Impact-Aware Manipulation by Dexterous Robot Control and Learning in Dynamic Semi-Structured Logistic Environments

Impact Posture Planning for Dynamic Manipulation

Dissemination level	Public (PU)
Work package	WP2: Learning
Deliverable number	D2.1
Version	F-1.0
Submission date	11/01/2023
Due date	31/12/2022

www.i-am-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871899

Authors

Authors in alphabetical order		
Name	**Organisation**	**Email**
Ahmed ZERMANE	CNRS	ahmed.zermane@lirmm.fr
Niels DEHIO	CNRS	Niels.Dehio@kuka.com
Abderrahmane KHEDDAR	CNRS	kheddar@lirmm.fr

Control sheet

Version history			
Version	**Date**	**Modified by**	**Summary of changes**
0.1	23/12/2022	Ahmed ZERMANE	TOC & first contents
0.11	30/12/2022	Abderrahmane KHEDDAR	Improvement first draft
0.12	31/12/2022	Ahmed ZERMANE	Improvement first draft (Adding clarifications)
0.5	31/12/2022	Abderrahmane KHEDDAR	Pre-final version ready for peer-review
0.51	09/01/2023	Jari VAN STEEN	Comments added as peer review
0.9	10/01/2023	Ahmed ZERMANE	Peer-review comments addressed
1.0	10/01/2023	Abderrahmane KHEDDAR	Revised version ready for submission, quality check

Peer reviewers		
	Reviewer name	**Date**
Reviewer 1	Michael BOMBILE	06/01/2023
Reviewer 2	Jari VAN STEEN	09/01/2023
Reviewer 3	Jos DEN OUDEN	10/01/2023

TABLE OF CONTENTS

ABBREVIATIONS

Abbreviation	Definition
BIC	Bayesian Information Criterion
DIMT	Double Integrator Minimum Time
EC	European Commission
PU	Public
QP	Quadratic Programming
WP	Work Package

EXECUTIVE SUMMARY

The main objectives of I.Learn work package is to learn models of impacts and objects' dynamics resulting from impact, based on high-resolution simulation for model, also to compute postures that prepare robots to generate impacts and release events with its surrounding, possibly simultaneously in multiple locations, employing the learned impact models with an outcome that is aligned with the user-specified dynamic manipulation goals.
For different types of impact (swiping, tossing, grabbing and boxing) WP2 also aims to learn robot controllers and QP-parameters for multi-contact planning under stability constraints needed in I.Control.
The work package comprises five (5) tasks as follows:

1. Learning uncertainty models at impact;
2. Impact Posture Generator for Dynamic Manipulation;
3. Learning of Impedance and Dynamical Systems for control with impacts;
4. Learning of QP control weights, gains, Impedance;
5. Learning Benchmark and Progress Definition and Evaluation.

This deliverable focuses on the second item, that is: devising an impact posture generator for dynamic manipulation. Simply saying, given the initial state of a given manipulator robot (pose and pose velocity), find a trajectory along which the robot is steered to given desired impact location and impact velocity, or equivalently, a desired pose and pose velocity of the impacting end-effector. This is more of a planning than a static posture generator problem that CNRS has previously developed in multi-contact planning. Since we need to find a trajectory, formulating the problem by means of numerical optimization tools would lead to a semi-infinite programming (SIP) problem. The latter are known to be hard to solve efficiently and certainly not in real-time.
We have therefore investigated two possible approaches:

1. the first one is a data-driven approach that will generate impact poses from simulated experiments of intended use-cases; in order to investigate what it pertains in terms of modules and working efforts, we applied the general idea to the tossing scenario (as it was the first scenario to investigate see deliverable D5.3). Section 2 details the approach and more importantly considers the tossing and throwing as a single coupled problem. Hence data where generated by simulating millions of throws and considering the as the thrown object finale pose on the conveyor.
2. the second one is a combined planning-optimization approach. Applied for throwing and dual-arm grabbing (application to boxing is straightforward), it decouples the problem into planning for the end-effector to reach a given position with a given velocity from a given initial robot state, and then used for whatever serve the purpose. For example, in tossing, the planning module is coupled to parametrized ballistic trajectory for given object and an upper layer optimization seeks for the best coupling between the planner and the ballistic trajectory. In grabbing, this planing brick is used to steer the two arms to as both end-effectors desired poses and velocities are reached at a rendez-vous. In boxing, we think that it's usage is straightforward as the robot shall simply achieve a given impact at a given location (eventually with a held object).

1 INTRODUCTION

1.1 I.AM. project background

Europe is leading the market of torque-controlled robots. These robots can withstand physical interaction with the environment, including impacts, while providing accurate sensing and actuation capabilities. I.AM leverages this technology and strengthens European leadership by endowing robots to exploit intentional impacts for manipulation. I.AM focuses on impact aware manipulation in logistics, a new area of application for robotics which will grow exponentially in the coming years, due to socio-economical drivers such as booming of e-commerce and scarcity of labour. I.AM relies on four scientific and technological research lines that will lead to breakthroughs in modeling, sensing, learning and control of fast impacts:

1. I.Model offers experimentally validated accurate impact models, embedded in a highly realistic simulator to predict post-impact robot states based on pre-impact conditions;
2. I.Learn provides advances in planning and learning for generating desired control parameters based on models of uncertainties inherent to impacts;
3. I.Sense develops an impact-aware sensing technology to robustly assess velocity, force, and robot contact state in close proximity of impact times, allowing to distinguish between expected and unexpected events;
4. I.Control generates a framework that, in conjunction with the realistic models, advanced planning, and sensing components, allows for robust execution of dynamic manipulation tasks.

