Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Dynamic Control of Collaborative Robot KUKA LBR iiwa - Demonstrator System

Maroš Mešter

Supervisor: Ing. Jiří Zemánek, Ph.D.
Field of study: Cybernetics and Robotics
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Mešter Maroš**	Personal ID number:	**507205**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Dynamic Control of Collaborative Robot KUKA LBR iiwa - Demonstrator System

Bachelor’s thesis title in Czech:

Dynamické řízení kolaborativního robota KUKA LBR iiwa - demonstrační systém

Guidelines:

The aim of this work is to explore and demonstrate the possibilities of dynamic control of the collaborative robot KUKA LBR iiwa. The project should cover the design of a suitable hardware setup for real-time robot control, provide an overview and comparison of various robot control methods, and investigate possibilities of feedforward and feedback control along pre-planned trajectories. The outcome of the work should be a demonstrator system showcasing dynamic control.
Tasks:
1. Design and document a complete hardware setup for real-time robot control, including safety features, sensor and actuator interfaces, and a local control computer.
2. Conduct an analysis and comparison of different robot control methods, including Fast Robot Interface (FRI), Matlab KUKA Sunrise Toolbox, and Robot Operating System (ROS), with a focus on easy development and testing of feedback controllers in Matlab/Simulink environment.
3. Design and implement demonstrative scenarios of dynamic robot control. For example: pendulum passing between obstacles, Kendama game, inverse spherical pendulum, or playing melodies on saw.
4. Create a mathematical model of the dynamics of the demonstrator system and propose suitable control strategies.
5. Perform simulations and experiments both on the mathematical model and on the real system, and evaluate their results.

Bibliography / sources:

[1] M. N. Vu, C. Hartl-Nesic, and A. Kugi, „Fast swing-up trajectory optimization for a spherical pendulum on a 7-dof collaborative robot,“ 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 10114-10120, 2021.
[2] C. Hartl-Nesic, J. Kretschmer, M. Schwegel, T. Glück, and A. Kugi, “Swing-up of a spherical pendulum on a 7-axis industrial robot,” IFAC-PapersOnLine, vol. 52, no. 15, pp. 346–351, 2019.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jiří Zemánek, Ph.D.   Department of Control Engineering FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **16.02.2024**   Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Jiří Zemánek, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I want to express my sincere gratitude to my advisor and thesis supervisor, Ing. Jiří Zemánek, Ph.D, for his invaluable guidance and support throughout the whole development process. I would also like to attribute special thanks to Ing. Krištof Pučejdl for his help with the construction of the polystyrene blocks and aluminium pendulum rod. Finally, I would like to thank my family and friends for their overwhelming support and encouragement.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2024

Signature

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe dňa 24. mája 2024

podpis autora práce

Abstract

In this bachelor thesis, we investigate the possibilities of dynamic control of the collaborative robot KUKA LBR iiwa and its incorporation into dynamic control experiments. We begin by documenting the setup for robot control from external computer and provide a comprehensive comparison of software options for robot control, including KUKA Sunrise Toolbox, Fast Robot Interface and ROS 2. Subsequently, we have developed two demonstrations showcasing dynamic control which involved regulating oscillations of a ball attached to the robot's end effector by a string. This feedforward control was performed by indirectly controlling the end effector acceleration via real-time position commands. Additionally, we have developed a physical prototype of a pendulum module, which can be connected to the end effector and provide feedback data intended for more advanced control tasks.

Keywords: KUKA LBR iiwa, manipulator dynamic control, Fast Robot Interface, KUKA Sunrise Toolbox, demonstration of dynamic control

Supervisor: Ing. Jiří Zemánek, Ph.D.
Resslova 307/9
12000 Praha 2

Abstrakt

V tejto bakalárskej práci skúmame možnosti dynamického riadenia kolaboratívneho robota KUKA LBR iiwa a jeho využitie v experimentoch obnášajúcich dynamické riadenie. V úvode popisujeme konfiguráciu externého počítača určeného na ovládanie robota. Zároveň poskytujeme obsiahle porovnanie softvérových možností ovládania robota, vrátane softvérového doplnku KUKA Sunrise Toolbox, rozhrania Fast Robot Interface a systému ROS 2. Následne prezentujeme dve demonštrácie dynamického riadenia, ktoré spočívali v regulácii výchylky guľôčky voľne zavesenej na koncovom efektore robota. Toto dopredné riadenie bolo uskutočnené skrz nepriame ovládanie zrýchlenia koncového efektora pomocou posielania polohových príkazov v reálnom čase. V závere práce popisujeme vývoj a fyzickú konštrukciu kyvadlového modulu, ktorý môže byť pripojený na koncový efektor robota a poskytovať spätnú väzbu určenú pre pokročilejšie úlohy riadenia.

Kľúčové slová: KUKA LBR iiwa, dynamické riadenie manipulátora, Fast Robot Interface, KUKA Sunrise Toolbox, demonštrácia dynamického riadenia

Preklad názvu: Dynamické riadenie kolaboratívneho robota KUKA LBR iiwa - demonštračný systém

Contents

Figures

Tables

Chapter 1
Introduction

In today's world, robotic manipulators are no longer just slow, heavy machines used in industrial settings. Modern collaborative robots, besides being suitable for human-robot interaction, are often lightweight, kinematically redundant, capable of achieving higher joint accelerations, and come equipped with force or torque sensors. All of these characteristics make collaborative robots a valuable tool that can be utilized for various research purposes.

A particularly interesting use case for robotic manipulators is their incorporation into dynamic control experiments. In these experiments, manipulators are often used to drive, stabilize or otherwise actuate a mechanical system with dynamic behaviour. One of the most popular robots for dynamic control is the KUKA LBR iiwa. This is mainly due to the fact that this manipulator can reach sufficient acceleration and is equipped with an interface for high speed communication (Fast Robot Interface). It has been documented that this robot in particular was programmed to, for instance, successfully swing up and stabilize a spherical pendulum [2].

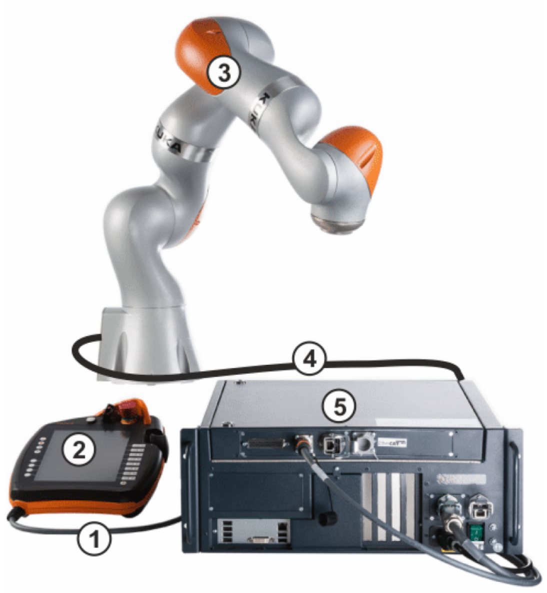


Figure 1.1: KUKA LBR iiwa robot system. Image source: [1]

KUKA LBR iiwa was developed by KUKA Deutschland GmbH and primarily designed for delicate assembly work and human-robot collaboration. This work will focus on the KUKA LBR iiwa 7 R800 model. The complete robot system, depicted in Figure 1.1, consists of:

- the manipulator,
- robot controller (KUKA Sunrise Cabinet),
- control panel (KUKA smartPAD).

This thesis aims to investigate how the KUKA LBR iiwa can be incorporated into demonstrations of dynamic control. Imagine a scenario in which an underactuated dynamical system, for example, a rigid pendulum, is attached to the end effector of LBR iiwa (as illustrated in Figure 1.2). Let us assume that the goal in this scenario is to stabilize the pendulum in its upright position. Traditional method for manipulator control, i.e. programming the manipulator to follow a precomputed trajectory with a constant velocity, is not sufficient to fulfil this goal. For this reason, a real-time control system which takes into account the attached pendulum is necessary. To build this control system, a feedback on the states of the pendulum as well feedback data from the manipulator should be fed into a control algorithm (most probably running on an external machine), output of which should be sent back to the robot controller that can adjust the manipulator motion accordingly. This concept is clearly illustrated in Figure. 1.3

The structure of this thesis is as follows. In Chapter 2 we provide a description of the KUKA LBR iiwa 7 R800 model and document the steps we undertook in order commission the robot and set up the network communication necessary for the integration of an external computer into the control structure from Figure 1.3.

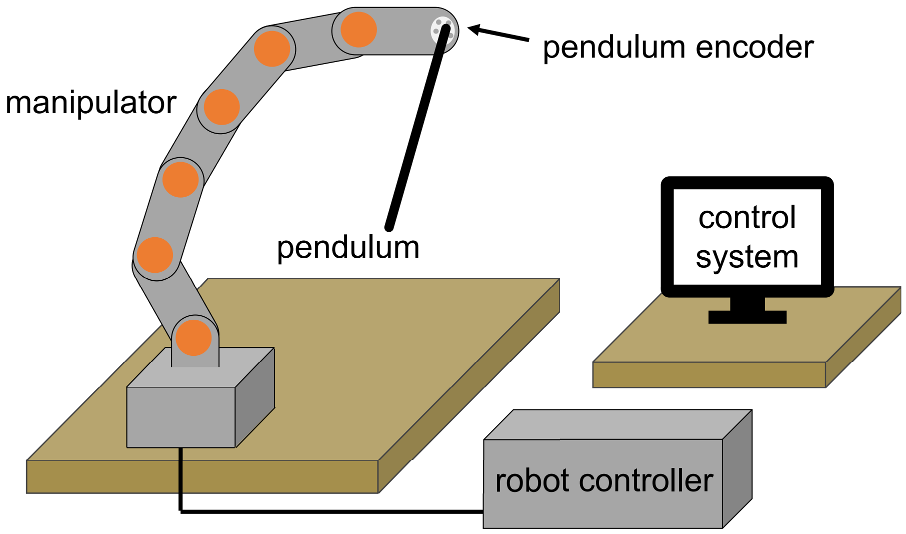


Figure 1.2: Example scenario showcasing dynamic control of a rigid pendulum via the KUKA LBR iiwa manipulator.

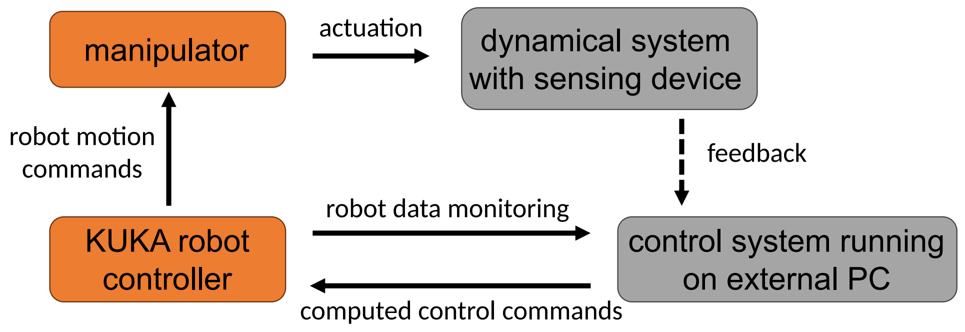

manipulator
actuation
dynamical system with sensing device
robot motion commands
feedback
robot data monitoring
KUKA robot controller
control system running on external PC
computed control commands

Figure 1.3: Diagram illustrating the ideal control structure for dynamic control experiments on LBR iiwa.

Chapter 3 consists of an analysis of software options and control methods available for LBR iiwa. In Chapter 4 we present a working demonstration of a feedforward control of the manipulator utilizing the MATLAB KUKA Sunrise Toolbox. Chapter 5 showcases how the Fast Robot Interface can be used to send motion commands to the robot in real-time and thus achieve a desired end effector acceleration. Finally, Chapter 6 documents the construction of a pendulum module mountable to the robot's end effector, which, thanks to the pendulum's encoder, can provide a feedback of its states.

Chapter 2
Robot commissioning and hardware

2.1 Manipulator description

The LBR iiwa manipulator (Figure 2.1a) is a 7 DOF redundant kinematic open chain (RRRRRRR). Every joint contains position, torque and temperature sensors that provide signals for robot control and safety mechanisms [1]. Denavit-Hartenberg (DH) parameters of the manipulator were not provided by the manufacturer, however they have already been measured in other works, such as [3] and [4]. DH parameters differ based on the mounted flange option. As we have used a robot with the *Touch Electric* flange, the DH parameters for our manipulator can be found in Table 2.1. The orientation of the robot base frame is illustrated in Figure 2.1b.

For the purposes of dynamic control, it is crucial to know the mechanical joint limits, limits for joint velocity and other higher-order time derivatives of position. KUKA provides the permissible range for angular joint positions and the maximum joint velocities in the technical specification of the robot [1], but does not list the limits for joint acceleration or jerk. Fortunately, these have also been measured experimentally in similar works (see [4]). All data related to mechanical motion limits of KUKA LBR iiwa can be found in Tables 2.2 and 2.3.

Table 2.1: DH parameters of the KUKA LBR iiwa 7 R800 manipulator equipped with the media flange *Touch Electric*. Data by [4].

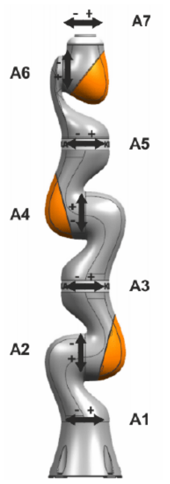
	θ	d	a	α
$0 \rightarrow 1$	θ_1	0.340	0	$-\frac{\pi}{2}$
$1 \rightarrow 2$	θ_2	0	0	$\frac{\pi}{2}$
$2 \rightarrow 3$	θ_3	0.400	0	$\frac{\pi}{2}$
$3 \rightarrow 4$	θ_4	0	0	$-\frac{\pi}{2}$
$4 \rightarrow 5$	θ_5	0.400	0	$-\frac{\pi}{2}$
$5 \rightarrow 6$	θ_6	0	0	$\frac{\pi}{2}$
$6 \rightarrow 7$	θ_7	0.152	0	0

Table 2.2: Joint angular position range and maximal joint velocity of the LBR iiwa 7 R800 model as stated in the official technical specification by KUKA [1].
*The technical specification states that the (maximum) rated payload is 7 kg.

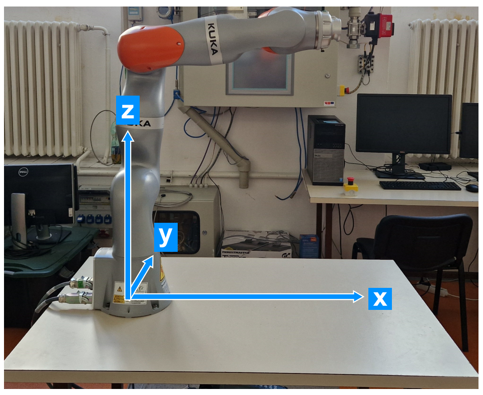
Joint	Angular position range	Velocity limit*
A1	± 170°	98 °/s
A2	± 120°	98 °/s
A3	± 170°	100 °/s
A4	± 120°	130 °/s
A5	± 170°	140 °/s
A6	± 120°	180 °/s
A7	± 175°	180 °/s

Table 2.3: Experimentally determined maximal values of joint accelerations and jerks of the LBR iiwa 7 R800 model obtained by C. Larsen in [4].

Joint	Maximum measured acceleration [$^\circ/s^2$]	Acceleration standard deviation [$^\circ/s^2$]	Maximum measured jerk [$^\circ/s^3$]	Jerk standard deviation [$^\circ/s^3$]
A1	491	0.3	10384	697.8
A2	491	0.3	7757	754.5
A3	501	0.0	9986	464.8
A4	651	0.3	9700	674.9
A5	701	0.0	19406	128.9
A6	901	0.1	32260	453.6
A7	901	0.0	33028	357.8



(a) : Names and orientations of the manipulator's axes. Image source: [1]



(b) : Orientation of the robot base frame. The x and y axes are perpendicular to the table edges and the $z = 0$ is at the the table level.

Figure 2.1: KUKA LBR iiwa robot system.

2.2 External computer

2.2.1 Computer setup

In order to be able to configure and write software applications for the robot, a separate computer has to be connected to the KUKA Sunrise Cabinet robot controller. We have decided to designate the Dell OptiPlex 9020 MT (Figure 2.2) as the computer for this role. We believe that this machine is a decent choice, as it has sufficient computing power (Intel Core i7-4790 CPU) and memory (16 GB DDR3) for executing real-time client applications and has good networking and I/O options (2 built-in LAN ports, multiple USB slots).

The official software for commissioning, configuring and programming of LBR iiwa is called `KUKA Sunrise Workbench`. Considering both the fact that this software is natively distributed for Windows OS only, and the fact that we wanted to develop applications in C++/ROS 2 for which Linux-based OS are usually the better choice, we opted to the install a dual-boot setup on the computer. The options for booting that we chose were Windows 10 and Ubuntu 24.04 LTS.

2.2.2 Networking

The default LAN port on the robot controller is the KLI (X66) port. This port is used to synchronize `Sunrise Workbench` projects between the robot controller and the user PC. In order to utilize the real-time capable Ethernet connection provided by the Fast Robot Interface - FRI (see section 3.4), an additional physical connection between the controller and the user PC needed to be established using the KONI port on the controller. We connected the KLI and the KONI ports directly with the network interface cards (NIC) in our desktop PC as the KUKA manual for the FRI [5] advises not to connect any additional network devices. The local network consisting of the robot controller and the desktop PC needed additional configuration. Based on the instructions provided in [6], we set the IP addresses of the devices to static and arranged for the FRI communication to occur within a distinct subnet. Detailed specification of our local network configuration is presented in Table 2.4. This network configuration has been set on both operating systems.

