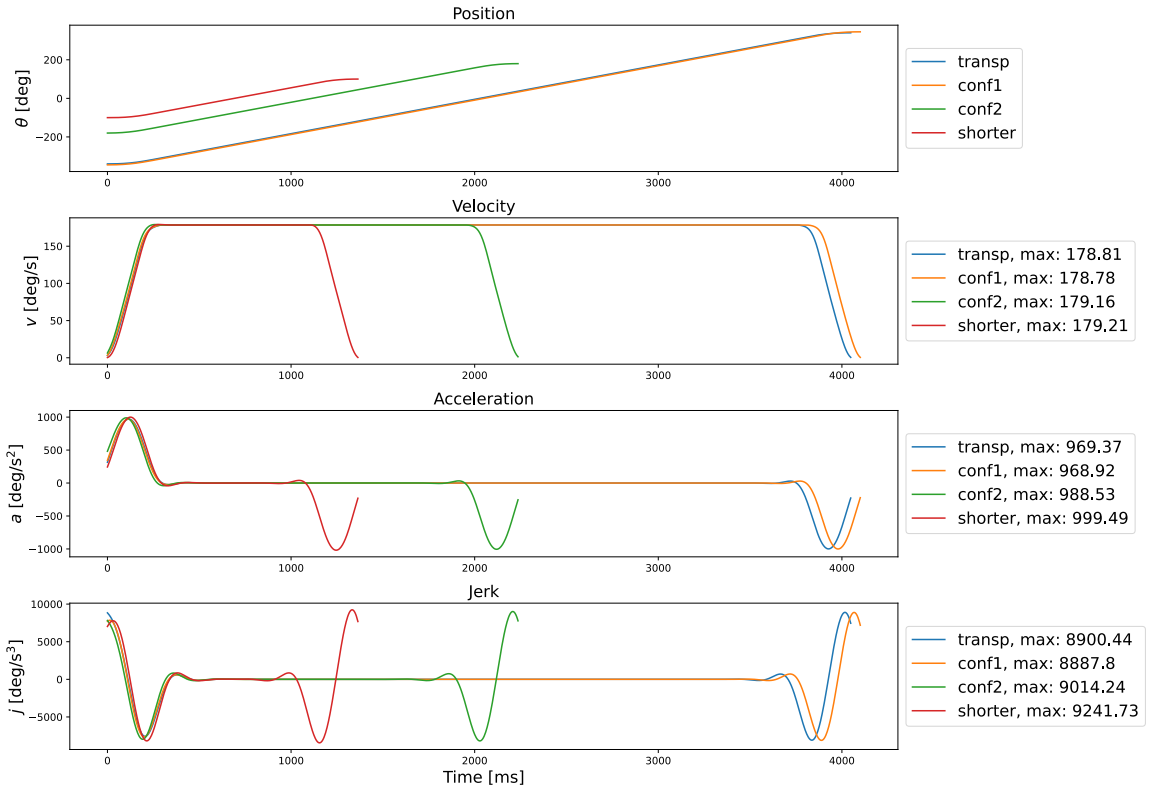

Figure A-6: Robot joint: A4, Profiles comparison between similar motions executed with no payload but different robot poses.