This integrated paradigm, I.AM, brings robots to an unprecedented level of manipulation abilities. By incorporating this new technology in existing robots, I.AM enables shorter cycle time (10%) for applications requiring dynamic manipulation in logistics. I.AM will speed up the take-up and deployment in this domain by validating its progress in three realistic scenarios: a bin-to-belt application demonstrating object tossing, a bin-to-bin application object fast boxing, and a case depalletizing scenario demonstrating object grabbing.

1.2 Deliverable topic background

Intentional impacts require to steer a robot in a configuration (that we call end posture) that allows it reaching the operational impact targets, i.e., task-space desired impact at best. Such configurations shall fulfill the set of objectives dictated by the tasks (including the desired pre- or post- impacts parameters if needed), under various constraints dictated by the limitations of the robot in terms of maximum resultant impulses that can be possibly handled/absorbed by the hardware. This is crucial in complex robotic systems (e.g., dual-arm robot manipulators eventually ported with a mobile base), where it has been used to find feasible multi-contact postures in both planning and control [1, 2, 3]. For a given robotic system, we consider the problem of finding a configuration to achieve a set of objectives, a subset of which can be impact-based (e.g., make a box start sliding with a desired acceleration by hitting it with the end effector). The outcome would answer the question: what posture and pre-impact speed does a robot has to take in order to achieve a desired post-impact speed in the task space?
Prior to I.AM project, CNRS developed such a “posture generator” for any kind of robotic systems that computes static posture for a set of tasks (including contact force as decision variable, i.e., as part of the

"posture") [3]. Indeed, the desired force, that will be extended here into desired impact with a given location will have a great influence on (i) the configuration the robot has to take and eventually on (ii) the other contact locations. The software consisted in an open-source customized non-linear optimization solver that operates directly on manifolds. At the writing of the proposal, our idea was to add constraints and models to the existing posture generator in order to extend it to impact tasks. However, in the course of the project we realized that this idea is not possible for the following reasons:

- the posture generators developed in [1, 2, 3] generate static postures, even if the final goal is to make a contact on a surface, eventually with a given force, it reasons on pure geometry $x = g(q)$, g being the forward geometry, x the operational robot space and q the robot joint space, and using the known relation resulting from the virtual work principle in mechanics $\tau = J(q)f$, τ being the actuators (joint) torques, $J(q)$ the robot Jacobian and f the contact forces. As you may notice, initial posture and condition have no influence on the resulting posture. That is to say, once a posture is computed and exist, there is no guarantee that a trajectory exist that bring the robot from any initial condition to the desired posture (e.g., steering methods in planning). Since a desired impact means a desired velocity in the operational space, we could have used the relation $\dot{x} = J(q)\dot{q}$, where $\dot{x}$ and $\dot{q}$ are the time derivative (velocity) in the operational and joint spaces respectively and simply make sure to add bounds on the joint velocities in the general optimization problem (increased with new decision variables). Similarly, we have no guarantee that the returned solution is feasible.

- the optimization problem being non-linear, we have experienced the solver to get stuck in local minima problem and considerable computation time. In I.AM. we are rather interested by having an impact solution that is effectively feasible and if possible, fast enough to be used in industry setting with repetitive and cycling toss, grab, box of various objects.

1.3 Purpose of the deliverable

This deliverable aims at providing a so called Impact-driven "posture generator", in fact it is an impact-driven planner. That is to say, a software module that takes as input: (i) the robot model; (ii) the current robot configuration (current joint positions and joint velocities); and (iii) desired task-space impact of a given end-effector (or equivalently the desired position and velocity of impact) and provides as output the robot trajectory that achieves goal (iii) when it exists. The planner can be coupled to parametrized ballistic trajectories for tossing scenarios, or used in synchrony with dual-arm grabbing scenarios, or used as is in boxing scenarios.

1.4 Intended audience

The dissemination level of this report (D2.1) is 'public' (PU) – meant for members of the Consortium (including Commission Services) and the general public.

2 DATA-DRIVEN POSTURE GENERATOR

In order to generate desired impact postures, we have investigated first a data-driven approach. The idea is to generate various motions for a given robot and register the set of reachable space with various velocities for each point starting from different initial conditions. However, we could also consider the whole impact task process. For instance, in the tossing use-case, instead of learning only the operational reachable pose and velocity states, we could consider learning the entire throwing process and consider later as input only the desired landing spot and configuration of a given thrown object. We can then use this example to build rather a learning methodology and if successful, the methodology can be extended to the grabbing and boxing use-cases.