2.3 External emergency stop device

The LBR iiwa can operate in one of its three modes of operation:

- T1 (test mode, Cartesian end effector velocity reduced to 250 mm.s^{-1}),
- T2 (test mode, no velocity limit),
- or AUT (automatic mode).

Although the smartPAD control panel is equipped with an integrated emergency stop button, without any additional safety features, the robot is only able to operate

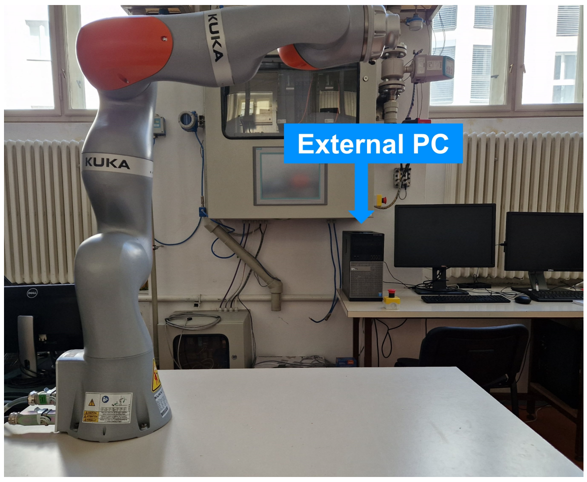


Figure 2.2: Photo of the external PC.

Table 2.4: Configuration of the local network for interfacing the KUKA LBR iiwa.

		static IP address	subnet mask
Sunrise Workbench project synchronization	robot controller	172.31.1.147	255.255.0.0.
	PC: NIC 1	172.31.1.149	255.255.0.0.
Fast Robot Interface	robot controller	192.170.10.2	255.255.255.0.
	PC: NIC 2	192.170.10.3	255.255.255.0.

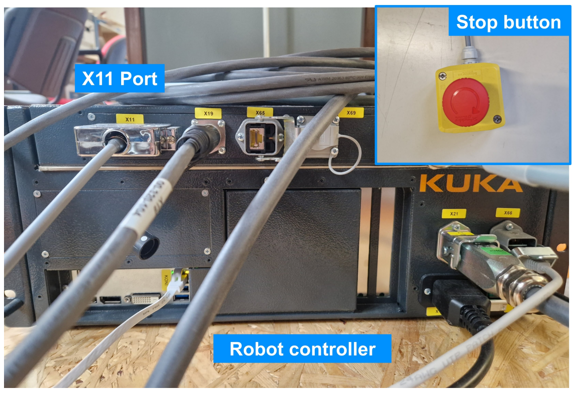
in the T1 mode [6]. For this reason, we have installed an external emergency stop button. The button that we chose consists of 2 normally closed contacts, which we connected with corresponding pins of the X11 port on the robot controller using a 4 core shielded cable, see Figure 2.3. The newly installed safety feature had to be configured in the `Sunrise Workbench` project. In the safety configuration file, in the customer PSM[1] tab, a new row representing the external emergency stop device was added, see Figure 2.4.

2.4 Camera

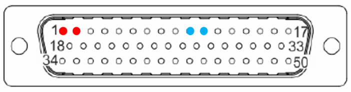
During some experiments (in Chapters 4 and 5) basic object tracking from a prerecorded video was necessary in order to sufficiently compare simulations with experiments. For this reason, a camera was added to our setup. It was a USB camera capable of recording up to 60 FPS. As can be seen in Figure 2.5, we aligned the camera to point in the direction perpendicular to the plane in which the experiments were happening and mounted it to the table using a camera stand. We have written a simple object tracking script in Python using the OpenCV[2] library. This script (available in Appendix A) enabled us to analyze a video from the experiment and plot the time evolution of object position.

[1]permanent safety-oriented monitoring

[2]Online documentation of the OpenCV library: `https://docs.opencv.org/4.x/index.html`. [Accessed: 2024-05-23]

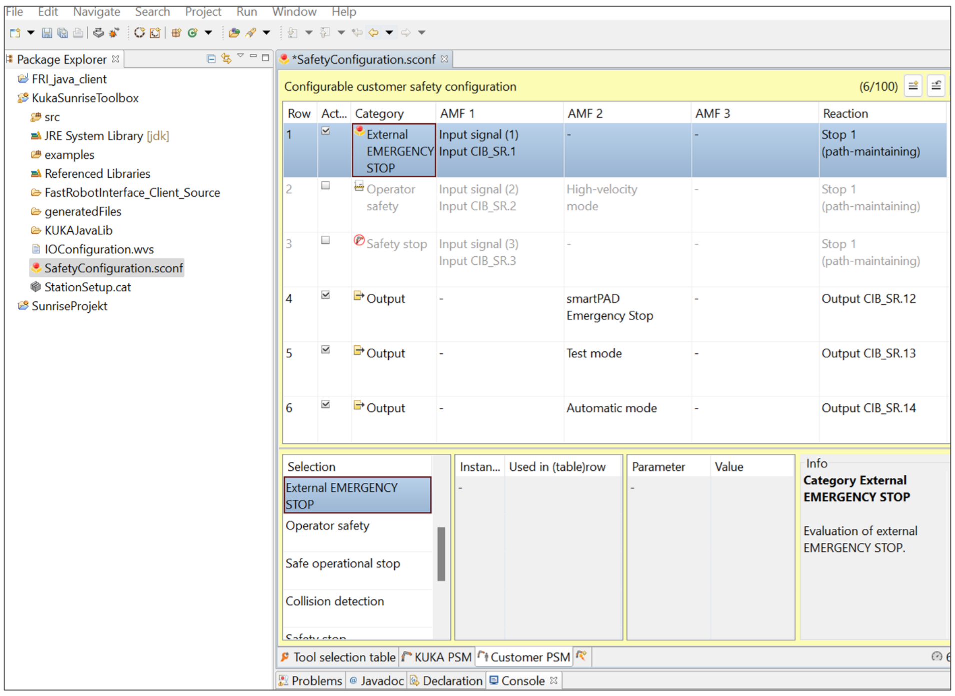


(a) : Back panel of the controller and the stop button.



(b) : D-SUB connector.

Figure 2.3: (a) The stop button and the X11 safety port on the back panel of the robot controller. (b) The D-SUB connector and the pins used for connecting the 2 channels of the emergency stop button. Channel A (red), channel B (cyan). The D-SUB connector is plugged into the X11 port on the robot controller. Pins have been chosen based on the recommendation in manual [7].



Configurable customer safety configuration (6/100)

Row	Act...	Category	AMF 1	AMF 2	AMF 3	Reaction
1	☑	External EMERGENCY STOP	Input signal (1) Input CIB_SR.1	-	-	Stop 1 (path-maintaining)
2	☐	Operator safety	Input signal (2) Input CIB_SR.2	High-velocity mode	-	Stop 1 (path-maintaining)
3	☐	Safety stop	Input signal (3) Input CIB_SR.3	-	-	Stop 1 (path-maintaining)
4	☑	Output	-	smartPAD Emergency Stop	-	Output CIB_SR.12
5	☑	Output	-	Test mode	-	Output CIB_SR.13
6	☑	Output	-	Automatic mode	-	Output CIB_SR.14

Selection
External EMERGENCY STOP
Operator safety
Safe operational stop
Collision detection
Safety stop

Instan...	Used in (table)row
-	

Parameter	Value
-	

Info
Category External EMERGENCY STOP

Evaluation of external EMERGENCY STOP.

Figure 2.4: Screenshot of the `SafetyConfiguration.sconf` file in our `Sunrise Workbench` project.

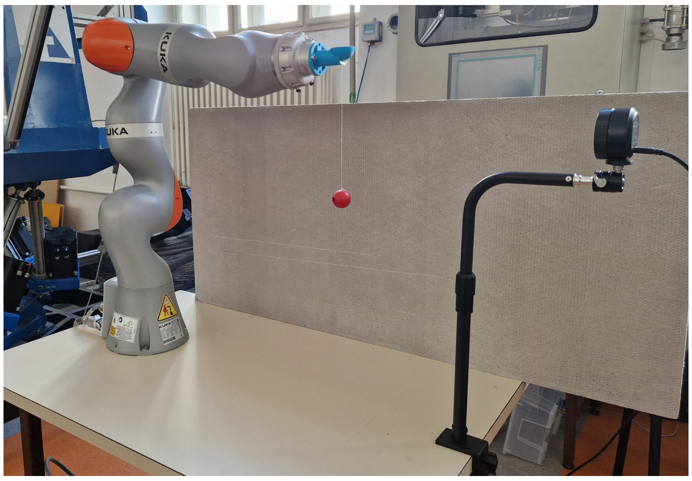


Figure 2.5: Camera mounted to the table with KUKA LBR iiwa.

Chapter 3

Analysis of different software options for robot control

3.1 Key factors of the analysis

The goal of this chapter is to provide an overview and a comprehensive comparison of different software options of controlling the KUKA iiwa manipulator in tasks where dynamic control is involved. Because of this, our analysis focused on assessment of capabilities of each software option in the following fields:

1. Diversity of motion control commands, i.e., the type of control commands a given software option provides. For example, joint position control, joint torque control or end effector position control.
2. Real-time control capabilities.
3. Matlab/Simulink integration.

The examined software options include:

- KUKA Sunrise.Workbench (the default software supplied with the manipulator),
- KUKA Sunrise Toolbox (Matlab add-on),
- Fast Robot Interface (FRI) with a custom user application,
- FRI integrated to the Robot Operating System (ROS).

3.2 KUKA Sunrise.Workbench

Sunrise.Workbench is the default software for the KUKA LBR iiwa manipulator supplied by the manufacturer. It is a complete development environment, meant to be installed on an external desktop computer running Windows [6]. It consists of **robot configuration tools** (tools for system software installation, safety configurations, I/O configurations) and **application development tools** – tools for programming robot applications in Java. These Java applications may then be uploaded to the robot's controller, from which they can be executed.

Table 3.1: List of "set" methods used to edit motion parameters of Java motion commands.

Method name	The method sets the:
`setCartVelocity()`	end effector's absolute velocity.
`setJointVelocityRel()`	relative velocity of robot joints.
`setCartAcceleration()`	end effector's absolute acceleration.
`setJointAccelerationRel()`	relative acceleration of robot joints.
`setCartJerk()`	end effector's absolute jerk.
`setJointJerkRel()`	relative jerk of robot joints.

Sunrise.Workbench provides a relatively wide range of Java classes for motion programming. For instance, the `PTP` object can either be used to move the robot to a given configuration in joint space, or to a given configuration in task space. Trajectory between these points is computed internaly and is the fastest one [6]. Furthermore, the objects `LIN`, `LINREL`, `CIR` and `SPL` can be used to move the end effector in task space along pre-programmed trajectories. Every motion instruction has its motion parameters, which can be set by the "set" methods in Table 3.1. Although this approach provides a way to set time derivatives of end effector's position up to the fourth derivative, the options are still quite limited. We have not found a native way of editing a given motion parameter during the execution of the trajectory - meaning it has to be constant. Another shortcoming is the fact that the "set" methods only accept positive numeric values [6], which in case of acceleration or jerk is quite limiting.

While the Sunrise.Workbench environment offers sufficient methods for motion programming along pre-planned trajectories, the environment alone, without additional Sunrise.Workbench software packages, lacks the functionalities for real-time control.

3.3 KUKA Sunrise Toolbox

Certain ports on the robot controller are enabled for communication via UDP or TCP/IP with external devices. KUKA Sunrise Toolbox (KST), developed by M. Safeea and P. Neto [8], takes advantage of this feature. KST is a free Matlab toolbox that serves as TCP/IP client application to a server application which also comes with the KST and has to be installed on the robot controller. Architecture and communication scheme of the KST can be seen in Figure 3.1.

Among other features, the toolbox provides functions for point-to-point motion, means of calculating kinematic properties (forward/inverse kinematics, Jacobian matrix...) and dynamic properties (mass matrix, forward/inverse dynamics...) and soft real-time control. It has to be noted that the user has to have access to the `DirectServo` and `SmartServo` software extensions by KUKA and these extensions ought to be installed on the robot controller prior to the set-up of KST in order to utilize the soft real-time control.

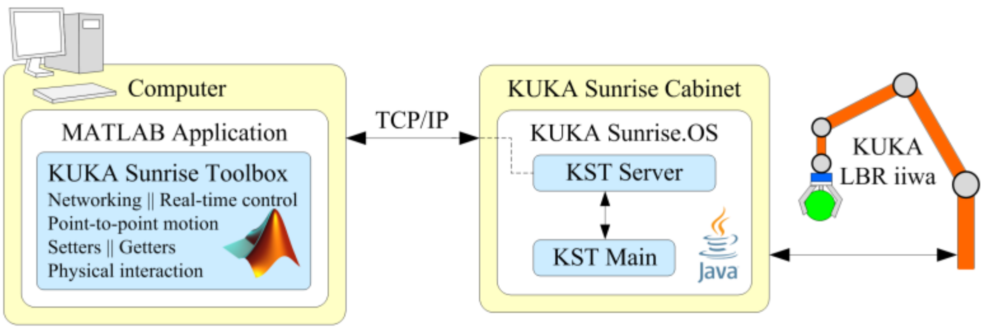


Figure 3.1: Architecture and communication scheme of the KUKA Sunrise Toolbox. Source: [8]

Table 3.2: Basic types of real-time commands provided by KST.

Method name	Description
`sendJointsPositions()`	Used to send target joint positions. Input is a 1×7 array in radians.
`sendEEfPosition()`	Used to send target position of the end effector. Input is a 1×6 cell array where the first 3 elements represent the target positions of EE (mm) and the last three elements represent the fixed rotation angles of EE (radians).
`sendJointsVelocities()`	Used to set target joint velocities. Input is a 1×7 cell array representing the target angular velocity for each joint, in rad.s^{-1}.

3.3.1 Offline motion commands

Because the KUKA Sunrise Toolbox is just an interface between Matlab and Sunrise.Workbench, the of set of offline motion commands is similar. PTP motion in joint space as well as in task space is available, however, only the *velocity* between the points can be specified by the programmer. Furthermore, the set of trajectories is extended by the option to move the end effector along a line with respect to the end effector reference frame and by the option to move the effector along a planar arc [9].

3.3.2 Real-time control

The KST offers soft real-time functionalities, claiming to be able to update the robot's motion at a frequency of 275 Hz [8],[9]. The highest communication frequency achieved on our setup was 63 Hz (measured using the time measurig constructs in MATLAB). Nevertheless, this speed is sufficient for large amount of tasks and the ease of implementation makes this an attractive approach for real-time control.

The basic types of commands used in real-time control of the KST are shown in Table 3.2. Notably, position control commands in joint space as well as position

Table 3.3: Real-time commands of the KUKA Sunrise Toolbox that also monitor a certain physical property.

Method name	**Return type**
sendEEfPositionExTorque()	Joint torques due to external forces acting on robot.
`sendEEfPositionMTorque()`	Absolute joint torques as measured by the sensors.
`sendEEfPositionGetActualJpos()`	Joint positions as measured by the encoders.
`sendEEfPositionGetActualJpos()`	Actual position and orientation of the end effector relative to the base frame of the robot.

control commands in Cartesian task space of the end effector are provided. In case of joint space, velocity control is also available. Additionally, all of the basic commands in Table 3.2 come in different versions, for instance `sendEEfPositionf()`, is the same as `sendEEfPosition()` but it is non-blocking, meaning it does not wait for an acknowledgment from the server before returning the execution. As a result, the `sendEEfPositionf()` is faster but could potentially cause some execution issues [9]. Other versions of the commands implement a way of receiving some data from the manipulator back to the external computer. List of KST methods that can return measured data can be found in Table 3.3

3.4 Fast Robot Interface

KUKA offers `Sunrise.FRI` software extension to the Sunrise.Workbench environment, which provides an additional way of interfacing the manipulator, called the Fast Robot Interface (FRI). FRI is an Ethernet-based UDP interface [5] via which data can be exchanged in real time between an application on the robot controller and an FRI client application on an external system. Hardware-wise, it is recommended to designate a separate networking cable to mediate the FRI connection, see section 2.2.2. The FRI architecture scheme is illustrated in Figure 3.2.

3.4.1 Real-time control

FRI allows hard real-time control of the manipulator at up to 1 kHz control loop rates [5]. Specifically, the cycle time for data transfer can be configurated, the available range is from 1 to 100 ms. One can use the FRI in order to *monitor* certain values or to *send* certain commands. The list of all the properties that can be monitored through FRI can be found in Table 3.4. As can also be seen in the aforementioned table, there are only 3 types of FRI client command modes. They are:

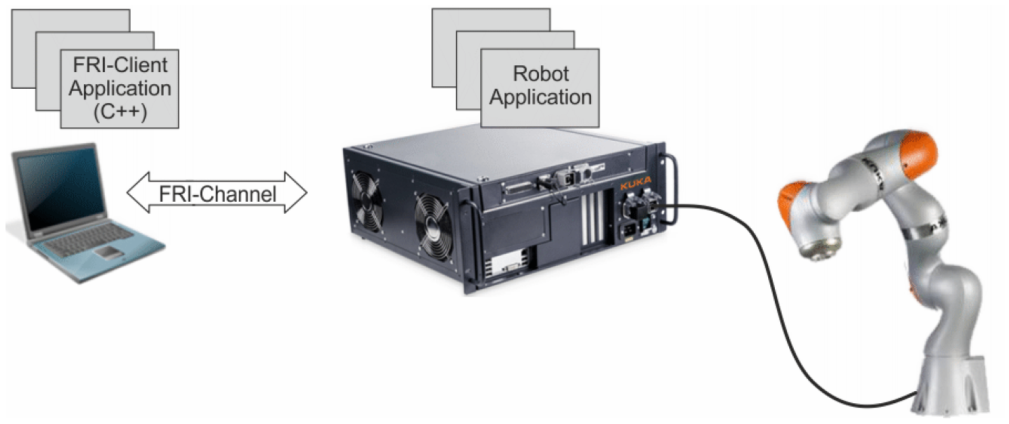


Figure 3.2: Architecture and communication scheme of the Fast Robot Interafce. Source: [5]

- **position command mode**: the client application continuously sends the the desired positions in joint space in radians, $[q_1, q_2, q_3, q_4, q_5, q_6, q_7]$,
- **torque command mode**: the client application continuously sends the the desired joint torques in Nm, $[\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7]$,
- **wrench command mode**: the client application continuously sends the end-effector wrench vector consisting of 6 elements, $[F_x, F_y, F_z, \tau_A, \tau_B, \tau_C]$, the first 3 elements being translational forces in Newtons along the axes XYZ and the next 3 elements being torques along the α, β, γ Euler angles (ZYX) in Nm.