A-2-3 Joint A5

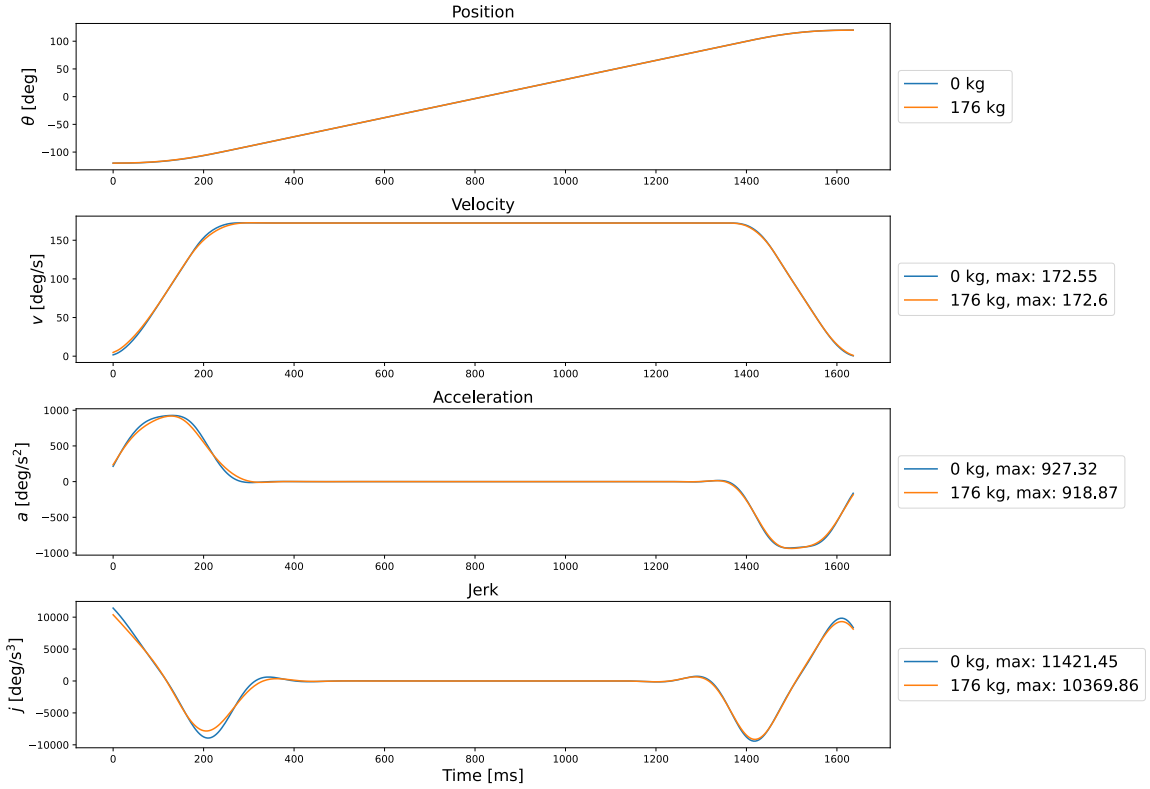

Figure A-7: Robot joint: A5, Profiles comparison between the same motion executed with the same robot pose 'transp' but different payloads.

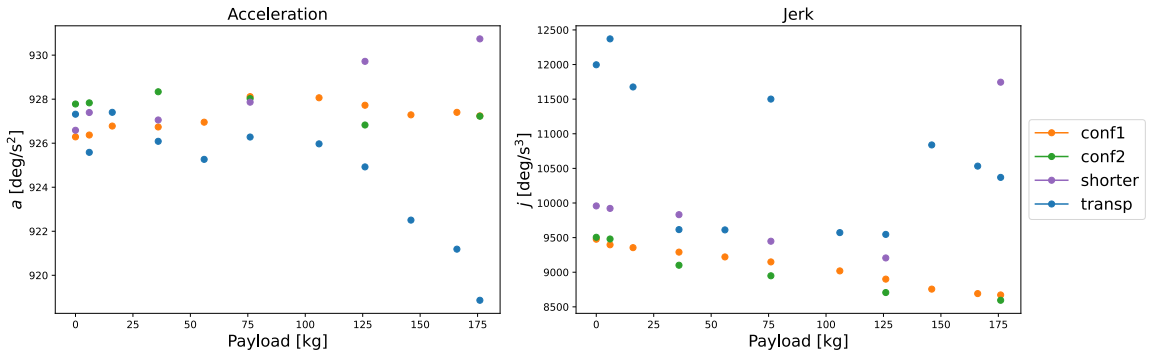

Figure A-8: Robot joint: A5, Maximum acceleration and jerk values spread depending on the payload weight and robot pose.