2.1 Overview

We focus on the robotic tossing use-case of objects in a scenario defines in I.AM. (cf. Fig. 1). The research problem is to toss the object with the robot manipulator such that it lands at a conveyor belt with a given orientation or a given state at rest. A common reason for that is a bar code that needs to be oriented in a certain way to be scanned easily. First, we consider objects that can be treated as a single rigid body with known dynamics and shape. Generalization can be done in a second stage by generating data with objects having various sizes, weight, shape and that can be soft.
In our approach, we decompose the challenge of tossing objects in two subsequent sub-problems:

1. select a reasonable release configuration;
2. plan and execute with the robot a potentially non-linear point-to-point motion in joint space that passes through the desired release configuration.

Both sub-problems constitute constrained optimization problems, as the actuated robot itself has limited capabilities due to its shape and mechatronics limitations. The complete pipeline of our method is illustrated in Fig 2.

2.1.1 Environment

The experimental platform is a 7 degrees-of-freedom Panda manipulator from Franka Emika, controlled at 1 ms update rate. The robot's hardware limits are specified on the manufacturer's website[1]. We attach to the flange a special gripper provided by Smart Robotics with a suction cup. We employ our new interface implementation to operate Panda and pump simultaneously at different control frequencies[2], see also the related publication [4]. In the experiments presented here, we toss a cardboard box having a mass of 0.566 kg (shape $0.272\ m \times 0.155\ m \times 0.114m$).
We use a realistic, validated model of the innovation lab at TU/e as the simulation environment. In this setup, the panda robot is mounted on a table. A given conveyor is located nearby and placed such that the conveyor belt moves with a constant speed of 1 m/s along the negative y-axis w.r.t the world frame, cf. Fig 1.
Realistic friction and restitution parameters for the box-impact events have been identified and validated based on real-world impact data. This ensures that simulated box-impacts by *AGX Dynamics*[3] are similar

[1] https://frankaemika.github.io/docs/control_parameters.html#constants
[2] https://github.com/jrl-umi3218/mc_franka
[3] https://www.algoryx.se/agx-dynamics

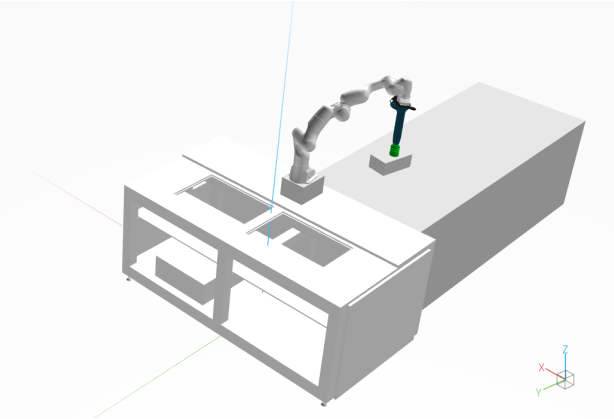


Figure 1: Realistic simulation model of the innovation lab (validated). The conveyor belt moves along the negative y-axis (green) of the world frame, drawn in the lower right corner.

to the real box-impacts. In this work, we are ignoring effects due to wind or air drag which is reasonable for short distances and considerable object weights.

2.2 Selecting Optimal Release Configurations

In a robotic tossing scenario, the object's trajectory flying in space, impacting, bouncing, tumbling, sliding, and finally landing at a certain rest pose depends on the robot's end-effector pose and end-effector velocity at the moment of release, hereafter referred to as end-effector release configuration[4]. It is important to note that the robot's nullspace posture and nullspace speed at the moment of release does neither affect the object's flight trajectory nor its rest pose.
Our goal is to select an optimal end-effector configuration for releasing the object given a specific target rest orientation. The term "optimal" here means that we are aiming for a release configuration that is reliable with respect to slight variations (or disturbances), and hence, robustly ensures the object is landing with the desired surface in contact. Note that the robot in practice likely fails to drop off the object at the exact release pose with the exact release velocity. Thus, it is advantageous to aim for a release configuration that is robust to deviations.
The given research problem requires an inverse impact model. The impact dynamics can be accurately simulated forward in time. Due to the highly non-smooth dynamical impact effects (such as friction of two materials against each other, restitution coefficients...) it is however almost impossible to analytically infer an inverse impact model. Therefore we are utilizing learning techniques that can circumvent this problem. In order to acquire a data-set for the given robot manipulator comprising release configurations and corresponding object rest states, we propose to simulate the object's flight and landing trajectories, thereby including also highly non-smooth dynamical impact effects.

[4]Of course it depends also on the dynamical properties of the object and the environment (conveyor).