It is important to note that the commanded FRI values are not applied directly, rather they are first sent to the controller's internal interpolator which plans the motion in real-time. It takes into account for example the dynamic limits of the machine. If the commanded values represent a motion that would violate those limits, the controller attempts to perform the motion in the closest form possible [5]. For this reason, the monitorable property "tracking performance" (see Table 3.4) is quite useful.

3.4.2 FRI in the robot controller

The Java application in the robot controller needs to be adjusted to be able to function as a component of the FRI communication scheme depicted in Figure 3.2. We have developed 3 Java applications for the robot controller (one for each client command mode) which were heavily inspired by examples provided by KUKA[1]. In contrast to the example files, our applications contain code snippets responsible for:

- moving the manipulator to a desired position before the start of the FRI communication,
- setting the total time duration of the FRI communication,

[1]The example files are proprietary scripts which are a part of the commercially available `KUKA Sunrise.FRI` software extension.

Table 3.4: Data available to monitor through FRI channel.

Data	**Note**
measured joint positions	[rad]
commanded joint positions	positions commanded by FRI user [rad]
measured joint torques	[Nm]
commanded joint torques	torques commanded by FRI user [Nm]
external joint torques	[Nm]
"Ipo" joint positions	positions commanded by internal interpolator [rad]
client command mode	POSITION, WRENCH, TORQUE, NO_COMMAND_MODE
tracking performance	the quality of tracking FRI commands sent to the manipulator $\in < 0, 1 >$, 1 being perfect tracking
sample time	the period of sending FRI packets, in [s]
digital IO value	$\{0, 1\}$
analog IO value	
session state	IDLE, MONITORING_WAIT, MONITORING_READY, COMMANDING_WAIT, COMMANDING_ACTIVE
connection quality	POOR, FAIR, GOOD, EXCELLET
safety state	NORMAL_OPERATION, STOP0, STOP1, STOP2
operation mode	T1, T2, AUT
drive state	OFF, TRANSITIONING, ACTIVE
overlay type	NO_OVERLAY, JOINT, CARTESIAN
control mode	POSITION_CONTROL, CART_IMP_CONTROL, JOINT_IMP_CONTROL, NO_CONTROL
time stamp	Unix time [s, ns]

- editing the stiffness and damping values (in case of torque or wrench control).

The algorithm for the commissioning of the FRI connection is depicted in Figure 3.3. In the figure, actual Java objects used in our applications are referenced. Interestingly, the FRI commands can be used to *overlay* Java motion commands (for example the `PTP` command), meaning they can be used to alter (and/or monitor) pre-programmed trajectories in real-time. In the case when the robot motion is to be controlled solely by the incoming FRI commands, the Java object of the `PositionHold` type can be used, which, by itself, does not move the robot, but can be overlayed by the incoming FRI commands.

Noteworthy third-party Java applications include the multipurpose `LBRserver.java` app[2] developed by M. Huber [10] which supports all 3 client

[2]The `LBRserver.java` file can be found at: `https://github.com/lbr-stack/fri/tree/ros2-fri-1.15/server_app`.

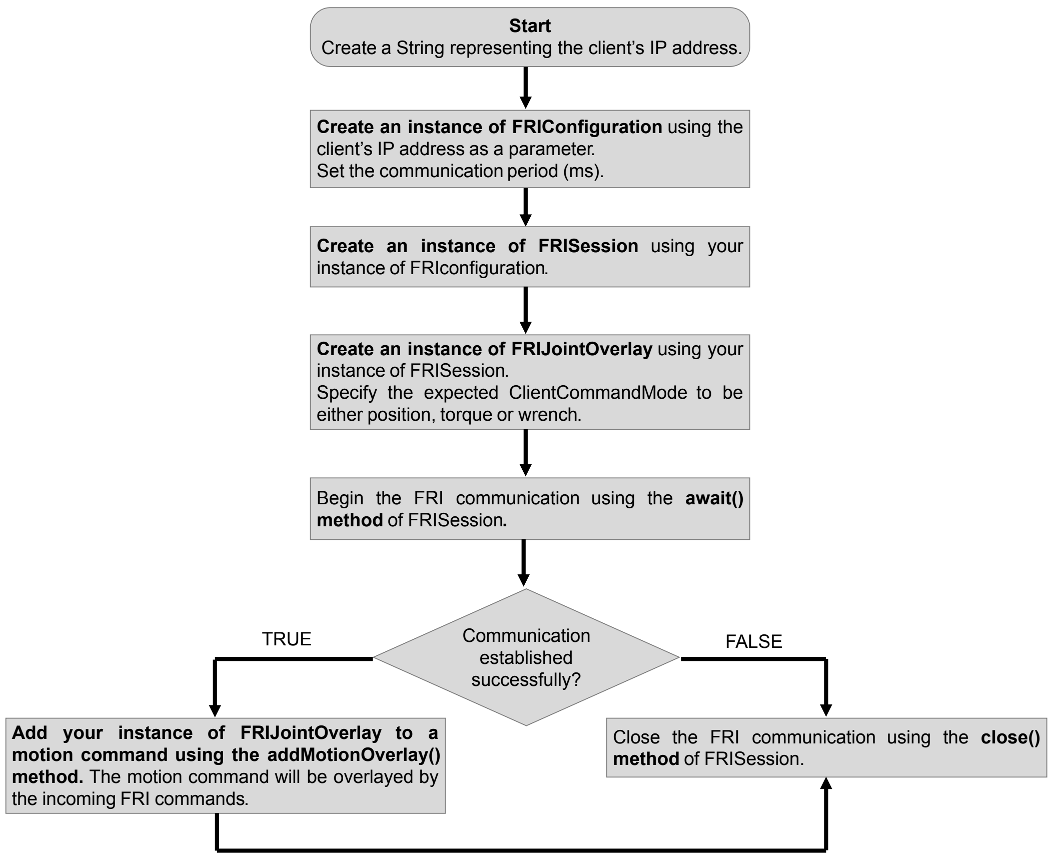

Start
Create a String representing the client's IP address.
Create an instance of FRIConfiguration using the client's IP address as a parameter.
Set the communication period (ms).
Create an instance of FRISession using your instance of FRIconfiguration.
Create an instance of FRIJointOverlay using your instance of FRISession.
Specify the expected ClientCommandMode to be either position, torque or wrench.
Begin the FRI communication using the await() method of FRISession.
Communication established successfully?
TRUE
FALSE
Add your instance of FRIJointOverlay to a motion command using the addMotionOverlay() method. The motion command will be overlayed by the incoming FRI commands.
Close the FRI communication using the close() method of FRISession.

Figure 3.3: Flowchart describing the steps necessary to initiate the FRI connection from the robot's controller side. In the flowchart, actual Java objects are referenced.

command modes and enables the user to configure FRI parameters through the GUI on the KUKA smartPAD.

3.4.3 FRI client application (C++)

FRI client application is an executable software script that runs on an external computer and communicates with the KUKA robot controller, see Figure 3.2. KUKA supplies the user with C++ or Java-based software development kits (SDK)[3] that facilitate the actual creation of the UDP connection and create blank methods for the user to fill in. We have decided to use the C++ version of the SDK. The following text aims to briefly explain the how the SDK works, list the necessary steps in FRI client application creation and present some generic client applications that we have made.

Structure and working principle of the SDK

The software development kit provided by KUKA consists of multiple C++ files (source and header files) and is written in the object-oriented programming paradigm.

[3]These software development kits were supplied with the `Sunrise.FRI` software extension. For this reason, they are not accessible online.

The simplified structure of the source code can be seen in Figure 3.4 in the form of a UML class diagram[4]. The user is only responsible for creating their own `UserClient` class and modifying the `monitor()` and `command()` methods to meet their needs. With the tools provided by the SDK at hand, the algorithm for communicating with the FRI-enabled application in the robot controller is quite straightforward. First, the client application creates an instance of the `ClientApplication` class (which incorporates an instance of the user-defined `UserClient` class). Then, if the FRI channel is successfully opened, the application cyclically calls the `step()` method which manages the process of receiving and sending UDP packets. The complete working principle of the FRI client application is illustrated in Figure 3.5.

Development of custom FRI client applications

The actual implementation of the `UserClient` class (see Figure 3.4) is task-specific and is up to the user to implement it. Here we provide a short list of supplementary C++ methods of the `UserClient` class that we have developed, as we believe that these methods are generic and useful enough to be utilized in future projects. These methods provide means for monitoring joint limit violations, writing and reading CSV files, calculating forward kinematics and other functionalities. We have gathered the methods into a single application template called `LBRMethods` which is available in Appendix A. Methods and necessary attributes are visualised in a UML diagram in Figure 3.6 and full description of the methods is provided in Table 3.5. Using these methods, we were able to create examples of FRI client applications, the purpose of which is to demonstrate the monitoring and commanding functionalities of the FRI. Here, we present 3 standalone applications:

1. `LBRMonitor`,
2. `LBRJointPTP`,
3. `LBRReadWrite`.

The `LBRMonitor` application showcases a way to record physical properties (for example the real-time joint torques) during the robot's motion and save them to a CSV file. Thanks to the method `FKtrans()` it also able to record the Cartesian position of the end effector. The `LBRJointPTP` showcases the method `ptpJointSpace()`. It expects the desired joint configuration and the desired time duration of the trajectory as a user input. Finally, the `LBRReadWrite` application reads a precomputed joint space trajectory from a CSV file, executes it and records data in the same way as the `LBRMonitor` application does. All of these applications can be found in Appendix A.

[4]Standard rules for a UML class diagrams: `https://en.wikipedia.org/wiki/Class_diagram`

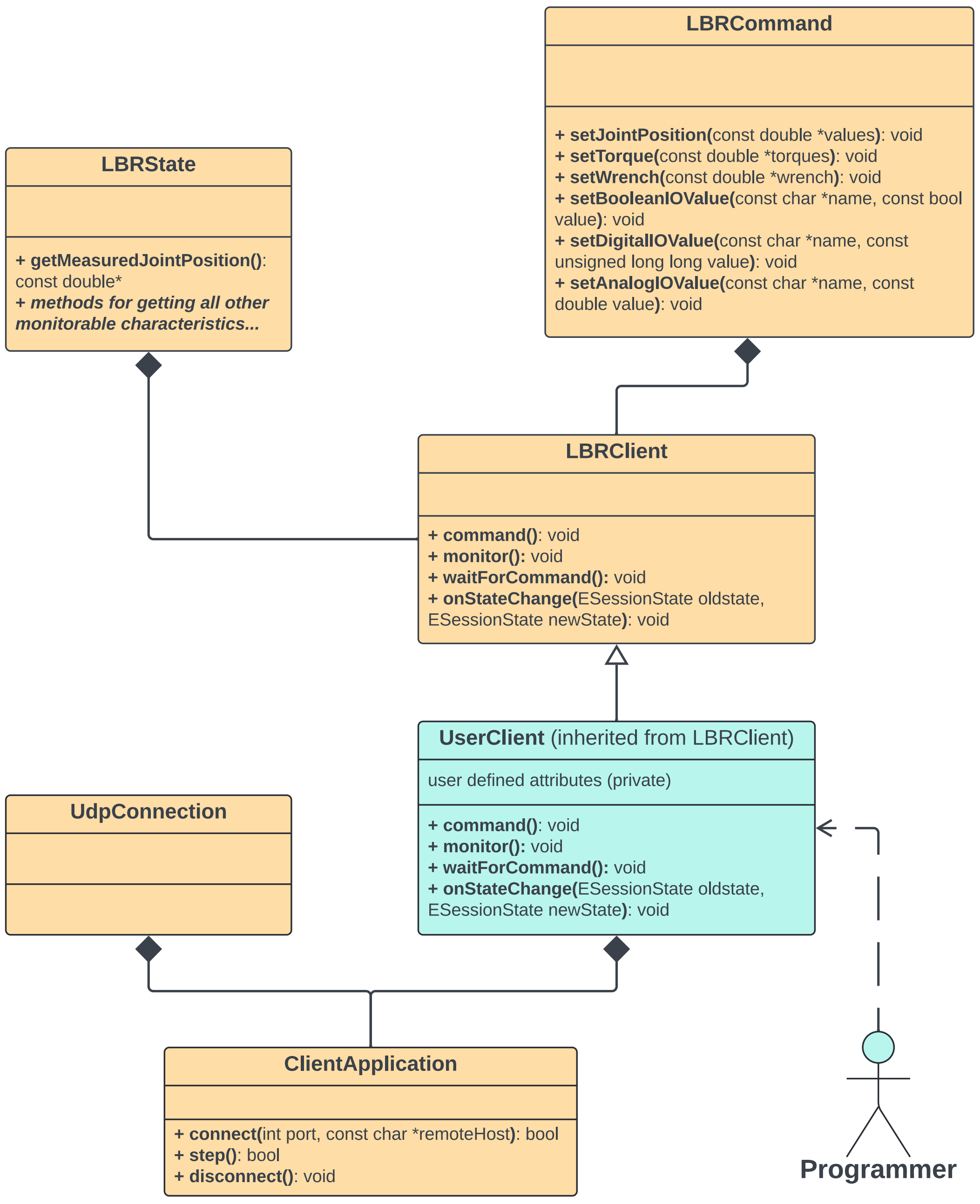

LBRCommand
+ setJointPosition(const double *values): void
+ setTorque(const double *torques): void
+ setWrench(const double *wrench): void
+ setBooleanIOValue(const char *name, const bool value): void
+ setDigitalIOValue(const char *name, const unsigned long long value): void
+ setAnalogIOValue(const char *name, const double value): void
LBRState
+ getMeasuredJointPosition(): const double*
+ methods for getting all other monitorable characteristics...
LBRClient
+ command(): void
+ monitor(): void
+ waitForCommand(): void
+ onStateChange(ESessionState oldstate, ESessionState newState): void
UserClient (inherited from LBRClient)
user defined attributes (private)
+ command(): void
+ monitor(): void
+ waitForCommand(): void
+ onStateChange(ESessionState oldstate, ESessionState newState): void
UdpConnection
ClientApplication
+ connect(int port, const char *remoteHost): bool
+ step(): bool
+ disconnect(): void
Programmer

Figure 3.4: Software structure of the FRI client application in the form of standard UML class diagram. The diagram consists of blocks representing classes from the C++ SDK provided by KUKA (orange) and block representing the user-written class (aqua). Only the most important attributes and methods are depicted. For the complete list of all methods of the `LBRState` class, refer to the table 3.4 and the FRI manual [5]. Figure source: [5] (redrawn and altered).

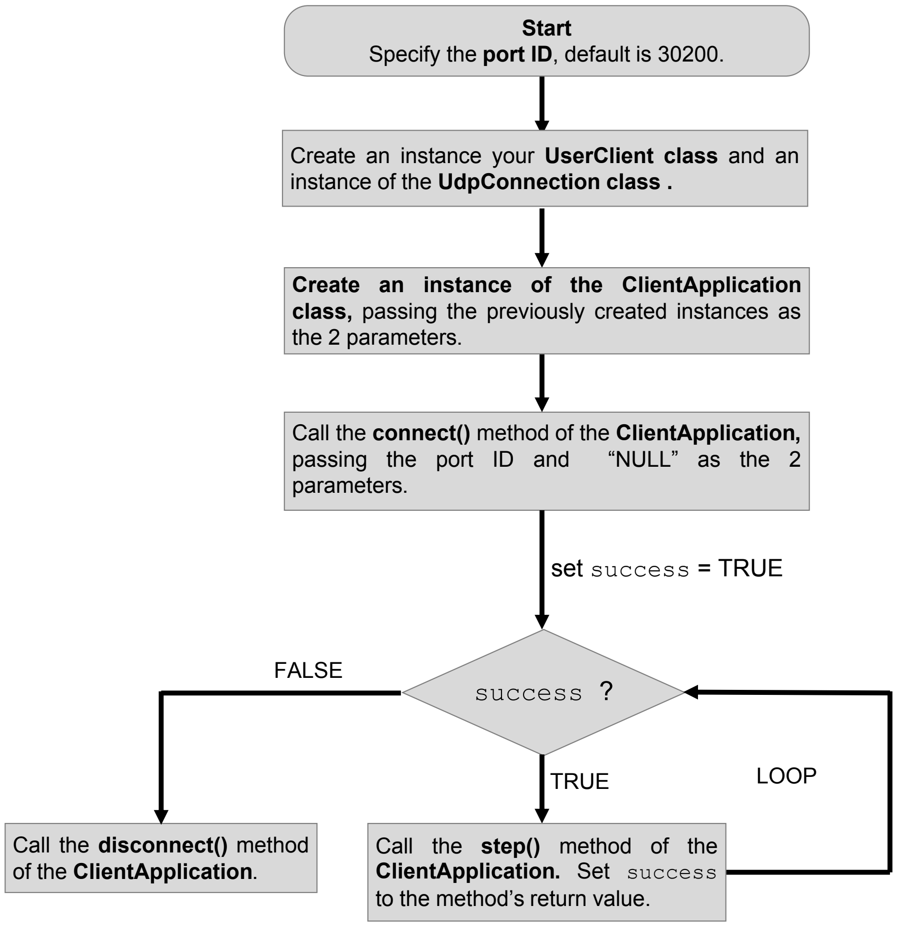

Start
Specify the port ID, default is 30200.
Create an instance your UserClient class and an instance of the UdpConnection class .
Create an instance of the ClientApplication class, passing the previously created instances as the 2 parameters.
Call the connect() method of the ClientApplication, passing the port ID and "NULL" as the 2 parameters.
set success = TRUE
success ?
FALSE
TRUE
LOOP
Call the disconnect() method of the ClientApplication.
Call the step() method of the ClientApplication. Set success to the method's return value.