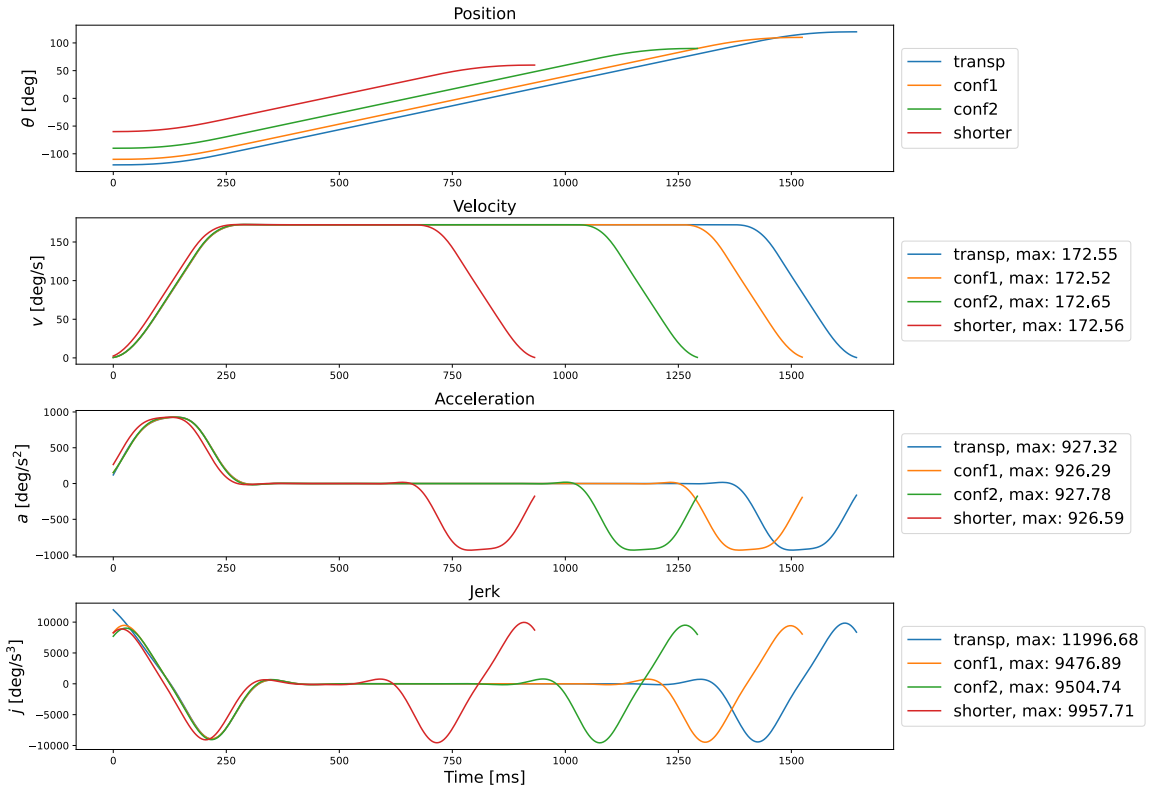

Figure A-9: Robot joint: A5, Profiles comparison between similar motions executed with no payload but different robot poses.