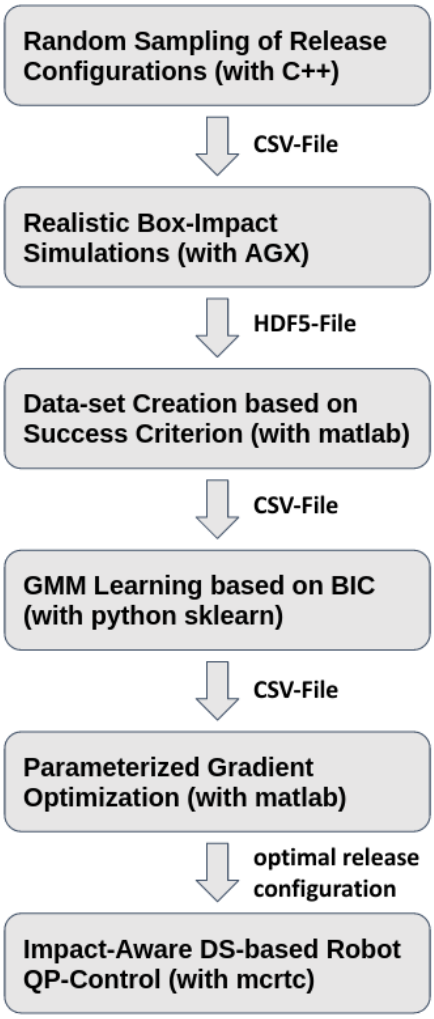


Figure 2: I.AM. tossing pipeline developed at CNRS.

2.2.1 Reduced Problem

In the following, we consider an end-effector configuration $\boldsymbol{\lambda} \in \mathbb{R}^{12}$ as the stacked vector of end-effector pose $\mathbf{x} \in \mathbb{R}^6$ (position & orientation) and end-effector (linear & angular) velocity $\dot{\mathbf{x}} \in \mathbb{R}^6$

$$\boldsymbol{\lambda} = \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix}$$

A naive approach explores all twelve dimensions of $\boldsymbol{\lambda}$ by testing systematically feasible joint displacements and velocities and applying forward kinematics. This approach would, however, result in an intractable amount of simulation trials since many tosses miss the desired target region.
In order to reduce the problem dimension for learning, we vary only five task-space dimensions and consider the remaining seven dimensions as constant:

- translation along x-axis $\rightarrow$ constant at 0.33 m.

- translation along y-axis $\rightarrow$ varied.
- translation along z-axis $\rightarrow$ varied.
- rotation around x-axis $\rightarrow$ varied.
- rotation around y-axis $\rightarrow$ constant at 0 rad.
- rotation around z-axis $\rightarrow$ constant at 0 rad.
- linear velocity along x-axis $\rightarrow$ constant at 0 m/s.
- linear velocity along y-axis $\rightarrow$ varied.
- linear velocity along z-axis $\rightarrow$ varied.
- angular velocity around x-axis $\rightarrow$ constant at 0 rad/s.
- angular velocity around y-axis $\rightarrow$ constant at 0 rad/s.
- angular velocity around z-axis $\rightarrow$ constant at 0 rad/s.

The world frame is indicated in Fig 1.

2.2.2 Constraints for Sampling Release Configurations

We aim at generating a list of end-effector release configurations $\boldsymbol{\lambda}$ as input for the dynamic box-impact simulations. The sampling in task-space is not trivial since we need to fulfill the robot's capabilities which are defined in the joint-space. Accordingly, the task-space limits are configuration-dependent.
We denote the Jacobian related to angular velocities as $\mathbf{J}_\mathbf{w}$ and related to linear velocities as $\mathbf{J}_\mathbf{v}$. Further, $\mathbf{w}$ and $\mathbf{v}$ denote angular and linear task-space velocities respectively.
The well-known non-linear mapping between joint-space velocities $\dot{\mathbf{q}}$ and task-space velocities is given by $\mathbf{w} = \mathbf{J}_\mathbf{w}\dot{\mathbf{q}}$ and $\mathbf{v} = \mathbf{J}_\mathbf{v}\dot{\mathbf{q}}$.
It is important to ensure that randomly generated release configurations are feasible, i.e., within the robot's joint position limits $\underline{\mathbf{q}} \leq \mathbf{q} \leq \bar{\mathbf{q}}$ and joint velocity limits $\underline{\dot{\mathbf{q}}} \leq \dot{\mathbf{q}} \leq \bar{\dot{\mathbf{q}}}$.
In addition, we further incorporate prior knowledge by enforcing release configurations with linear end-effector velocities that point towards the target: $-\infty \leq \mathbf{v}_y = \mathbf{J}_{\mathbf{v}_y}\dot{\mathbf{q}} \leq \mathbf{0}$ and $\mathbf{0} \leq \mathbf{v}_z = \mathbf{J}_{\mathbf{v}_z}\dot{\mathbf{q}} \leq \infty$. As mentioned above, we enforce zero linear velocity along the x-axis $\mathbf{v}_x = \mathbf{J}_{\mathbf{v}_x}\dot{\mathbf{q}} = \mathbf{0}$ and zero angular velocities $\mathbf{J}_\mathbf{w}\dot{\mathbf{q}} = \mathbf{0}$. Also, we aim for large velocities by enforcing $0.4 \leq -\frac{1}{2}\mathbf{v}_y + \frac{1}{2}\mathbf{v}_z \leq \infty$. In practice, ∞ is replaced by upper ou lower bounds.