Figure 3.5: Flowchart illustrating the working principle of the FRI client application. Details of the C++ objects referenced in this flowchart can be found in the Figure 3.4 and in the FRI manual [5].

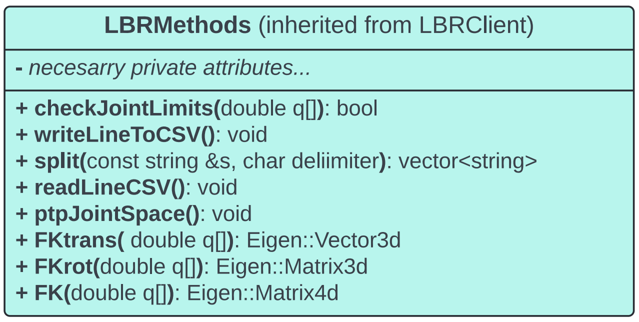

LBRMethods (inherited from LBRClient)
- necesarry private attributes...
+ checkJointLimits(double q[]): bool
+ writeLineToCSV(): void
+ split(const string &s, char deliimiter): vector<string>
+ readLineCSV(): void
+ ptpJointSpace(): void
+ FKtrans(double q[]): Eigen::Vector3d
+ FKrot(double q[]): Eigen::Matrix3d
+ FK(double q[]): Eigen::Matrix4d

Figure 3.6: UML diagram of the user-implemented FRI client class, named `LBRMethods` in this case.

Table 3.5: List of C++ methods useful for further FRI client application development. The methods governing forward kinematics were implemented based on the DH parameters of LBR iiwa, see Table 2.1.

Method header	**Description**
bool checkJointLimits(double q[])	Checks whether the joint configuration q[] violates the joint angle limits. Returns `true` if the configuration is valid (safe).
void writeLineToCSV()	Writes data in the CSV file specified by the `_writeFile` attribue. See method body for which data is being written.
vector<string> split(const string &s, char delimiter)	Splits a string by a delimiter and stores the parts in a vector. Helper method for the `readLineCSV()` method.
void readLineCSV()	Reads a single line from a CSV file specified by `_readFile`, sets `_qCommand` with the data it read.
void ptpJointSpace()	Executes a trajectory in joint space to the configuration `_qGoal` in real time, velocity is specified by the requested trajectory time `_time`.
Eigen::Vector3d FKtrans(double q[])	Returns the 3x1 translational vector specifying the position of end effector in the base frame based on the robot configuration q[].
Eigen::Matrix3d FKrot(double q[])	Returns the 3x3 rotational matrix of the end effector in the base frame based on the robot configuration q[].
Eigen::Matrix4d FK(double q[])	Returns the full 4x4 matrix representing the transformation from the end effector to the base frame based on the robot configuration q[].

3.5 FRI + ROS 2

Robot Operating System (ROS) is a popular open-source set of software frameworks and tools used for robot software development. It implements an architecture of nodes - individual processes that can communicate with each other via standardized messaging system. This standardization of communication is beneficial in the long run, because it simplifies the integration of additional devices, such as sensors and cameras.

Since the release of the LBR iiwa, many open-source projects have been developed with the aim of integrating the manipulator into the ROS or ROS 2 architecture. Some projects lack the integration of the FRI [11], [12] or focus solely on ROS and exclude ROS 2 [13]. For this reason, we have decided to deploy the `lbr_fri_ros2_stack`[5] [10] to our system because it supports both ROS 2 and all client commands modes of the FRI. This framework integrates the UDP-based FRI communication into the ROS 2 build system by creating one publisher node for FRI monitoring and one subscriber node for receiving FRI commands. The framework creates custom ROS 2 message types for this purpose. The complete architecture of the `lbr_fri_ros2_stack` framework is depicted in Figure 3.8.

In order to utilize the ROS 2 topic streams provided by `lbr_fri_ros2_stack`, we developed our own "user" ROS2 node that can both publish and subscribe to corresponding topics. To achieve this, the user node has to recognize the message types that are streamed in the topics. We accomplished this by integrating the `lbr_fri_ros2_stack` package responsible for message type definition into our local ROS 2 workspace. This communication between the user node and the framework is illustrated in Figure 3.7. To test this communication, we have written a user node in Python called `joint_commander` that functions similarly as our `LBRReadWrite` application (see 3.4.3) in that it commmands preplanned joint configurations from a CSV file. The complete local ROS 2 workspace, including the `joint_commander` Pyhton node is attached in Appendix A.

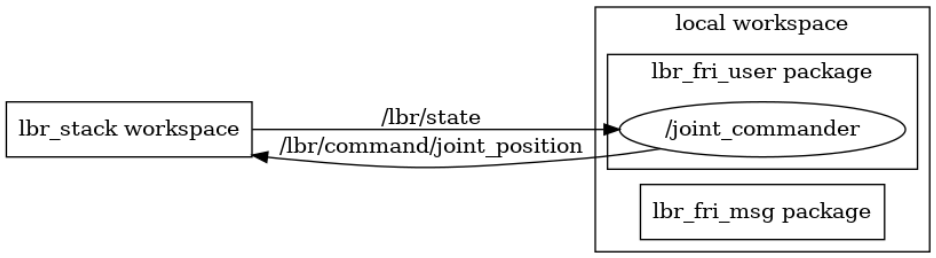
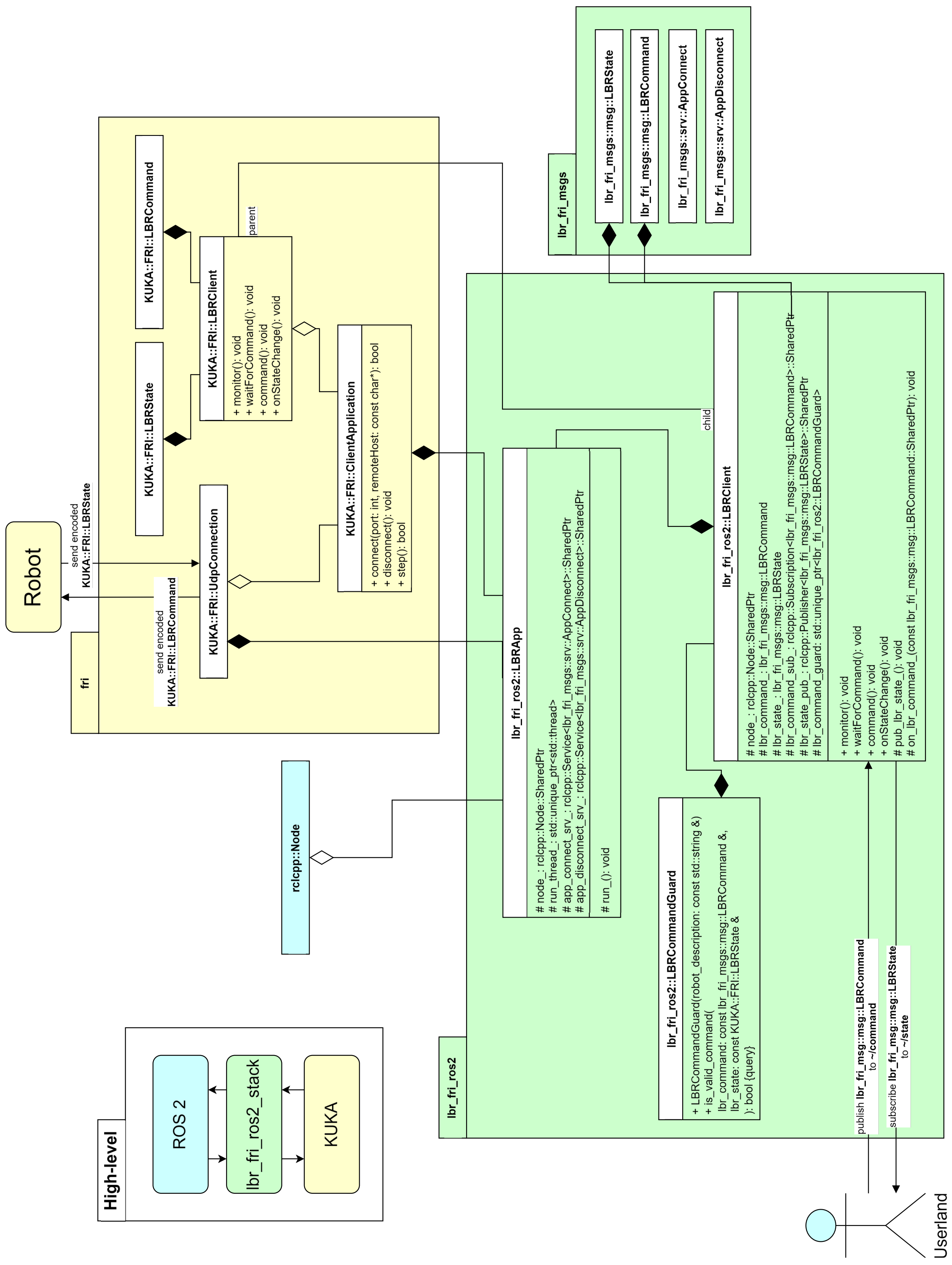


Figure 3.7: ROS 2: communication between `lbr_fri_ros2_stack` framework and the user node in local workspace. The name of the user node is `joint_commander`. This node subscribes to the `/lbr/state` topic and publishes to the `/lbr/command/joint_position` topic.

[5]Link to Github repository: `https://github.com/lbr-stack/lbr_fri_ros2_stack`



Robot
send encoded KUKA::FRI::LBRState
send encoded KUKA::FRI::LBRCommand
fri
KUKA::FRI::LBRCommand
KUKA::FRI::LBRState
KUKA::FRI::LBRClient
+ monitor(): void
+ waitForCommand(): void
+ command(): void
+ onStateChange(): void
parent
KUKA::FRI::UdpConnection
KUKA::FRI::ClientApplication
+ connect(port: int, remoteHost: const char*): bool
+ disconnect(): void
+ step(): bool
lbr_fri_msgs
lbr_fri_msgs::msg::LBRState
lbr_fri_msgs::msg::LBRCommand
lbr_fri_msgs::srv::AppConnect
lbr_fri_msgs::srv::AppDisconnect
lbr_fri_ros2
lbr_fri_ros2::LBRApp
# node_: rclcpp::Node::SharedPtr
# run_thread_: std::unique_ptr<std::thread>
# app_connect_srv_: rclcpp::Service<lbr_fri_msgs::srv::AppConnect>::SharedPtr
# app_disconnect_srv_: rclcpp::Service<lbr_fri_msgs::srv::AppDisconnect>::SharedPtr
# run_(): void
child
lbr_fri_ros2::LBRClient
# node_: rclcpp::Node::SharedPtr
# lbr_command_: lbr_fri_msgs::msg::LBRCommand
# lbr_state_: lbr_fri_msgs::msg::LBRState
# lbr_command_sub_: rclcpp::Subscription<lbr_fri_msgs::msg::LBRCommand>::SharedPtr
# lbr_state_pub_: rclcpp::Publisher<lbr_fri_msgs::msg::LBRState>::SharedPtr
# lbr_command_guard: std::unique_ptr<lbr_fri_ros2::LBRCommandGuard>
+ monitor(): void
+ waitForCommand(): void
+ command(): void
+ onStateChange(): void
# pub_lbr_state_(): void
# on_lbr_command_(const lbr_fri_msgs::msg::LBRCommand::SharedPtr): void
lbr_fri_ros2::LBRCommandGuard
+ LBRCommandGuard(robot_description: const std::string &)
+ is_valid_command(
lbr_command: const lbr_fri_msgs::msg::LBRCommand &,
lbr_state: const KUKA::FRI::LBRState &
): bool {query}
rclcpp::Node
publish lbr_fri_msg::msg::LBRCommand to ~/command
subscribe lbr_fri_msg::msg::LBRState to ~/state
Userland
High-level
ROS 2
lbr_fri_ros2_stack
KUKA

Figure 3.8: Software architecture of the `lbr_fri_ros2_stack` framework. Image source: [10] [Accesed on: 05/03/2024]

3.6 Comparison

After reviewing each software option individually, we are able to properly assess them. The default KUKA Sunrise.Workbench offers a plethora of motion commands, but the environment without additional extensions does not provide any means of real-time control. The KUKA Sunrise Toolbox provides a similar range of motion commands as well as many desirable features for research - such as methods for calculating forward/inverse kinematics/dynamics and dynamic parameters of LBR iiwa. It also provides soft real-time control and the fact that it is a MATLAB add-on makes it the ideal candidate for MATLAB and Simulink integration. The Fast Robot Interface provides even more reliable version of real-time control but lacks the desirable methods offered by KST. Finally, the integration of FRI into the ROS 2 infrastructure provides the comfort of programming robot applications in Python and it could also potentially support MATLAB/Simulink nodes, but more research needs to be done in this area. To complete this analysis, we present a comparison of software capabilities is in Table 3.6.

Table 3.6: Complete comparison of the examined software options for the KUKA LBR iiwa manipulator. Main table (top) is supported by its legend (bottom). "EE" stands for end effector.

	Sunrise Work-bench	KST	FRI	FRI + ROS 2
🟦 PTP in joint space	✔	✔	✔**	✔**
🟦 PTP in EE space	✔	✔	✔**	✔**
🟦 Adjustable trajectory velocity	✔	✔	✔**	✔**
🟦 Adjustable trajectory acceleration	✔*	✘	✔**	✔**
🟦 Adjustable trajectory jerk	✔*	✘	✔**	✔**
🟧 Real-time control (any type)	✘	✔	✔	✔
🟧 Real-time joint space configurations	✘	✔	✔	✔
🟧 Real-time joint space velocities	✘	✔	✘	✘
🟧 Real-time joint space torques	✘	✘	✔	✔
🟧 Real-time EE space configurations	✘	✔	✘	✘
🟧 Real-time wrenches in EE space	✘	✘	✔	✔
⬛ Access to flange I/O	✔	✘	✔	✔

Color/symbol	**Meaning**
🟦	offline functionality
🟧	real-time functionality
⬛	other
✔*	parameter is adjustable, but only positive values are allowed
✔**	functionality is not built in, but can be provided by the suporting Sunrise Workbench appplication

Chapter 4

Demonstration: Crane-like system

4.1 Demonstration overview

One of many practical applications of dynamic control is the control of a load suspended on a rope or a string, which is attached to a moving object (for example an overhead gantry crane or a helicopter). The goal of such control is usually to minimize the magnitude of swings that the load experiences during the motion of the object to which it is attached, while also minimizing the time it takes for the object to move from one location to another. In general, two control strategies for this task exist: open-loop control, in which the reference signal is being shaped based on prior knowledge of system dynamics, or closed-loop control, which actively compensates system vibrations.

In this chapter, we will present a simple demonstration of open-loop control of a crane-like system. During the demonstration, the LBR iiwa's end effector will function as a crane, to which a small load suspended on a string will be attached. The goal of this demo is to:

1. develop an open-loop control strategy that regulates the angle of deviation of the load during the end effector's motion with a secondary goal of planning a time-optimal trajectory; and
2. showcase this control on an example where the robot has to navigate the ball through a layout of obstacles without making contact with them.

4.2 Theoretical background

4.2.1 Dynamic model of the crane-like system

In order to develop control strategy for this task, an appropriate dynamic model has to be designed. Let us assume a planar model (Figure 4.1) consisting of a solid body capable of motion along the horizontal $\hat{x}$ direction with mass M. Another object with mass m (a load) is attached to the horizontally moving body by a string of length l. This model has 2 degrees of freedom and can therefore be described using 2 generalized coordinates x and θ (angle between the string and the vertical). By

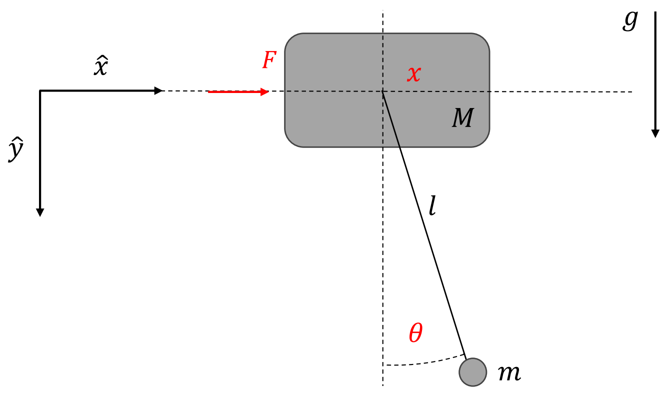


Figure 4.1: Dynamic model of the demonstrator system, a diagram.

solving the Euler-Lagrangian equations of this model we have obtained a system of two second-order nonlinear differential equations,

$$(M+m)\ddot{x} - ml\dot{\theta}^2 \sin\theta + ml\ddot{\theta}\cos\theta = F \ , \tag{4.1}$$

$$l\ddot{\theta} + \ddot{x}\cos\theta + g\sin\theta = 0 \ , \tag{4.2}$$

where F is the driving (input) force acting on the mass M in the $\hat{x}$ direction. This model has also been derived in [14] and similar models are often used to describe inverted pendulums in so-called cart-pendulum systems.