A-2-4 Joint A6

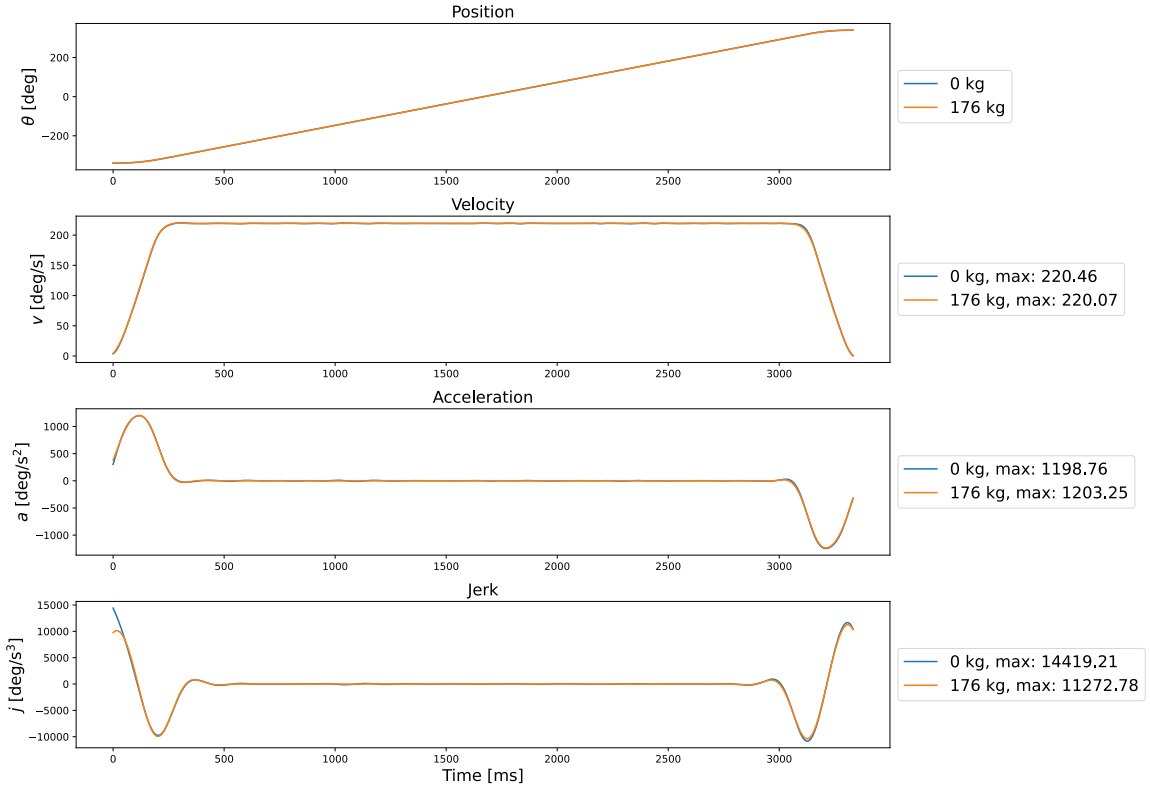

Figure A-10: Robot joint: A6, Profiles comparison between the same motion executed with the same robot pose 'transp' but different payloads.

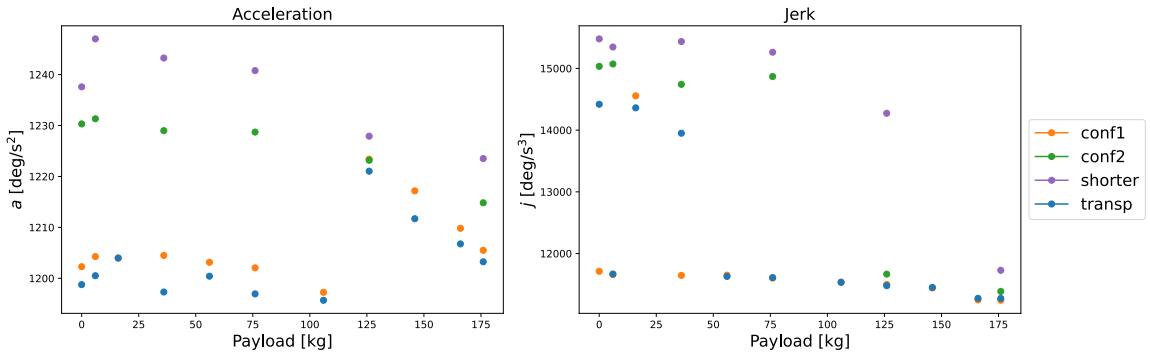

Figure A-11: Robot joint: A6, Maximum acceleration and jerk values spread depending on the payload weight and robot pose.