2.2.3 Procedure for Sampling Release Configurations

Our sampling procedure for obtaining random but feasible release configurations is conducted in consecutive steps as follows:

1. Sample a random end-effector pose by varying the y-position, z-position, and the rotation-angle around the x-axis. The remaining three pose parameters are constant.
2. Perform inverse kinematics in order to obtain a corresponding (self-)collision-free joint posture within the joint angle limits. Thereby, confirm that the end-effector pose is within the robot's reachable space.

3. Map the robot's joint velocity limits for that particular joint posture onto the task-space, resulting in a convex polytope of feasible end-effector velocities (this requires a sophisticated configuration-dependent mapping technique to be explained below).

4. Sample a feasible end-effector velocity (linear along y- and z-axes) utilizing the generated task-space polytope. The remaining four velocity parameters are zero.

2.3 Polytope Mapping for Feasible Task-Space Velocities

When gathered, the previous inequalities form a convex, high-dimensional polytope that represents the set of feasible solutions in terms of $\mathbf{v}_y$ and $\mathbf{v}_z$. The task-space velocity bounds along y- and z-axis are coupled and cannot be treated separately. By choosing a vector $\mathbf{d} \in \mathbb{R}^2$ (treated as a ray that points in a certain direction), we obtain a feasible two-dimensional tuple (or vertex) consisting of extreme linear velocities along y- and z-axis through the linear program

$$
\begin{aligned}
&\underset{\dot{\mathbf{q}},\mathbf{v}_y,\mathbf{v}_z}{\arg\min} \quad [\mathbf{v}_y, \mathbf{v}_z]\,\mathbf{d} \qquad (1)\\
&\text{s. t.} \quad
\begin{bmatrix}
\mathbf{J}_{\mathbf{w}_x}, & \mathbf{0}, & \mathbf{0}\\
\mathbf{J}_{\mathbf{w}_y}, & \mathbf{0}, & \mathbf{0}\\
\mathbf{J}_{\mathbf{w}_z}, & \mathbf{0}, & \mathbf{0}\\
\mathbf{J}_{\mathbf{v}_x}, & \mathbf{0}, & \mathbf{0}\\
\mathbf{J}_{\mathbf{v}_y}, & -1, & \mathbf{0}\\
\mathbf{J}_{\mathbf{v}_z}, & \mathbf{0}, & -1
\end{bmatrix}
\begin{bmatrix}\dot{\mathbf{q}}\\ \mathbf{v}_y\\ \mathbf{v}_z\end{bmatrix} =
\begin{bmatrix}\mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}\\
&\qquad\quad
\begin{bmatrix}\underline{\dot{\mathbf{q}}}\\ -\infty\\ \mathbf{0}\\ 0.4\end{bmatrix} \leq
\begin{bmatrix}
\mathbf{I}, & \mathbf{0}, & \mathbf{0}\\
\mathbf{0}, & 1, & \mathbf{0}\\
\mathbf{0}, & \mathbf{0}, & 1\\
\mathbf{0}, & -\frac{1}{2}, & \frac{1}{2}
\end{bmatrix}
\begin{bmatrix}\dot{\mathbf{q}}\\ \mathbf{v}_y\\ \mathbf{v}_z\end{bmatrix} \leq
\begin{bmatrix}\bar{\dot{\mathbf{q}}}\\ \mathbf{0}\\ \infty\\ \infty\end{bmatrix}
\end{aligned}
$$

Solving (1) multiple times for different rays $\mathbf{d}$, we collect a list of extreme vertices, representing a convex polytope in the two-dimensional space of linear velocities $\mathbf{v}_y$, $\mathbf{v}_z$, whereas angular velocities $\mathbf{w} = \mathbf{0}$, and the linear velocity $\mathbf{v}_x = \mathbf{0}$ are constrained to zero. The resulting vertex representation allows for uniformly sampling a feasible end-effector velocity within the mapped polytope for the given joint configuration. Refer to [5] for an efficient algorithm to select useful search directions (rays). To our best knowledge, constructing such task-space velocity polytope for a redundant manipulator has not been proposed before. This mapping of course depends on the given Jacobian, and hence, has to be recomputed for each new joint posture.
In the next step, we can also allow non-zero angular velocities around the x-axis. This increases the problem dimension by one additional parameter, but allows for more complex object motion. Choosing

$\mathbf{w}_x \neq \mathbf{0}$ yields

$$\underset{\dot{\mathbf{q}},\mathbf{w}_x,\mathbf{v}_y,\mathbf{v}_z}{\arg\min} \quad [\mathbf{w}_x, \mathbf{v}_y, \mathbf{v}_z]\,\mathbf{d} \tag{2}$$

$$\text{s. t.} \quad \begin{bmatrix} \mathbf{J}_{\mathbf{w}_x}, & -1, & \mathbf{0}, & \mathbf{0} \\ \mathbf{J}_{\mathbf{w}_y}, & \mathbf{0}, & \mathbf{0}, & \mathbf{0} \\ \mathbf{J}_{\mathbf{w}_z}, & \mathbf{0}, & \mathbf{0}, & \mathbf{0} \\ \mathbf{J}_{\mathbf{v}_x}, & \mathbf{0}, & \mathbf{0}, & \mathbf{0} \\ \mathbf{J}_{\mathbf{v}_y}, & \mathbf{0}, & -1, & \mathbf{0} \\ \mathbf{J}_{\mathbf{v}_z}, & \mathbf{0}, & \mathbf{0}, & -1 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{w}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