Within the context of this work, the horizontally moving body from the previous model can represent the robot's end effector and the smaller body can represent the load attached to it. However, the representation of the end effector using the mass M is impractical. Let us instead assume we can reliably control the end effector's translational acceleration $\ddot{x}$. We also assume that $m << M$ so that the load's motion has minimum impact on the motion of the end effector. Under these assumptions the model can be simplified into 1 DOF system governed solely by equation (4.2) where $\ddot{x}$ is now considered to be the system's input.

4.2.2 Control strategy

Because of the fact that our model describes only a 2-dimensional crane, we limit this demonstration to trajectories consisting of 90° turns only. This means that the load will only swing along one task space coordinate. It also puts a constraint on the motion of the end effector:

$$\dot{x}(\tau_n) = 0 \ , \tag{4.3}$$

where τ_n is the time of n-th turning point. Let us further examine the impact of end effector acceleration $\ddot{x}$ on the system dynamics. As small angle values of $\theta(t)$ are expected, equation (4.2) may be approximated as:

$$\ddot{\theta} + \omega_n^2\theta = -\frac{\omega^2}{g}\ddot{x} \ , \tag{4.4}$$

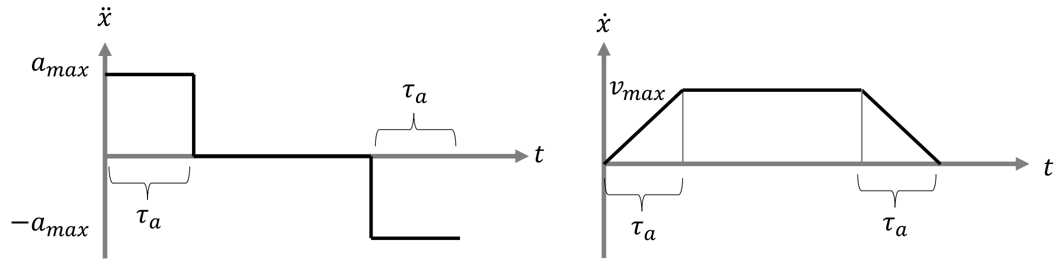


Figure 4.2: Example illustration of acceleration and velocity profiles which are the result of trajectory planning by acceleration shaping. Image source: [15] (redrawn).

where ω_n is the natural frequency of load oscillations simply given by:

$$\omega_n = \sqrt{\frac{g}{l}} \,,$$

where l is the string length and g is the acceleration due to gravity. Equation (4.4) shows that the acceleration $\ddot{x}$ directly influences the magnitude of load swing. Hoang et al. in [15] further argue that if one considers $\ddot{x} = a = \text{const.}$, then the solution to the differential equation (4.4) may be written as

$$\theta(t) = \frac{a}{g} \left(\cos \omega_n t - 1 \right) \ . \tag{4.5}$$

In [15] it is proposed that by setting the acceleration time, τ_a, to a multiple of the load natural period T_n, more specifically

$$\tau_a = kT_n = \frac{2k\pi}{\omega_n} \,, \qquad k = 1,2,\ldots, \tag{4.6}$$

and subsequently substituting τ_a to (4.5),then $\theta(\tau_a) = 0$ and $\dot{\theta}(\tau_a) = 0$, which is exactly what we want to achieve. Hoang et al. in [15] call this approach *trajectory planning by acceleration shaping* and so we will do as well. In general, similar control strategies are usually found under labels such as *anti-sway control* or *time-optimal anti-sway control* [16]. The acceleration and velocity profiles that result from setting the acceleration time to τ_a are illustarted in Figure 4.2.

We have developed a simulation of the dynamic system in MATLAB for testing various shapes of the input acceleration. Some examples of end effector accelerations and the respective time evolutions of the load angle $\theta(t)$ in a system with string length l = 25 cm are presented in Figure 4.3. The simulations have shown that even the basic acceleration shaping can minimize the magnitude of oscillations to approximately 0.1 radians ($\approx$ 5.7 °). Although the input signals with more than one step can reduce the oscillations even further, they require more time to execute, as each individual step has a duration T/n where n signifies n-step acceleration shaping. For this reason, we have decided to use the basic acceleration shaping for the demonstration.

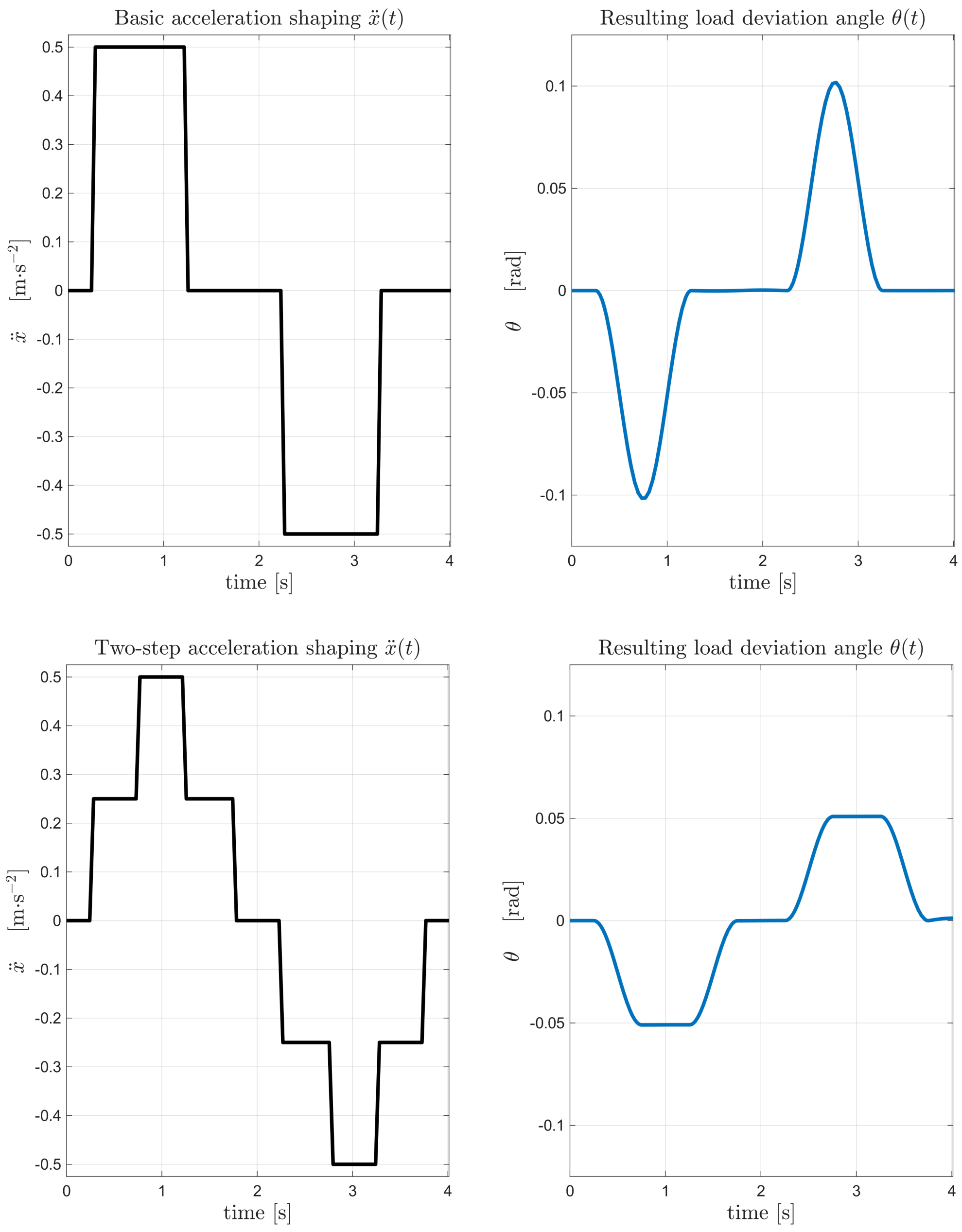


Figure 4.3: Simulations of the crane-like system with specific input signals. Basic acceleration shaping (top row), two-step acceleration shaping (bottom row). Simulations were performed on a system with string length $l = 25$ cm and period of oscillation $T \approx 1$ s.

4.3 Implementation

The entirety of this demonstration was developed in MATLAB. Our implementation can be divided into two parts. First a script regarding the offline trajectory planning in task space was developed in basic MATLAB. After that, a client app responsible for sending the preplanned trajectory to LBR iiwa in real-time was written using the tools provided by KUKA Sunrise Toolbox (see section 3.3).

In the script `Points2TrajectorySym.m` (available in Appendix A) we have developed a simple algorithm that builds a trajectory (meaning position data + time stamps) of the end effector in task space. The user has to provide horizontal coordinates of the turning points (i.e. the places where the end effector is supposed turn) and maximum permissible task space acceleration. With this user input the script constructs the trajectory incorporating the acceleration shaping control. This algorithm is described in detailed pseudocode in Figure 4.4. To reiterate, the trajectory is constructed in such a way that during its execution, the end effector will experience a horizontal acceleration $\ddot{x}$ which in the context of our dynamic model represents the model's input.

Figure 4.4 Task space trajectory algorithm (basic acceleration shaping)

```
 1: $x_0, y_0 \leftarrow$ coordinates of the start
 2: $xy \leftarrow$ Nx2 array of coordinates of N turning points
 3: $h \leftarrow$ time step
 4: $a_{max} \leftarrow$ maximum permissible task space acceleration
 5: $T \leftarrow$ natural period of load oscillation
 6:
 7: procedure PlanTrajectory($xy$, $x_0$, $y_0$, $h$, $a_{max}$, $T$)
 8:     $trajectory \leftarrow$ prepare a 2-dimensional array
 9:     $segments \leftarrow$ calculate distances between turning points in $xy$
10:     $s_{min} \leftarrow$ compute the shortest distance for acceleration shaping using $\frac{1}{2}a_{max}T^2$
11:     $v_{max} \leftarrow$ calculate maximum velocity, given by $a_{max}T$
12:
13:     for $s$ in $segments$ do
14:         if $s >= 2s_{min}$ then
15:             $t_{max} \leftarrow$ time of maximum velocity, given by $(s - 2s_{min}) / (v_{max})$
16:             $a \leftarrow a_{max}$
17:         else
18:             $t_{max} \leftarrow 0$
19:             $a \leftarrow$ acceleration given by $s/T^2$
20:         end if
21:         $t \leftarrow$ symbolic variable representing time
22:         $a_{sym} \leftarrow a\mathbb{1}(t) - a\mathbb{1}(t-T) - a\mathbb{1}(t-T-t_{max}) + a\mathbb{1}(t-2T-t_{max})$
23:         $s_{sym} \leftarrow$ integrate $a_{sym}$ twice with respect to $t$
24:         $s_{shaped} \leftarrow$ convert $s_{sym}$ to a standard (nonsymbolic) array
25:         append $s_{shaped}$ to the corresponding place in $trajectory$
26:     end for
27:
28:     $time \leftarrow$ array from 0 with step $h$ to value $h \cdot (length\ of\ trajectory)$
29:     return $trajectory, time$
30: end procedure
```

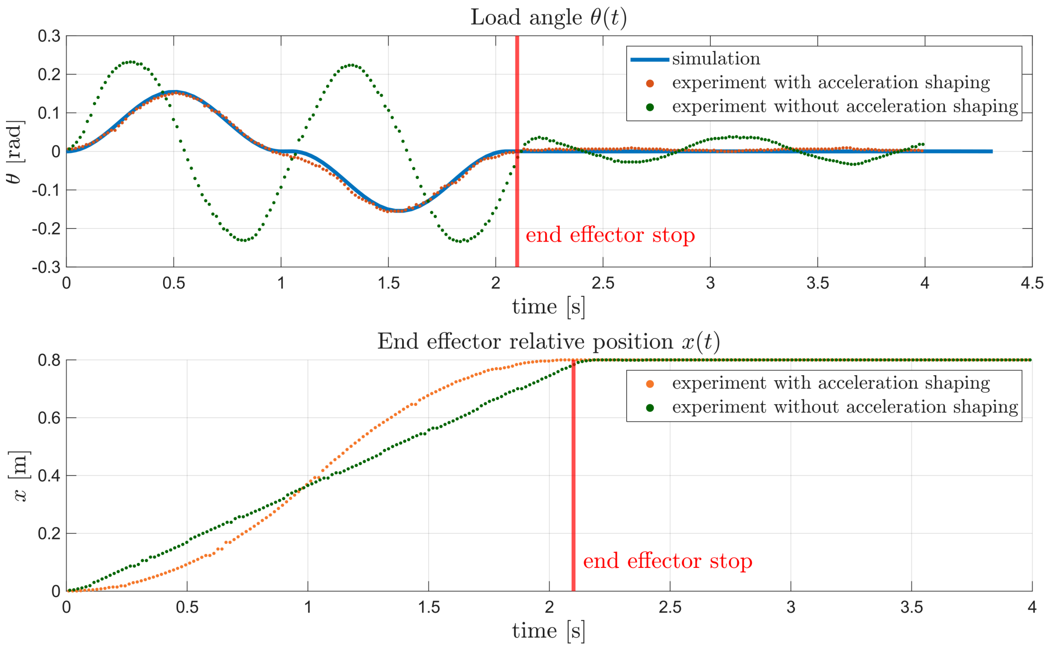


Figure 4.5: Comparison of load angle oscillation under shaped and unshaped commands. Data from the run without acceleration shaping is depicted dark green, data from the run with acceleration shaping is depicted orange. The predicted angle from the simulation is depicted in blue. String length in the experiment was 25 cm.

One of the most useful features of the KST is its capability to command task space end effector positions in real-time. Thanks to this, there was no need to manually convert task space trajectory into joint space trajectory. Our script `KST_IS_realtime.m` simply reads the end-effector's starting height and orientation, starts the TCP/IP connection and then it keeps commanding the task space trajectory data combined with the constant z-coordinate and orientation of the end effector.

4.4 Experiments and the demonstration

The validity of the designed open-loop control strategy was initially tested through a simple experiment. In this experiment, the end effector, with a wooden ball attached by a string, was commanded to move in a straight line - first without any open-loop control and then with opne-control via the acceleration shaping method. The video of this experiment is accessible through this link[1] and it also included in Appendix A. A comparison of data from the experiment with the simulation data is presented in Figure 4.5. In the figure, the effect of the acceleration shaping is clearly visible - the load oscillations during the end effector's motion are reduced and there is no observable motion of the load after the end effector has finished moving.

[1]Video from the open-loop control validation experiment: https://youtu.be/3J86vZgC-YY

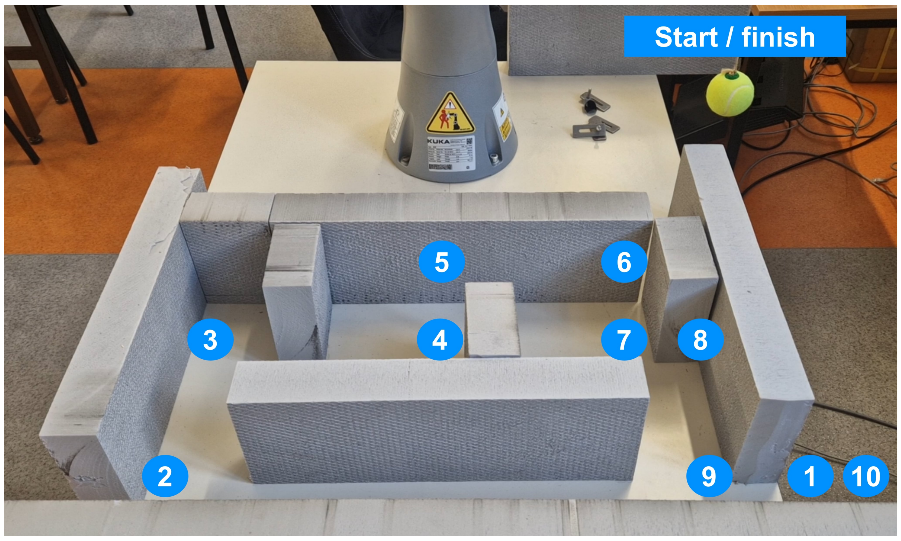


Figure 4.6: Setup for the demonstration of a crane-like system passing through a layout of obstacles. The points at which the end effector was supposed to change direction are depicted in order.

The final product of these efforts was a demonstration of the crane-like system. We have cut blocks of various lengths from extruded polystyrene (XPS) to serve as obstacles. We laid the blocks on the table next to the LBR iiwa, arranging them to form a path for the end effector with the load to pass through. Picture of the obstacle layout can be seen in Figure 4.6. The figure also marks the end effector turning points in order in which they were supposed to be reached. To perform the demonstration, coordinates of the turning points were supplied to the MATLAB scripts described in section 4.3. The video of the demonstration is accessible through this link[2] and is also included in Appendix A.

[2]Video of the crane-like system demonstration:https://youtu.be/N0izJOyTtNI

Chapter 5

Demonstration: Cup-and-ball game

5.1 Demonstration overview

The Cup-and-ball game consists of a cup to which a small ball is attached by a string. The goal of the game is to land the ball in the cup only by moving the cup itself, or in the case of the popular *Kendama*[1] toy, by moving the handle attached to the cup. This is a relatively popular task in the field of robotics and demonstrations of successful catches have been presented in works such as [17] and [18]. However, those works utilized various reinforcement learning methods. A successful demonstration leveraging traditional control-based methods has been documented in the thesis by M. Bujarbaruah [19] where the demonstration has been performed using the Ur5e manipulator. We believe a similar demonstration can be achieved with LBR iiwa.

This task can be divided into two phases. First is the *swing-up* phase, the goal of which is to drive the ball to the upper half of the y-plane, preferably directly above the cup. The goal of the second phase, the *catch* phase, is to catch the free-falling ball. This chapter documents our efforts to demonstrate the *swing-up* phase.