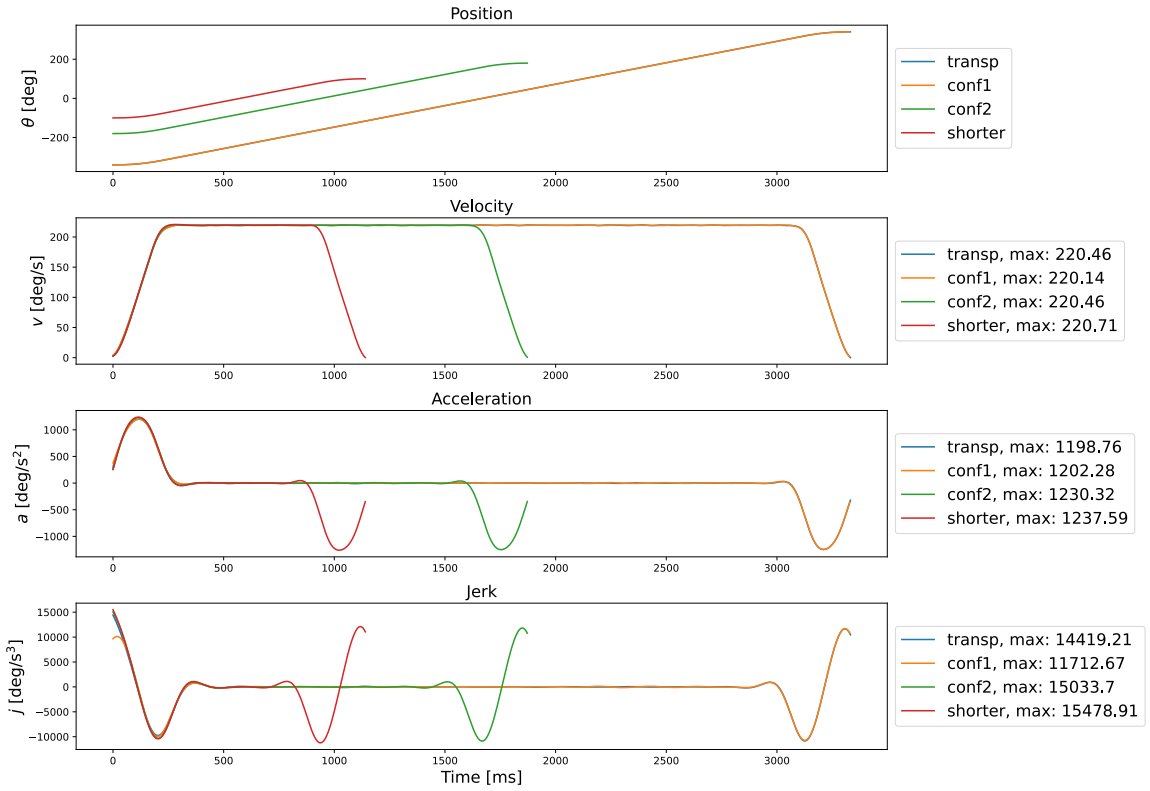

Figure A-12: Robot joint: A6, Profiles comparison between similar motions executed with no payload but different robot poses.