$$\begin{bmatrix} \underline{\dot{\mathbf{q}}} \\ \underline{\dot{\mathbf{v}}}_x \\ \underline{\dot{\mathbf{w}}}_y \\ \underline{\dot{\mathbf{w}}}_z \\ 0.4 \end{bmatrix} \leq \begin{bmatrix} \mathbf{I}, & \mathbf{0}, & \mathbf{0}, & \mathbf{0} \\ \mathbf{0}, & 1, & \mathbf{0}, & \mathbf{0} \\ \mathbf{0}, & \mathbf{0}, & 1, & \mathbf{0} \\ \mathbf{0}, & \mathbf{0}, & \mathbf{0}, & 1 \\ \mathbf{0}, & \mathbf{0}, & -\frac{1}{2}, & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{w}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix} \leq \begin{bmatrix} \bar{\dot{\mathbf{q}}} \\ \bar{\dot{\mathbf{v}}}_x \\ \bar{\dot{\mathbf{w}}}_y \\ \bar{\dot{\mathbf{w}}}_z \\ \infty \end{bmatrix}$$

with $\bar{\dot{\mathbf{v}}}_x = \infty$, and $\bar{\dot{\mathbf{w}}}_y = \mathbf{0}$, $\bar{\dot{\mathbf{w}}}_z = \infty$.

2.3.1 Post-processing the data set obtained from Simulations

The generated list of release configurations are independently simulated utilizing the developed AGX Dynamics simulation framework. These simulations do not consider a robot, as we are only interested in learning the flight and impact behavior of the object. Estimated friction and restitution parameters remain constant. A simulation stops

- if the relative velocity between the box and the moving conveyor belt is zero (success), or
- if the box reaches the floor (failure), or
- after a maximum of 2 seconds (failure).

We post-process the simulated trajectories and exclude all failure cases ($\approx 25\%$).
The data-set is then constructed by storing a six-dimensional feature vector $\psi \in \mathbb{R}^6$ for each simulation, comprising input information (i.e., for the release) and output information (i.e., resulting rest state). In other words, each individual simulation constitutes a separate *demonstration* ψ parameterized by:

- translation along y-axis
- translation along z-axis
- rotation around x-axis
- linear velocity along y-axis
- linear velocity along z-axis
- ID of the box surface in contact with the conveyor

2.3.2 Learning the Release-Impact-Rest Model

The data-set becomes a high-dimensional point-cloud when treating each feature vector as a six-dimensional data point. We model the probability distribution of this point-cloud using the *Gaussian Mixture Model* (GMM) [6] approach. The trained model, $\mathcal{M}_{\text{impact}}$ composed of K Gaussians, is represented as

$$\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1,\cdots,K}.$$

The k-th Gaussian is parameterized by prior π_k, mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$, respectively. These parameters are calculated employing the *Expectation Maximization* (EM) algorithm [7]. The optimal number of Gaussians K for the impact model can be determined utilizing the Bayesian Information Criterion (BIC) [8].
The probability density of a given feature vector $\boldsymbol{\psi}$ for the impact model $\mathcal{M}_{\text{reach}}$ is then given by the likelihood

$$\mathcal{P}(\boldsymbol{\psi}|\mathcal{M}_{\text{impact}}) = \sum_{k}^{K} \pi_k \mathcal{N}(\boldsymbol{\psi}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{3}$$

The likelihood is a measure of the density of feasible demonstrations in the immediate vicinity. Or, in other words, it is the evaluation of the probability density at $\boldsymbol{\psi}$. It represents a scalar fitness measure of the six-dimensional demonstration $\boldsymbol{\psi}$. In this specific case, it describes the probability to land the object with a given contact surface when releasing it at a given release configuration. High density (likelihood) in a region results from the fact that this region was more frequently explored by the provided demonstrations; hence, it has a high probability of representing a feasible end-effector configuration resulting in a smooth impact event, whereas less-dense regions (likelihood less than a threshold) are bad regions, as they are rarely seen in the provided demonstrations. The latter may be due to the robot's joint limits and (self-)collisions, or because of missing the conveyor belt, or due to chaotic impact behavior. In either way, in these low-density regions, the existence of a feasible and collision-free joint posture and the resulting object orientation at rest is known with much less certainty.
It is important to note that the impact model does not allow querying for joint configurations. It does not include inverse kinematics information that would allow doing this. However, due to the sampling process with positive collision-free examples, we know that for each reachable end-effector configuration, there exists at least one feasible corresponding joint-space configuration.
By embedding the set of suitable release configurations in a probability density function, we directly check if a certain end-effector configuration is reachable *and* favorable for releasing the object, without relying on impact simulations or an inverse kinematics solver at run-time.