5.2 Swing-up control via energy shaping

We hypothesise that is it possible utilize the model from section 4.2.1 to sufficiently describe the dynamics of the Cup-and-ball game in 2 dimensions under the condition that the string is tense at all times. If this assumption holds true, then a control strategy, called energy shaping, can be deployed. Energy shaping is a method for driving a dynamical system, even nonlinear one, to a desired total system energy E^d.

The second differential equation of our model in 4.2.1 can be rewritten as

$$\ddot{\theta} = -\frac{\ddot{x}}{l}\cos\theta - \frac{g}{l}\sin\theta \, . \tag{5.1}$$

[1]More information on *Kendama*: `https://en.wikipedia.org/wiki/Kendama`.

We note $u \equiv \ddot{x}$ as the input to our system. The total mechanical energy of a simple pendulum is

$$E = \frac{1}{2} m l^2 \dot{\theta}^2 - m g l \cos\theta \; . \tag{5.2}$$

However, equation (5.2) can also be used in the context our cart-pendulum system, if we consider the pendulum to be *decoupled*. The validity of this simplification is supported by the methodology for energy shaping controller design described in [20]. We can then investigate the change in energy

$$\dot{E} = m l^2 \dot{\theta} \ddot{\theta} - m g l \cos\theta, \tag{5.3}$$

and substitute $\ddot{\theta}$ from equation (5.1), obtaining

$$\dot{E} = -u m l \cos\theta \; . \tag{5.4}$$

The approach in [14] suggests to design a controller of the form

$$u = k_E \dot{\theta} \cos\theta \tilde{E}, \qquad k_E > 0 \; , \tag{5.5}$$

where $\tilde{E} = E - E^d$ because then the controller (5.5) makes the energy difference $\left|\tilde{E}\right|$ non-increasing. Although the controller (5.5) only regulates the system energy (and therefore the angle θ), the terms for controlling the position x and the velocity $\dot{x}$ can easily be added:

$$u = k_E \dot{\theta} \cos\theta \tilde{E} - k_p x - k_d \dot{x} \; . \tag{5.6}$$

The variables k_E, k_p, k_d are design parameters that we can tune.

5.3 Implementation

In theory, if the dynamic model of the Cup-and-ball system is trustworthy enough, it should be possible to simulate the behaviour of the system with the controller (5.6) offline, and thus compute the acceleration u in advance. For the purposes of demonstration using the LBR iiwa manipulator, the preplanned acceleration u can subsequently be transformed into the desired motion of the end effector.

5.3.1 Simulation of the dynamics

As we have illustarted in Figure 5.1, the first step in our implementation involved running the simulation of the Cup-and-ball system from which the desired end effector acceleration $\ddot{x}(t)$ could be obtained as the acceleration itself is the input to the Cup-and-ball system. To be more exact, we have used MATLAB's `ode45` numerical solver to integrate the equation 5.1. The numerical integration yielded the ball angle $\theta(t)$ and the relative position of the end effector $x(t)$. Example of the simulation with controller parameters k_E=1.7, k_p=k_d=0.05 is provided in Figure 5.2. The MATLAB file with simulation can be found in Appendix A as `CAB_main.m`.

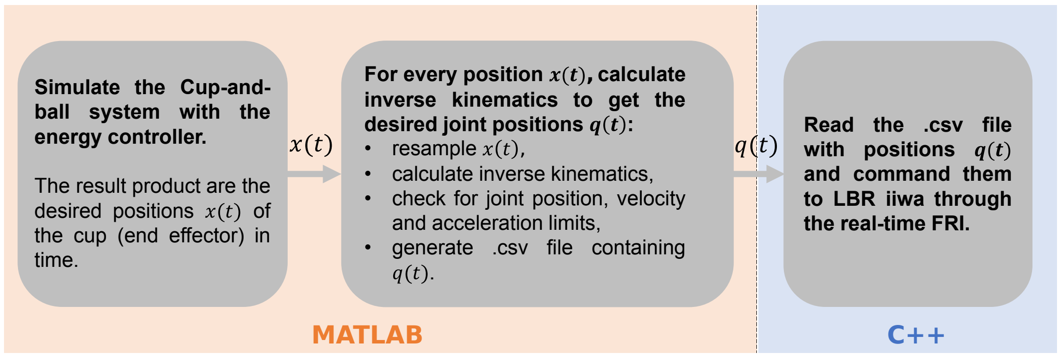

Simulate the Cup-and-ball system with the energy controller.
The result product are the desired positions $x(t)$ of the cup (end effector) in time.
$x(t)$
For every position $x(t)$, calculate inverse kinematics to get the desired joint positions $q(t)$:
• resample $x(t)$,
• calculate inverse kinematics,
• check for joint position, velocity and acceleration limits,
• generate .csv file containing $q(t)$.
$q(t)$
Read the .csv file with positions $q(t)$ and command them to LBR iiwa through the real-time FRI.
MATLAB
C++

Figure 5.1: Visualisation of the software implementation of the swing-up phase in the Cup-and-ball game demostration.

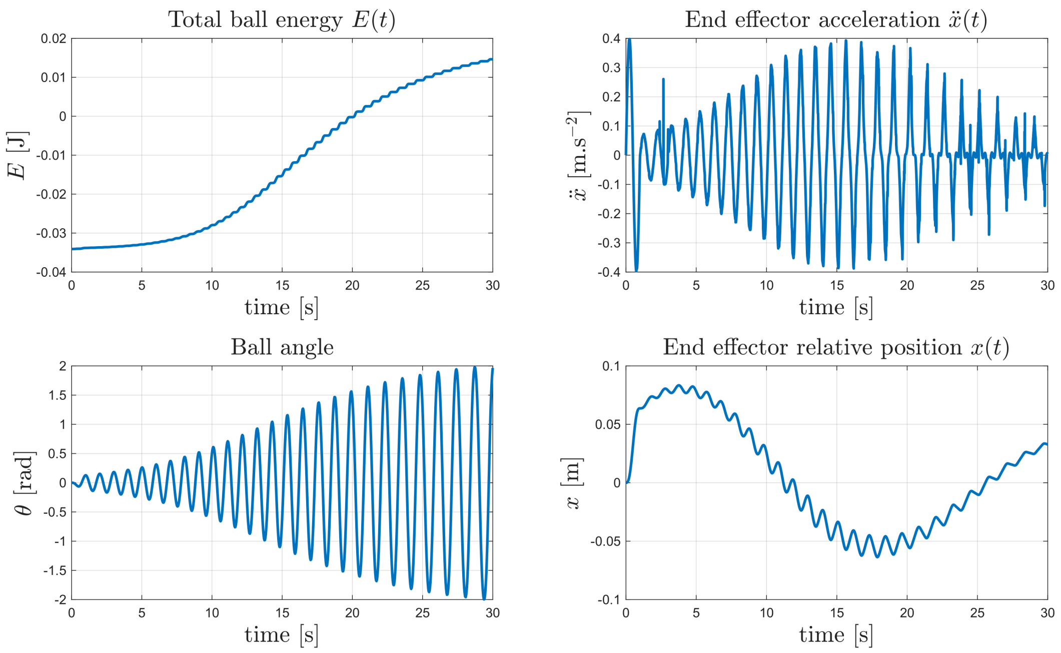


Figure 5.2: Simulation of the Cup-and-ball system with the integrated energy shaping controller. The desired ball angle was set to $\theta^d = 115° \approx 2$ rad. The string length was $l = 0.25$ m, and ball mass $m = 0.0139$ kg. The controller parameters were k_E=1.7, k_p=k_d=0.05.

5.3.2 Postprocessing in MATLAB

As this task requires precise timing, we have decided to utilize the Fast Robot Interface (section 3.4) to command the desired relative positions $x(t)$ of the end effector such that the end effector provides the needed acceleration input $\ddot{x}(t)$ to the Cup-and-ball system. The positions $x(t)$ were sampled unevenly, based on how the `ode45` integrated them. To resolve this, we have created a MATLAB function that resamples the $x(t)$ array to the needed FRI sampling value (5 ms for instance). Because of the fact that the FRI client library (section 3.4.3) does not natively provide any means of calculating forward or inverse kinematics, the calculation had to be done in advance. We have decided to use the joint configuration

$$q_{\text{start}} = [0,\, 0.071,\, 0,\, -1.3523,\, 0,\, 0.1301,\, 0]\ \text{rad}\ ,$$

as the starting position for the robot motion because this configuration sets the end effector's y-coordinate in base frame to y=0 and rotates the end effector's reference frame with respect to the base frame by 90°. More specifically, the transformation of q_{start} from the end effector to the base frame becomes

$$T_{\text{start}} = \begin{bmatrix} 0 & 0 & 1 & 0.585 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0.801 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Next we have generated a list od desired transformations by copying T_{start} and substituting the translational y-coordinate in each of them by a position from $x(t)$. Finally, we have calculated the inverse kinematic problem for each transformation using the method `gen_InverseKinematics()` provided by the KUKA Sunrise Toolbox (section 3.3) and obtained the robot trajectory in joint space $q(t)$.

Additionally, before exporting the trajectory, we have always numerically differentiated the planned trajectory $q(t)$ to receive an estimate $\hat{\dot{q}}(t)$ of joint velocities. We have used the symmetric approximation of the derivation in the form

$$\hat{\dot{q}}_k = \frac{q_{k+1} - q_{k-1}}{2h} , \tag{5.7}$$

where h is the FRI sampling step. Then we have repeated this process again, receiving an estimate $\hat{\ddot{q}}(t)$ of joint accelerations. We have done so in order to check for a potential violation of velocity and acceleration limits. In Figure 5.3, we have plotted the estimated joint velocities and accelerations that resulted from the processing of positions $x(t)$ from Figure 5.2.

5.3.3 Trajectory execution

The last step of this implementation involved exporting the desired joint space trajectory $q(t)$ to a CSV file that the FRI-enabled C++ client aplication could read and execute. We have utilized our `LBRReadWrite` application which was described in detail in section 3.4.3. The C++ application and the MATLAB scripts for simulation and postprocessing can be found in the corresponding folders of Appendix A.

5.4 Experiments on LBR iiwa

In order to perform experiments involving swing-up of the ball, we have designed and 3D printed a component resembling a cup that could be attached to the end effector of LBR iiwa by up to four M6 screws. This component can be seen in Figure 5.4. At the bottom of the 3D printed component, there was a small hole through which a

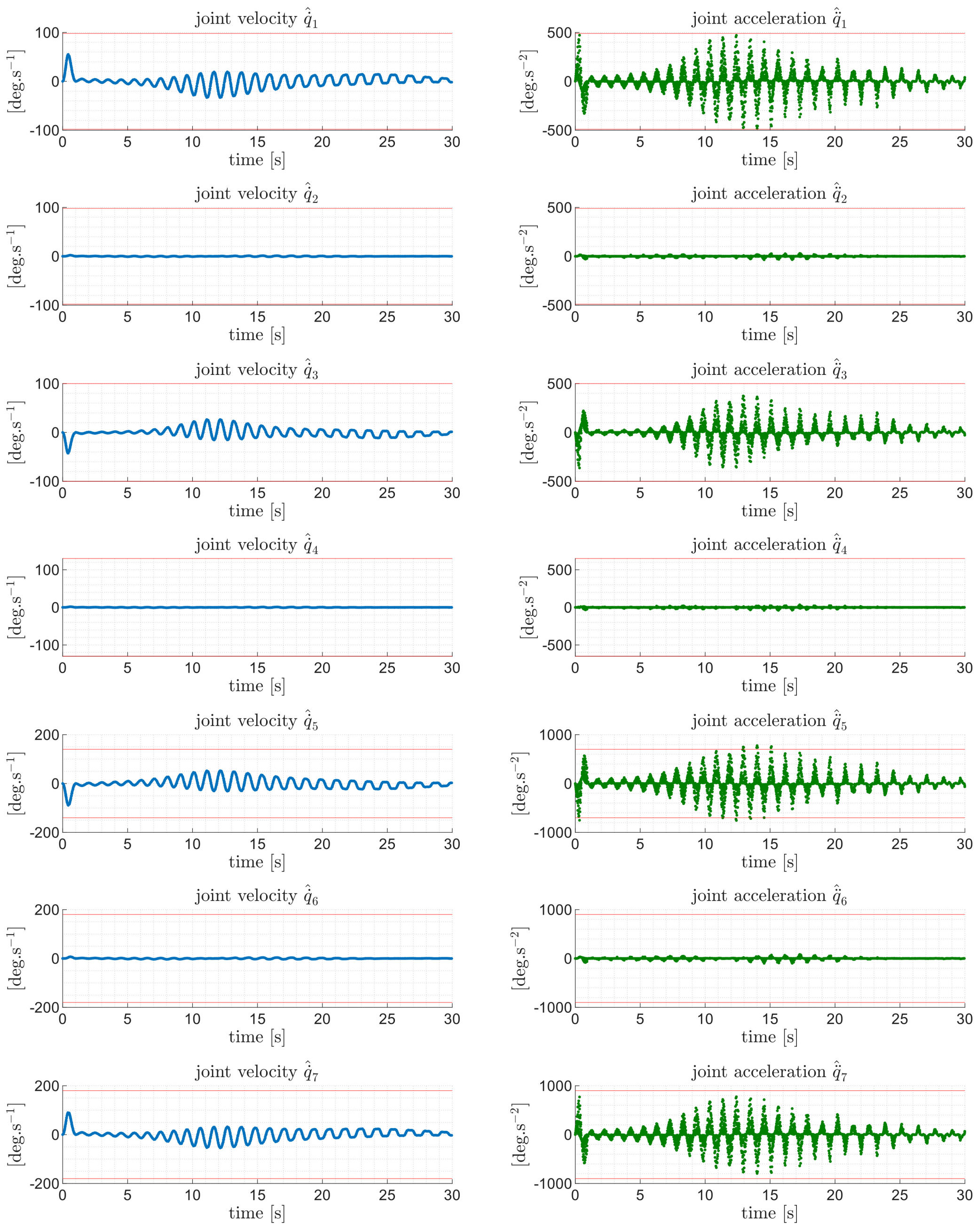


Figure 5.3: Example of joint velocity (blue) and joint acceleration (green) estimates. Velocity and acceleration limits (red) are depicted using data from Tables 2.2 and 2.3.

string was tied. For the purposes of comparing experiments with precomputed simulations, we recorded the experiments using the USB camera described in section 2.4. In the experiments, we purposely used a red ball to ensure easy position tracking. The constructed setup for Cup-and-ball demonstration can be seen in Figure 5.5. Results of experiments are presented in Figure 5.6. Experiments were performed with 2 different string lengths (15 and 25 cm) and with different parameters of the

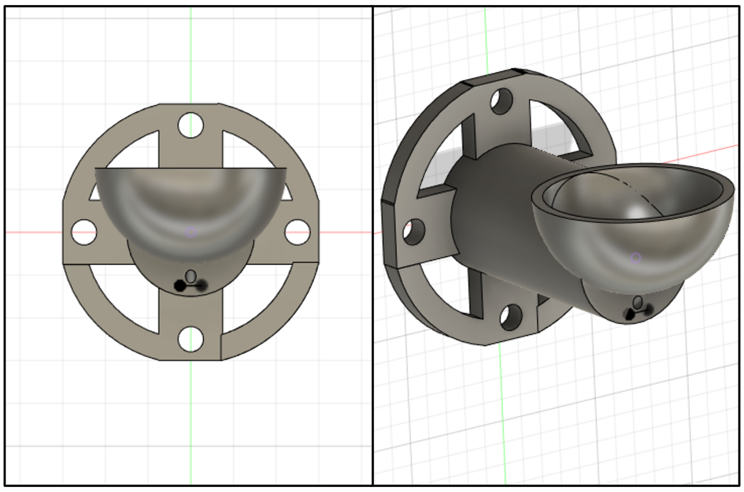


Figure 5.4: 3D printed component to which the ball for the Cup-and-ball demonstration was attached. The component was mounted to the end effector of LBR iiwa using screw holes in its back. Front view (left image) and side view (right image).

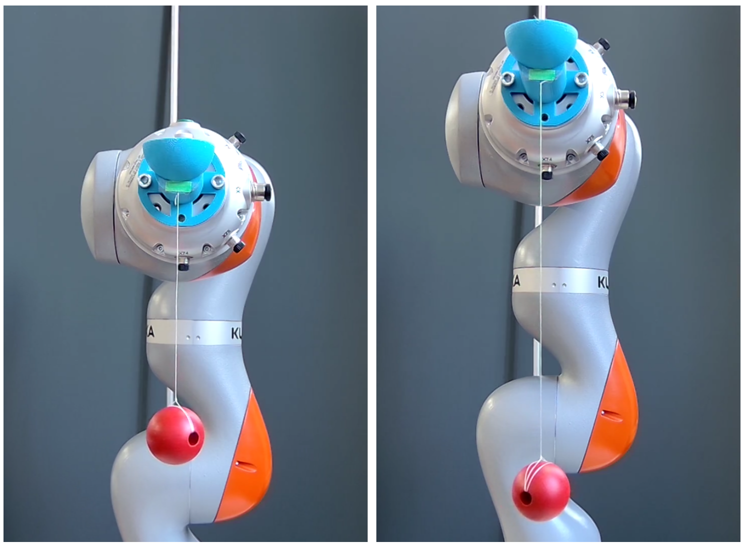


Figure 5.5: LBR iiwa equipped with components for the Cup-and-ball demonstration. Two string lengths are depicted, $l = 15$ cm (left) and $l = 25$ cm (right).

energy shaping controller. All results in 5.6 show that the behaviour of the real system fits the predicted model perfectly up until the ball reaches some critical angle $\theta \approx \frac{\pi}{2}$ rad at which point our model fails. In some experiments, we tried to change the controller parameters during the motion, because we hypothesised that, for instance, doubling the controller parameter k_E at the right moment would result in a greater swing. In Figure 5.6 we labeled those experiments as experiments with nonconstant parameters. However, the impact of this is not significant. The largest angle we were able to achieve with this strategy was $\theta = 1.73$ rad.