A-2-5 Joint E1

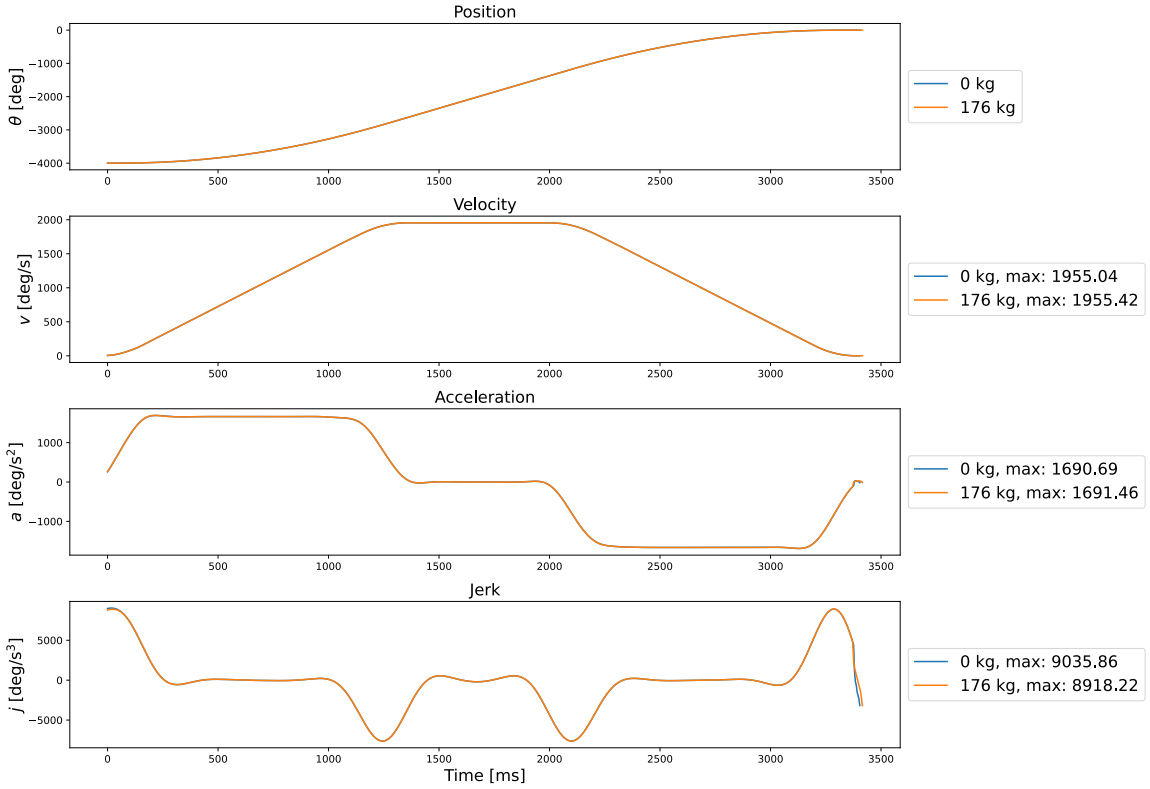

Figure A-13: Robot joint: E1, Profiles comparison between the same motion executed with the same robot pose 'transp' but different payloads

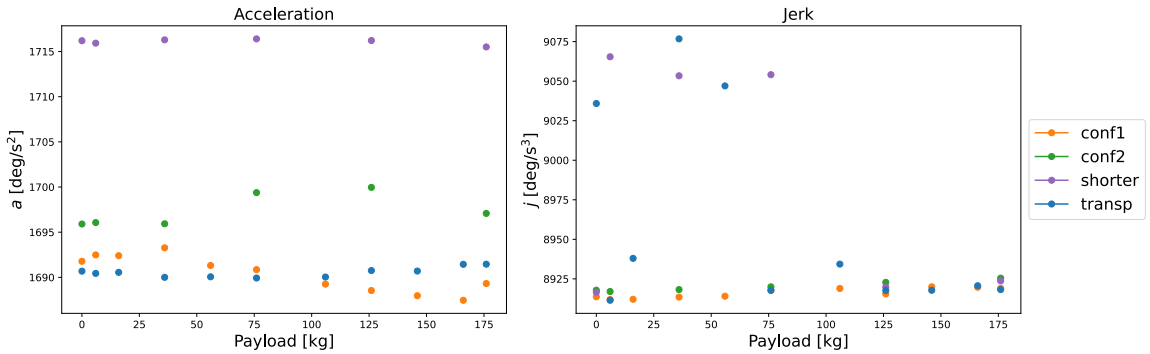

Figure A-14: Robot joint: E1, Maximum acceleration and jerk values spread depending on the payload weight and robot pose.