2.3.3 Release Configuration Optimization

The optimal end-effector configuration $\boldsymbol{\lambda}^*$ for releasing the object such that it lands with a specific rest orientation is related to the feature vector with the highest likelihood according to (3)

$$\boldsymbol{\psi}^* = \arg\max_{\boldsymbol{\psi}} \; \mathcal{P}\left(\boldsymbol{\psi}|\mathcal{M}_{\text{impact}}\right)$$

A locally optimal $\boldsymbol{\psi}^*$ is obtained by computing gradient ascent on the impact-model $\mathcal{M}_{\text{impact}}$ derived in the previous subsection. The Jacobian of this objective function is given by

$$\frac{\partial \mathcal{P}(\boldsymbol{\psi}|\mathcal{M}_{\text{impact}})}{\partial \boldsymbol{\psi}} = -\sum_{k}^{K} \pi_k \boldsymbol{\Sigma}_k^{-1} \left(\boldsymbol{\psi} - \boldsymbol{\mu}_k\right) \mathcal{N}(\boldsymbol{\psi}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4}$$

We initialize the gradient ascent with a random release configuration drawn from the learned GMM and the desired target rest orientation. In order to keep the specific rest orientation fixed during optimization, for the gradient ascent we consider only the first five dimensions of the Jacobian (4) related to the release configuration and nullify the remaining ones. The output of the optimization is a local optimum, i.e., a feature vector that has locally the highest likelihood.

2.4 Joint space release configuration

Considering a redundant robot manipulator, the desired optimal end-effector release configuration $\boldsymbol{\lambda}^*$ can be achieved by infinite different joint-space release configurations. The following method describes how to obtain a specific joint-space release configuration. First, apply inverse kinematics to select a random joint-space posture $\mathbf{q}^*$ that matches the desired end-effector position $\mathbf{x}^*$. Second, compute the associated Jacobian $\mathbf{J}$. Third, solve a quadratic program for that specific posture utilizing the weight matrix $\mathbf{W}$ to obtain joint velocities $\dot{\mathbf{q}}^*$ that generate the desired end-effector velocities $\dot{\mathbf{x}}^*$.

$$\begin{aligned} \underset{\dot{\mathbf{q}}}{\arg\min} \quad & \dot{\mathbf{q}}^T \mathbf{W} \dot{\mathbf{q}} \\ \text{s.t.} \quad & \mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{x}}^* \end{aligned} \tag{5}$$

This method returns a joint-space release configuration (consisting of joint-space posture $\mathbf{q}^*$ and joint velocities $\dot{\mathbf{q}}^*$) for the optimal end-effector release configuration. However, due to the iterative process, this method does not ensure that the best joint-space release configuration is selected with respect to trajectory planning and the robot's current state.

2.5 Tossing with the Panda Robot

In order to reach the desired optimal end-effector release configuration $\boldsymbol{\lambda}^*$, we need to plan (and execute with the robot) a potentially non-linear point-to-point motion in the joint space. The planning problem is constrained because of the robot dynamics and its mechanical and electrical hardware limitations. Adopting the method for softly catching an object in flight [9] allows us to meet $\boldsymbol{\lambda}^*$ with the end-effector as a via-point.
EPFL has devised a robot-independent second-order task-space dynamical system (DS) that has been integrated into the mc_rtc framework [10]. Utilizing the newly developed AGX interface, we can execute the DS-based mc_rtc controller with AGX dynamics simulation and for tosses on the real setup.
As an alternative, also second-order joint-space DS can steer the robot towards the release configuration providing joint-space accelerations, see Fig 3.
This controller represents a quadratic program (QP) that is solved every millisecond. It includes the robot hardware limits and robot dynamics as strict constraints. As the quadratic objective, the QP minimizes joint accelerations such that the reference acceleration provided by the DS are tracked as close as possible. Therefore, the mc_rtc controller requires no specification of meta-parameters. Finally, the QP joint commands are provided to the newly developed interface for the panda robot. This interface also allows actuating the suction pump.

2.6 Quantitative Evaluation

We generated and simulated $300,000$ release configurations which resulted in $225,407$ trials with the box landing on the moving conveyor belt within 2 seconds. The distribution of surfaces in contact with

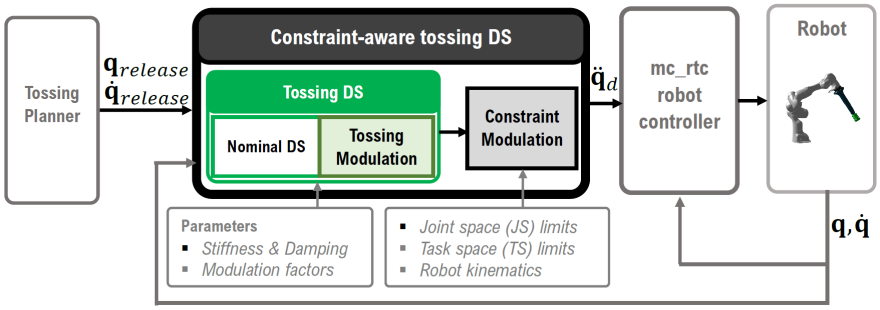


Figure 3: Overview of the constraint-aware tossing DS and its integration within the system

the belt is shown in Fig. 4. As expected, the box lands only with four different contact surfaces. This is due to the release configuration sampling procedure.