We attribute the discrepancy between our model an experiments to the fact that our model assumed the string to be fully stretched at all times. As we observed this is not true, because the string can loose its tension during the motion. In the following section we propose a solution to this problem.

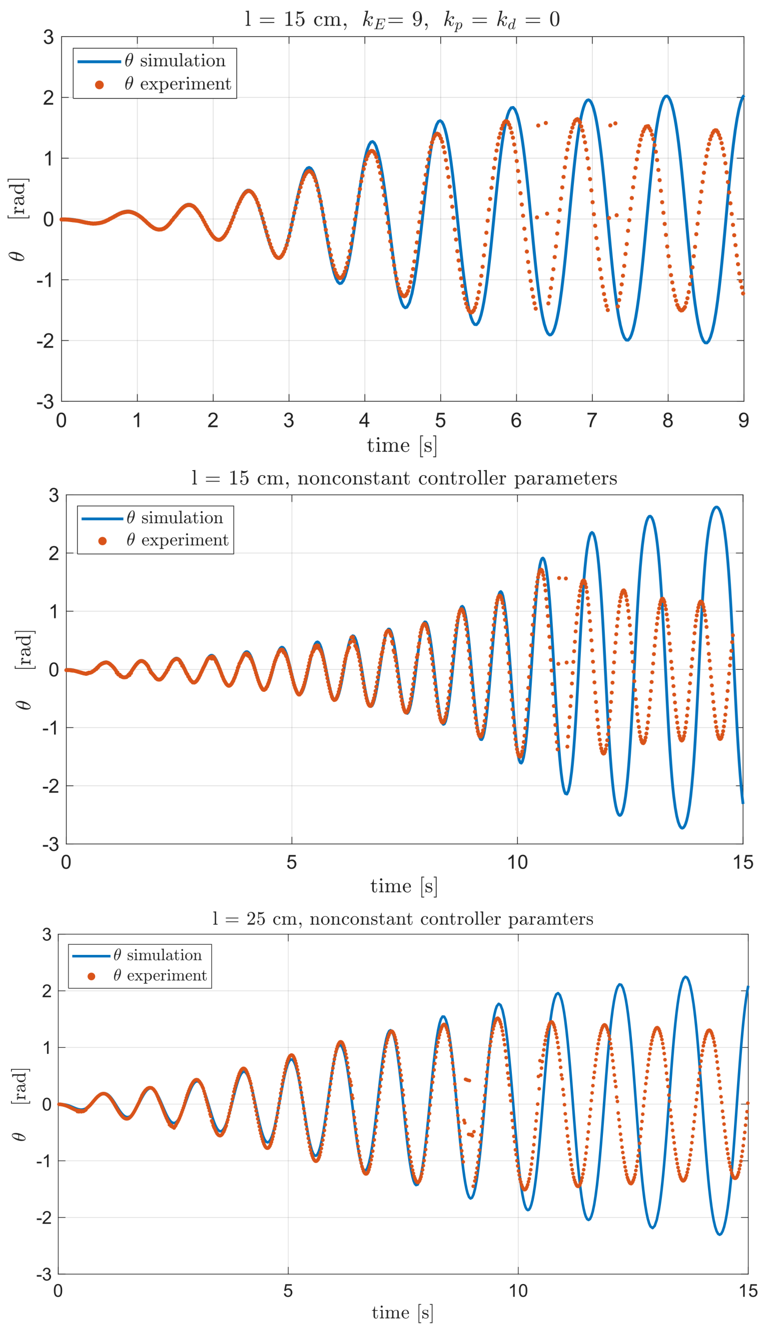


Figure 5.6: Comparison of 3 experiments to their respective simulations. Each graph illustrates the predicted (simulated) angle θ in time (blue) to angle θ measured in the experiment (orange). Graphs list the parameters of experiments in their titles.

5.5 Tension condition

To overcome the difficulties that arised during our experiments with swing-up described in section 5.4, we analysed the forces acting on the ball with the goal of obtaining a rigorous condition describing whether tension is present in the string. To be as general as possible, we expand our model from section 4.2.1 by allowing the input force to have 2 components, F_x and F_y. Other relevant forces include the centrifugal force F_c and the radial component of the gravitational force F_g acting on the ball. These force are depicted in Figure 5.7. We propose that if the forces

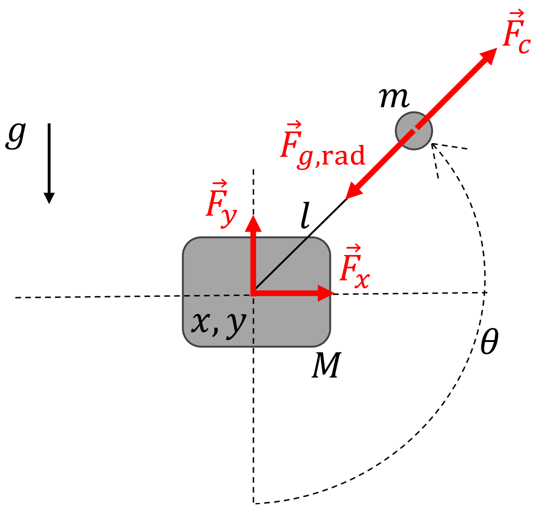


Figure 5.7: Forces acting on the ball in the Cup-and-ball system. Diagram also shows the direction of increasing θ angle.

satisfy the condition

$$F_c + F_g \cos\theta - F_x \sin\theta + F_y \cos\theta > 0 \ , \tag{5.8}$$

then the string experiences tension. The condition can be rewritten in terms of the system variables and parameters as

$$ml\dot{\theta}^2 + mg\cos\theta - M\ddot{x}\sin\theta + M\ddot{y}\cos\theta > 0 \ . \tag{5.9}$$

We believe that this condition is useful because it can be added to the numerical simulation from section 5.3.1 and serve as a stopping condition. Even more significantly, it can also be utilized in other methods for trajectory planning because the relation (5.9) can be viewed as an inequality constraint of an optimization problem, such as the one described in [19].

Chapter 6

Development of feedback-capable pendulum module

A traditional benchmark of dynamic control is the stabilization of pendulum in its upright position. This chapter covers the process of designing, prototyping and programming a planar pendulum module which is supposed to be mounted on the end effector of LBR iiwa.

6.1 Design of mechanical components

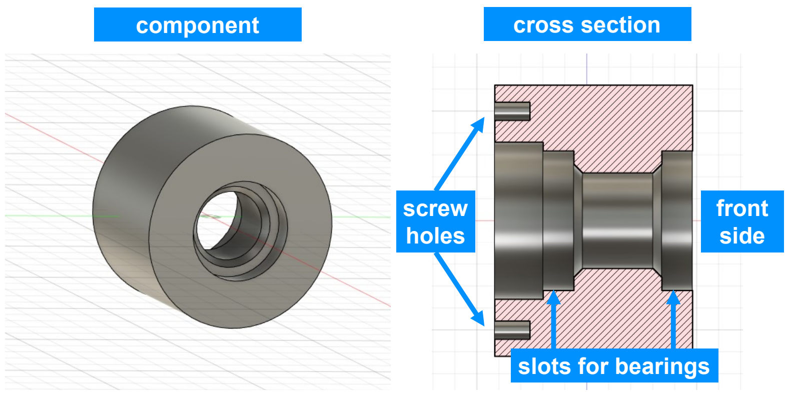
There were a few design criteria that we strived to follow. We wanted the pendulum module:

- to be compact,
- to limit the pendulum motion to a single plane,
- to support easy exchange of pendulum rods.

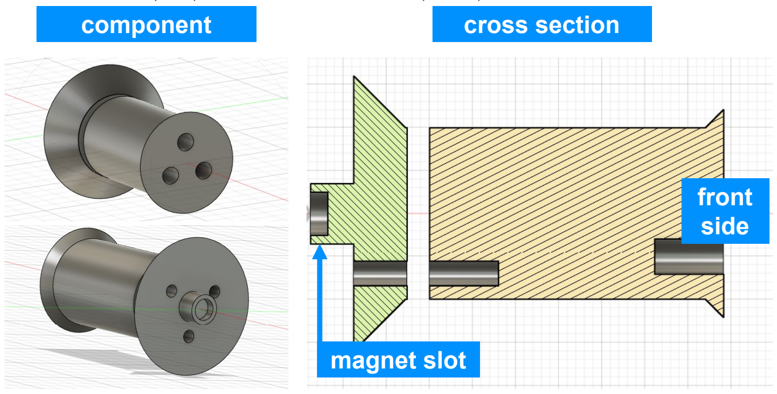
The module consisted of two main parts, which together formed a cylindrical body. The first (front) part of the module, displayed in Figure 6.1, was designed to hold 2 metal bearings coaxially along the cylinder's central axis, 20 mm apart. Inside these bearings, the shaft of the pendulum was inserted. The front of the shaft featured three screw holes (M4) for connecting the pendulum rod, while the back of the shaft featured a cavity for a disc magnet. This magnet is a component of the hall effect sensor of the rotary encoder described in section 6.2. The second (back) part of the module was also cylindrically shaped, see Figure 6.2. Its purpose was to house the rotary encoder on one side and provide screw holes (M6) for connecting the whole pendulum module to the end effector. Furthermore, this part featured a small platform on its top for additional electronics. These components as well as pendulum rods of various sizes have been 3D printed. Additionally, one 75 cm long pendulum rod made out of aluminium was also constructed, see Figure 6.3.

6.2 Electronics

In order to supply power for external electronics, internal VCC and GND cable connections of the X3 port on the Touch Electric media flange were utilized. However,



(a) : Outer housing for the bearings and the shaft. View of the component (left), cross section view (right).



(b) : The shaft of the pendulum. The pendulum rod connects to the front side and the shaft itself slides into the housing. View of the shaft (left), cross section view (right).

Figure 6.1: Front part of the pendulum module (CAD model). Figure (a) depicts the outer component which houses the bearings and shaft. Figure (b) depicts the shaft.

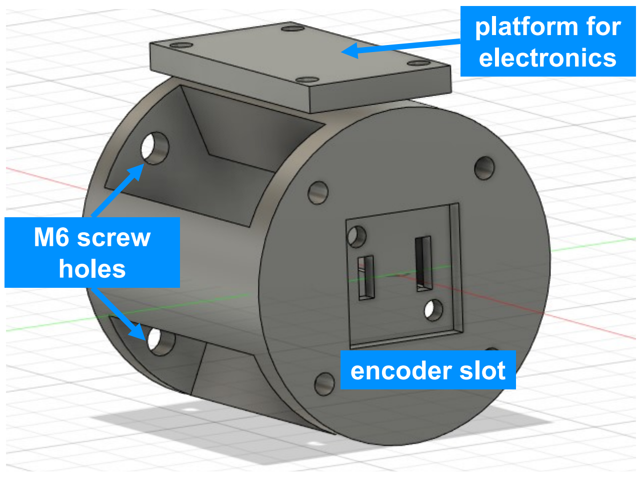


Figure 6.2: Back part of the pendulum module (CAD model). This component is the middle part between the end effector and the module front part displayed in Figure 6.1.

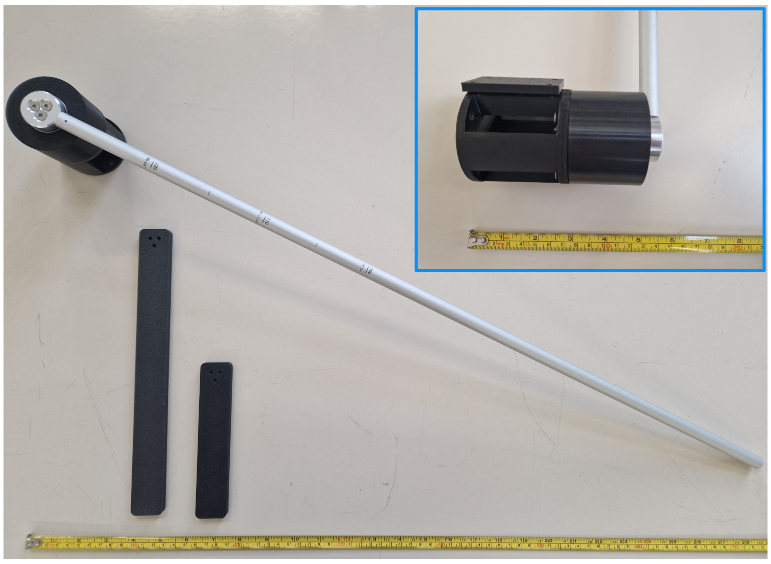


Figure 6.3: 3D printed pendulum module with multiple pendulum rods.

because the LBR iiwa internally runs on 24 V logic, the VCC = 24 V was too high and a DC-DC step-down converter module had to be used. The voltage was stepped down to 5 V.

To provide real-time feedback on the pendulum angular position, we installed the AS5600 contactless magnetic rotary encoder inside the pendulum module. This encoder measures absolute angle, has 12 bit resolution, sampling rate of 150 μs[1] and communicates over the standard I^2C interface. The encoder was connected to an Arduino Nano, which supplied it with the necessary 3.3 V input voltage and further processed the I^2C data. The Arduino itself was powered from the pins of the previously mentioned step-down converter.

The communication between the Arduino Nano on the pendulum module and the PC with a robot application was facilitated by a pair of NRF24L01 wireless radio transmitter-receiver modules. This type of wireless communication was preferred over Bluetooth or Wi-fi, because it is supposed to achieve the lowest latency [21]. The NRF24L01 transmitter was connected to the Arduino on the pendulum module and the NRF24L01 receiver was wired to a stationary Arduino connected to the PC via USB cable. We illustrate the schematics of the electronic circuitry in Figure 6.4 and the physical prototype can be seen in Figure 6.5. The Arduino on the pendulum module was programmed to continuously read the encoder data and send them over the radio. More specifically, the `AS5600.h` library was used to read the 12-bit encoder value x, which we converted to angle θ in radians using the relation

$$\theta = \frac{2\pi}{2^{12}} \cdot x \quad . \tag{6.1}$$

[1]Datasheet of the encoder is accessible here: `https://www.laskakit.cz/user/related_files/as5600_ds000365_5-00-1877365.pdf` [Accessed: 2024-05-19]

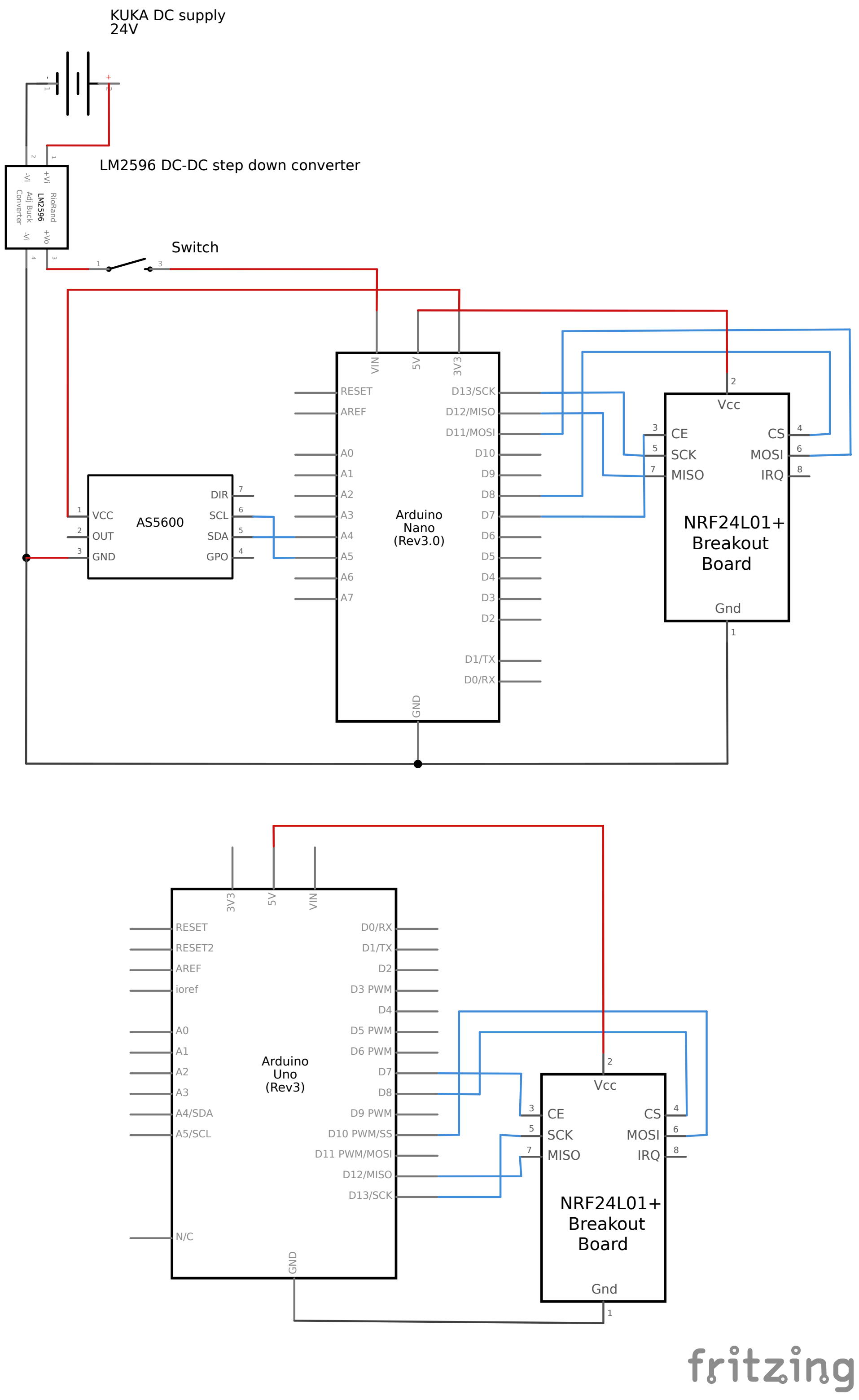


Figure 6.4: Schematics of the pendulum module electronics (top) and the radio receiver (bottom).