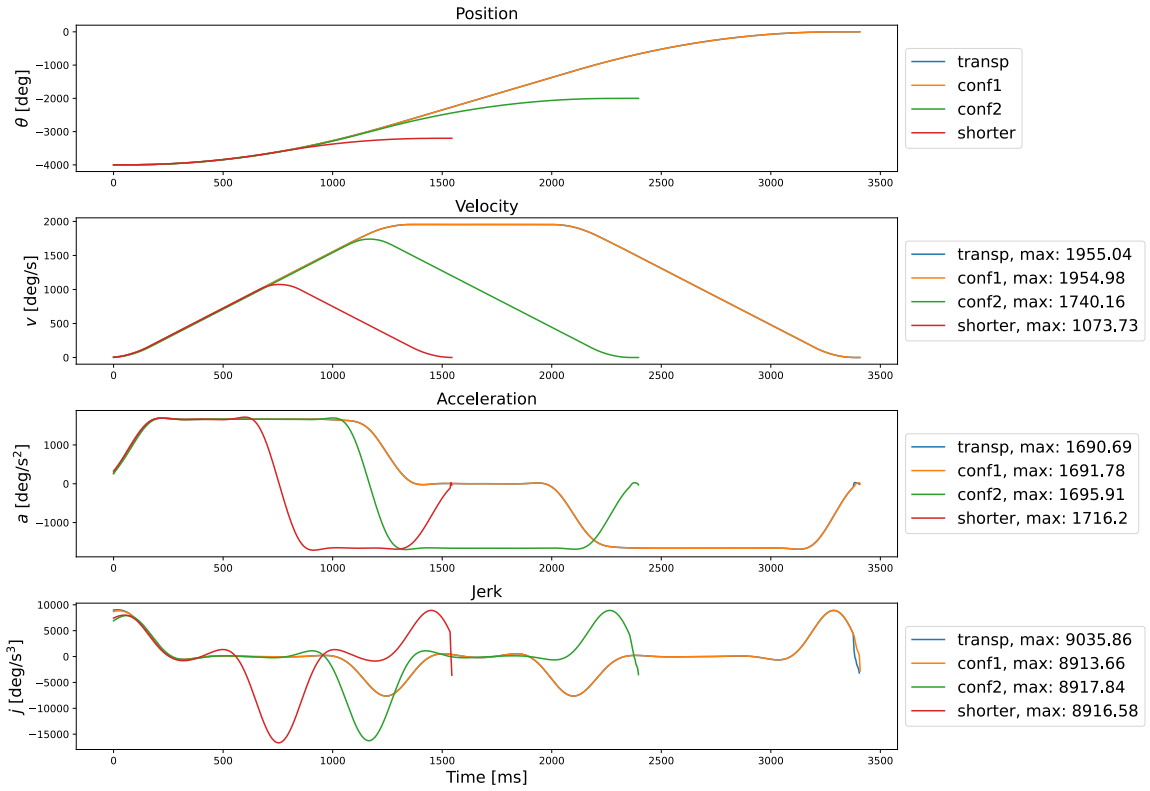

Figure A-15: Robot joint: E1, Profiles comparison between similar motions executed with no payload but different robot poses

Bibliography

[1] D. Gates, "Boeing abandons its failed fuselage robots on the 777X, handing the job back to machinists." www.seattletimes.com/business/boeing-aerospace/boeing-abandons-its-failed-fuselage-robots-on-the-777x-handing-the-job-back-to-machinists, 2019. Accessed: 2023-03-15.

[2] KUKA Roboter GmbH, "KR System Software 8.3, Operating and Programming Instructions for System Integrators," 2015.

[3] KUKA Roboter GmbH, "KR QUANTEC extra With F and C Variants Specification," 2015.

[4] KUKA Roboter GmbH, "KR System Software 8.3, Operating and Programming Instructions for End Users," 2013.

[5] KUKA Roboter GmbH, "Industrial robotics _ Linear units and positioners - Catalog," 2017.

[6] Houstex, "A Guide to Automated Manufacturing Systems." www.houstexonline.com/news/guide-to-automated-manufacturing-systems, 2023. Accessed: 2023-03-15.

[7] Britannica, "Manufacturing applications of automation and robotics." www.britannica.com/technology/automation/Manufacturing-applications-of-automation-and-robotics, 2023. Accessed: 2023-03-15.

[8] A. Owen-Hill, "10 Fantastic Examples of Flexible Manufacturing." www.robodk.com/blog/10-examples-of-flexible-manufacturing, 2023. Accessed: 2023-03-15.

[9] Britannica, "Robots in manufacturing." www.britannica.com/technology/automation/Robots-in-manufacturing, 2023. Accessed: 2023-03-15.

[10] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010.

[11] D. Berrar, "Cross-validation," in *Encyclopedia of Bioinformatics and Computational Biology* (S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, eds.), pp. 542–545, Oxford: Academic Press, 2019.

[12] J. Gross, *Linear regression*, vol. 175. Springer Science & Business Media, 2003.