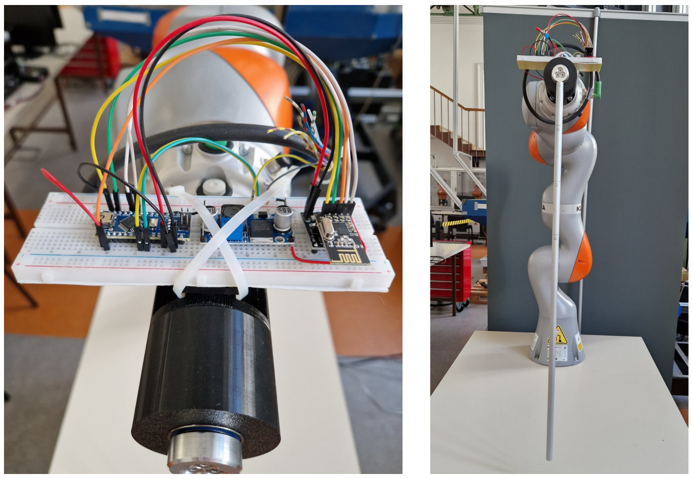
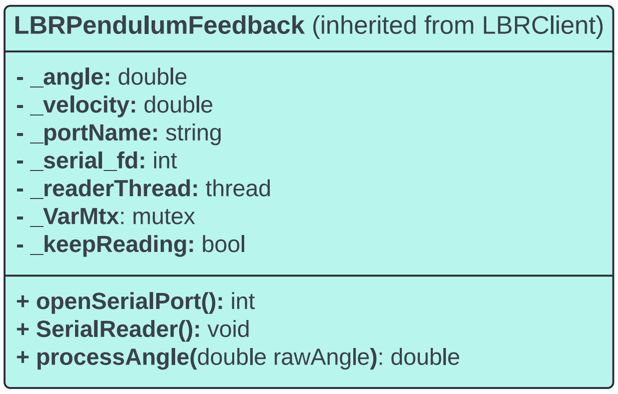


Figure 6.5: Electronics of the pendulum module installed temporally on a breadboard.

Another method from the `AS5600.h` library was then used to read immediate angular velocity $\dot{\theta}$ in rad.s^{-1}. Both of these values were then transmited over radio, utilizing the `RF24.h` Arduino library. The purpose of the Arduino script at the receiving end was to simply read the incoming data and send them over to the serial port of the computer. The Arduino C++ scripts for the transmitter and receiver can be found in Appendix A.

6.3 FRI client application

The features of the FRI client application had to be extended in order to correctly process the incoming data from the pendulum module. This was done by programming additional methods of the `UserClient` class (described in section 3.4.3). These additional methods are depicted in Figure 6.6. The first step involved opening


LBRPendulumFeedback (inherited from LBRClient)

- **_angle:** double
- **_velocity:** double
- **_portName:** string
- **_serial_fd:** int
- **_readerThread:** thread
- **_VarMtx**: mutex
- **_keepReading:** bool

+ **openSerialPort():** int
+ **SerialReader():** void
+ **processAngle(**double rawAngle**)**: double

Figure 6.6: UML diagram of `LBRPendulumFeedback` class.

the serial port. This is done by specifying a port name in the `_portName` attribute and calling `openSerialPort()` in the class constructor, which was programmed to return the file descriptor `_serial_fd` of the serial port. Our goal was to continuously process the serial port data in parallel with the FRI communication. For this reason, we implemented the serial port reading method `SerialReader()` as a separate programme thread running concurrently with the main FRI thread. The serial reading thread continuously updates the `_angle` and `_velocity` variables and the FRI thread reads them when necessary. The multithreading is illustrated in Figure 6.7. The client application with these features is available in Appendix A under the name `LBRPendulumFeedback`.

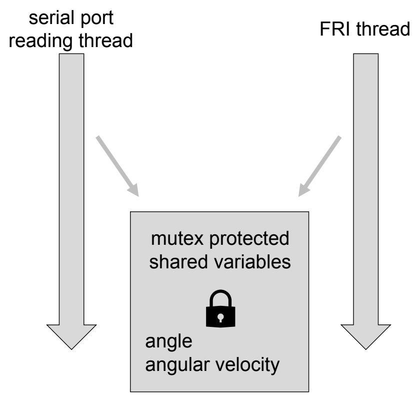


Figure 6.7: Multithreading in the FRI client application.

Chapter 7
Conclusion

In this bachelor thesis, we explored the various ways of controlling a KUKA LBR iiwa collaborative robot with a focus on its involment in dynamic control experiments. First of all, we have successfully set up the infrastructure necessary for robot control from an external desktop PC, including the setup of the PC itself, its network settings and an additional safety button for the manipulator. Our goal was to construct a setup in which real-time control systems (as illustarred in Figure 1.3) could be implemented.

We proceeded by implementing and testing most of the suitable software options for real-time robot control, covering mainly the KUKA Sunrise Toolbox and the Fast Robot Interface option. The KST appears to be a well-rounded option, combining the benefits of real-time control and ease of application development, due to the fact that KST is a MATLAB add-on. However, if precise control of communication speed is required, the FRI is the more suitable option. We have also extended the functionalities of the FRI C++ class templates by implementing C++ methods of our own. Altough the incorporation of FRI into the ROS2 environment was successfull (thanks to the open-source `lbr_stack` [10] framework), further development needs to be done in this area, partly because the `lbr_stack` framework itself is still in active development and has significantly changed even throughout writing this thesis.

Furthermore, we have developed two demonstrations of dynamic control which involved open-loop control of a ball suspended on a string from the LBR iiwa's end effector (Chapters 4 and 5). Videos of the crane-like system demonstration from Chapter 4 have been made available. The demonstration involving the Cup-and-ball game proved to be more challenging, but served as a good testing ground for the Fast Robot Interface. We suggested how the results of ball swing-up can be further improved by constructing the condition for maintaining tension in a string, see equation (5.9). In both demonstrations, acceleration of the end effector was (indirectly) controlled via real-time positions commands. During the preparation of this thesis, direct torque control of the LBR iiwa was also considered (see Appendix B) but it introduced other, more complex challenges.

Lastly we have designed and constructed a working pendulum module that can be attached to the end effector and provide real-time feedback on the pendulum's angle and angular velocity. We believe that from a hardware perspective, this module is ready for its use in experiments.

In conclusion, the KUKA LBR iiwa collaborative robot proved to be a good choice when it comes to actuating and controlling a dynamic system. There are several areas in which future work can be pursued. A natural continuation would be the implementation of software-based observer and controller for the pendulum module. Alternatively, other client command modes of the FRI, such as the joint torque control, can be examined, or the ROS2 infrastructure can be extended.

Appendix A
Software: scripts and other files.

The following public repository contains the software and other material referenced in this thesis:
https://gitlab.fel.cvut.cz/mestemar/dynamic-control-of-control-of-collaborative-robot-kuka-lbr-iiwa-demonstrator-system. The structure of the repository is as follows:

```
/
├── Sunrise Workbench
│   └── template.java
├── FRI
│   ├── FRI Applications for the robot controller (Java)
│   │   ├── MyJointHoldOverlay.java
│   │   ├── MyTorqueHoldOverlay.java
│   │   └── MyWrenchHoldOverlay.java
│   └── FRI Client Applications
│       ├── LBRMethods
│       ├── LBRMonitor
│       ├── LBRJointPTP
│       └── LBRReadWrite
├── Local ROS2 workspace
├── Crane-like system demonstration
│   ├── crane_2D_simulation.m
│   ├── Points2TrajectorySym.m
│   ├── KST_IS_realtime.m
│   ├── input_shaping_validation.mp4
│   └── demonstration.mp4
├── Cup-and-ball
│   ├── CAB_main.m
│   └── postprocessing.m
├── Pendulum
│   ├── STL_files
│   ├── nrf24_transmitter.ino
│   ├── nrf24_receiver.ino
│   └── LBRPendulumFeedback
└── object tracking in Python
```

Appendix B

Note on torque control of LBR iiwa via FRI

The purpose of this short section is to document the challenges of direct joint torque control of KUKA LBR iiwa through the Fast Robot Interface. As mentioned, in section 3.4.1 there are 3 total client command modes of the FRI, one of them being joint torque client command mode. In theory, this mode would be very appropriate for dynamic control experiments because a significant amount of dynamic models considers either translational force or torque as system input. However, the KUKA manual on FRI [5] does not provide any information on the manipulator dynamic model used nor any information about the underlying dynamic controller. For this reason, significant amount of reverse engineering would be necessary in order to use this client command mode properly. We document some of our findings here.

Because of the fact that the joint torque command mode only functions in joint impedance control mode of the FRI-enabled Java application in the robot controller, we hypothesize that the dynamic model of the manipulator used probably consists out of two parts:

$$\boldsymbol{\tau} = \text{physical model} + \text{impedance control} \tag{B.1}$$

$\boldsymbol{\tau}$ represents the 7-element vector of joint torques. We believe that by setting the stiffness and damping constants of the impedance control to 0 in the FRI Java application we effectively eliminate the impedance term in (B.1) so that the the dynamic model becomes [22]:

$$\boldsymbol{\tau} = \mathrm{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathrm{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \mathrm{G}\ , \tag{B.2}$$

where $\mathrm{M}, \mathrm{C}, \mathrm{G}$ are the mass matrix, Coriolis matrix and vector of gravity respectively. We tried to monitor joint torques during the execution of a fast trajectory of the manipulator in order to compare the predicted joint torques from equation (B.2) to the monitored data. This comparison can be seen in Figure B.1. From this comparison, it is clear that the model governed by B.2 is not entirely sufficient to describe what joint torques are needed to be commanded to perform a certain trajectory, as the predicted joint torques fit only in cases where large torque values have been monitored. We further hypothesize that static joint friction could play a role in the dynamic model. We have measured the values of static friction for each joint experimentally (by sending ramp signals to each joint, one at the time). The torque values at which the static friction has been overcome are available in Table B.1.

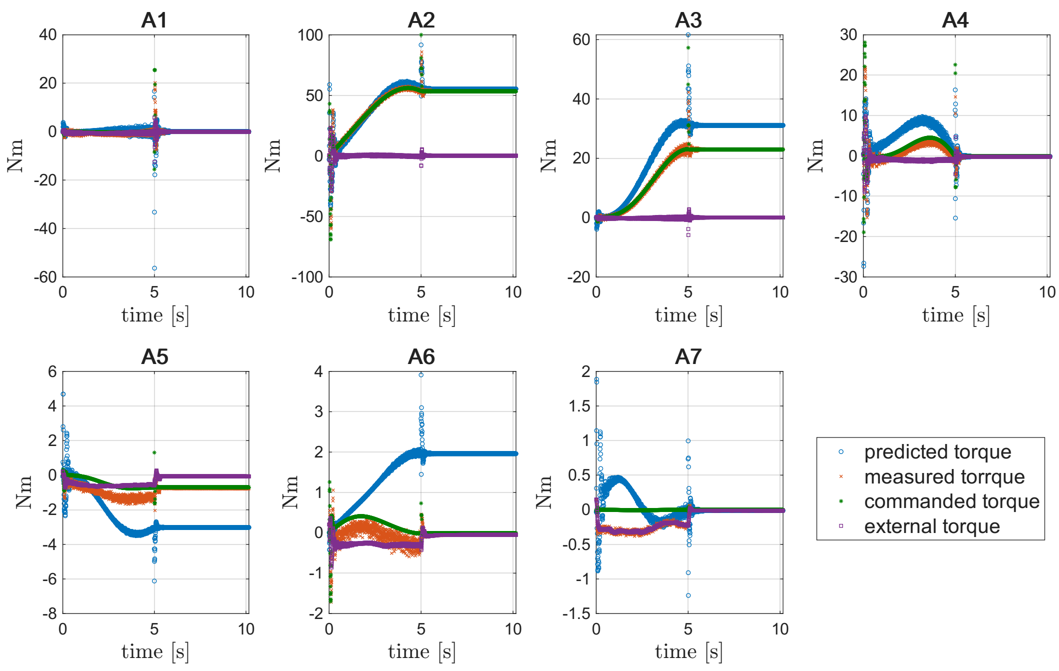


Figure B.1: Comparison of predicted and monitored torque for each joint.

Table B.1: List of threshold joint torques at which the static joint friction was overcomed.

joint label	A1	A2	A3	A4	A5	A6	A7
threshold joint torque [Nm]	1.22	2.89	2.66	2.43	1.25	0.98	0.83

Additionally, even if rigorous dynamic model of the manipulator would be known, torque control would also had to govern the gravity compensation for the tool attached to the end effector. In conclusion, the low-level joint torque control of LBR iiwa would be very fitting for dynamic control experiments, but it introduces new challenges to basic motion control. On the other hand, FRI joint position control has an internal dynamic controller built in which means that the real-time motion control is much more straightforward.

Appendix C

Bibliography

[1] KUKA Deutschland GmbH, Augsburg, Germany. *Robots LBR iiwa 7 R800, LBR iiwa 14 R20 Specification (KUKA manual)*, 2019.

[2] Minh Nhat Vu, Christian Hartl-Nesic, and Andreas Kugi. Fast swing-up trajectory optimization for a spherical pendulum on a 7-dof collaborative robot. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10114–10120. IEEE, 2021.

[3] Carlos Faria, João L. Vilaça, Sérgio Monteiro, Wolfram Erlhagen, and Estela Bicho. Automatic denavit-hartenberg parameter identification for serial manipulators. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 610–617, 2019.

[4] Christian Larsen. Including a collaborative robot in digital twin manufacturing systems. Master's thesis, Chalmers University of Technology, SE-412 96 Gothenburg, 2019. Avalaible at `https://odr.chalmers.se/server/api/core/bitstreams/b2122839-96c0-4926-8945-ef24510f446e/content`.

[5] KUKA Deutschland GmbH, Augsburg, Germany. *Software Option KUKA Sunrise.FRI 1.17 (KUKA manual)*, 2020.

[6] KUKA Deutschland GmbH, Augsburg, Germany. *Operating and Programming Instructions for System Integrators (KUKA manual)*, 2021.

[7] KUKA Deutschland GmbH, Augsburg, Germany. *Controller, KUKA Sunrise Cabinet, Provozní návod (KUKA manual)*, 2019.

[8] Mohammad Safeea and Pedro Neto. Kuka sunrise toolbox: Interfacing collaborative robots with matlab. *IEEE Robotics & Automation Magazine*, PP, 09 2017.

[9] Mohammad Safeea and Pedro Neto. *User's manual: KUKA Sunrise Toolbox*. Coimbra University & Ensam University, 2018. [Online]. `URL:https://github.com/Modi1987/KST-Kuka-Sunrise-Toolbox/blob/master/KST_User's_Guide.pdf`.

[10] Martin Huber, Christopher E. Mower, Sebastien Ourselin, Tom Vercauteren, and Christos Bergeles. Lbr-stack: Ros 2 and python integration of kuka fri for med and iiwa robots, 2023.

[11] Christoph Hennersperger, Bernhard Fuerst, Salvatore Virga, Oliver Zettinig, Benjamin Frisch, Thomas Neff, and Nassir Navab. Towards mri-based autonomous robotic us acquisitions: a first feasibility study. *IEEE transactions on medical imaging*, 36(2):538–548, 2017.

[12] Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, Dimitris Chrysostomou, Simon Bøgh, and Nestor Arana-Arexolaleiba. A scalable and unified multi-control framework for kuka lbr iiwa collaborative robots. In *2023 IEEE/SICE International Symposium on System Integration (SII)*, pages 1–5. IEEE, 2023.

[13] Konstantinos Chatzilygeroudis, Matthias Mayr, Bernardo Fichera, and Aude Billard. iiwa_ros: A ros stack for kuka's iiwa robots using the fast research interface. [Online]. URL: http://github.com/epfl-lasa/iiwa_ros.

[14] Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for mit 6.832. *Working draft edition*, pages 23–25, 2009. [Online]. URL: https://homes.cs.washington.edu/~todorov/courses/amath579/Tedrake_notes.pdf.

[15] Nguyen Quang Hoang, Soon-Geul Lee, Hyung Kim, and Sang-Chan Moon. Trajectory planning for overhead crane by trolley acceleration shaping. *Journal of Mechanical Science and Technology*, 28(7):2879–2888, Jul 2014.

[16] Michele Ermidoro, Simone Formentin, Alberto Cologni, Fabio Previdi, and Sergio M. Savaresi. On time-optimal anti-sway controller design for bridge cranes. In *2014 American Control Conference*, pages 2809–2814, 2014.

[17] Shidi Li. Robot playing kendama with model-based and model-free reinforcement learning. *ArXiv*, abs/2003.06751, 2020.

[18] Bojan Nemec, Matej Zorko, and Leon Zlajpah. Learning of a ball-in-a-cup playing robot. pages 297 – 301, 07 2010.

[19] Monimoy Bujarbaruah. *Robust Model Predictive Control with Data-Driven Learning*. PhD thesis, 10 2022. Available at https://www.proquest.com/openview/60fb2327d74409d51be30b1fdd5ac202/1?pq-origsite=gscholar&cbl=18750&diss=y.

[20] Chung Choo Chung and John Hauser. Nonlinear control of a swinging pendulum. *Automatica*, 31(6):851–862, 1995.

[21] Mahar Faiqurahman, Diyan Novitasari, and Zamah Sari. Qos analysis of kinematic effects for bluetooth hc-05 and nrf24l01 communication modules on wban system. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 4, 05 2019.

[22] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.