[13] X. Su, X. Yan, and C. Tsai, "Linear regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, 05 2012.

[14] M. Schonlau and R. Y. Zou, "The random forest algorithm for statistical learning," *The Stata Journal*, vol. 20, no. 1, pp. 3–29, 2020.

[15] "Documentation on the Random Forest Regressor from Scikit-learn." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. Accessed: 2023-02-02.

[16] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[17] L. Berscheid and T. Kröger, "Jerk-limited real-time trajectory generation with arbitrary target states," *CoRR*, vol. abs/2105.04830, 2021.

Glossary

List of Acronyms

CNC	Computer Numerical Control
KSS	KUKA System Software
KRL	KUKA Robot Language
RSI	Robot Sensor Interface
UDP	User Datagram Protocol
PTP	Point-to-point motions
LIN	Linear motions
CIRC	Circular motions
CoM	Centre of Mass
DoF	Degrees of Freedom
BFD	Backward Finite Differences
FFD	Forward Finite Differences
FD	Finite Differences
ROS	Robot Operating System
CV	Cross-validation
LR	Linear Regression
RF	Random Forest
XGBoost	Extreme Gradient Boosting
ATL	Automated Tape Layering
AFP	Automated Fiber Placement
UDP/IP	User Datagram Protocol/Internet Protocol
CSV	Comma-Separated values
XML	Extensible Markup Language

List of Symbols

α	Prediction objective function conservativeness parameter
β_1, ... β_p	Linear regression coefficients
$\Delta\theta$	Motion distance
Δt	Sampling time
ϵ	Soft constraint slack variable
γ	Prediction objective function tunning parameter
$\Phi(\hat{z}, z)$	Prediction objective function
θ_{final}	Final motion position
θ_{offset}	Positional offset at the end of simulation
θ_{ref}	Reference position
θ_{start}	Starting position
$^\circ$, [deg]	Degrees
$\bar{a}$	Acceleration upper bound
$\bar{j}$	Jerk upper bound
$\bar{v}$	Velocity upper bound
$\hat{f}$	Bounds mapping
$\hat{x}_h$	State estimation
$\hat{z}$	Predicted value
$\mathcal{B}(k+1)$	Next step bounds
θ	Position
θ_g	Generated position
θ_r	Recorded position
$\underline{a}$	Acceleration lower bound
$\underline{j}$	Jerk lower bound
$\underline{v}$	Velocity lower bound
a	Acceleration
A, A_c	Systems state matrix
a_g	Generated acceleration
a_r	Acceleration of recorded position
$a_{\max}$	Maximum acceleration
$a_{\min}$	Minimum acceleration
B	Linear system input matrix
C, C_c	Systems output matrix
d_o	Distance offset
e	Linear regression intercept
f	Butterworth filter frequency
f_0	Butterworth filter cut-off frequency
j	Jerk

j_g	Generated jerk
j_r	Jerk of recorded position
$j_{\max}$	Maximum jerk
$j_{\min}$	Minimum jerk
K	Kalman gain
k_{change}	Time sample of reference change
m	Butterworth filter order
m_w	Weight mass on each side
N_{steps}	Number of steps
n_{p_w}	Normalized payload weight
p_m	Constant motion parameters
p_w	Payload weight
$p_{\max}$	Rated payload
Q_2	Velocity reference parameter - optimization problem
r	Distance between barbell center to the first weight, along the bar
t_w	Total disks' width on each side
v	Velocity
v_g	Generated velocity
v_r	Velocity of recorded position
$v_{\text{adm}_{\max}}$	Maximum admissible velocity
$v_{\max}$	Maximum velocity
$v_{\min}$	Minimum velocity
v_{ref}	Reference velocity
x	Linear system state
x_g	Generated state
y	Linear system output
y_p	Predicted value
y_r	Real value
z	Value to predict
I_{xbb}	Barbell inertia term on the x-Axis
I_{ybb}	Barbell inertia term on the y-Axis
I_{zbb}	Barbell inertia term on the z-Axis
X, x_1 ... x_p	Linear regression features
Y	Linear regession output