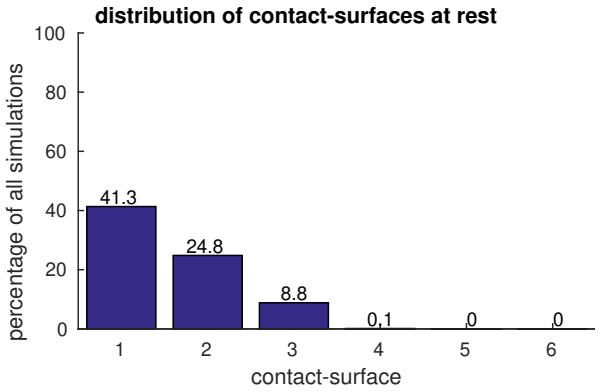


Figure 4: The box object has six flat surfaces with which it can be in contact at rest. The six bars indicate their percentage given all simulations.

For visualization, we plot 1000 randomly drawn initial configurations in Fig. 5 and Fig. 6. These distributions are not uniform because of the various constraints.

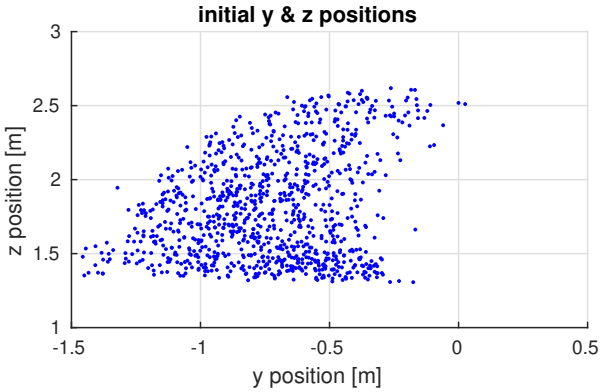


Figure 5: Distribution of 1000 initial release y- and z-axes positions.

We construct the data-set as described above and learn a GMM utilizing the python sklearn framework

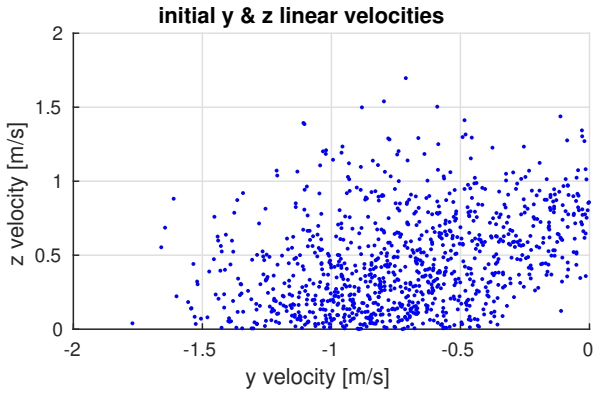


Figure 6: Distribution of 1000 initial release velocities along y- and z-axes.

for different number of Gaussian components and different types of Gaussians. The BIC curve is shown in Fig. 7. It exhibits classical insight. Simple models with few Gaussians result in large BIC values. Increasing the model complexity leads to lower BIC values up to a certain point, then the model starts over-fitting the data and the BIC values converge. Based on this analysis, the optimum number of components is determined as 40, selecting full Gaussians. This model is used in the following experiments.

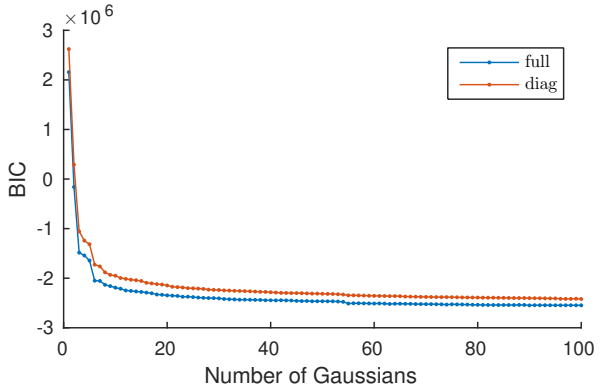


Figure 7: Distribution of 1000 initial release velocities along y- and z-axis.

For each of the four explored surface numbers, we perform the aforementioned gradient ascent utilizing backtracking line search 50 times with different random number initialization and starting with randomly drawn initial states. As this approach yields local optima, we here select the 10 most promising feature vectors with the highest likelihood. Simulating the corresponding release configurations reveals the following success rates:

- $10/10$ trials succeed for surface number 1
- $10/10$ trials succeed for surface number 2
- $10/10$ trials succeed for surface number 3
- $0/10$ trials succeed for surface number 4

Finally, we initialize the DS with these optimal release configurations and toss the box with the panda robot in AGX simulation. This work was made in a joint CNRS-EPFL collaboration.

2.7 Conclusion

Generating postures using a data-driven approach is possible by following in gross a similar methodology. This work was integrated in the TOSS scenario, deliverable D5.3. However, CNRS does not possess enough knowledge in data-science to pursue further in this direction and EPFL partner has taken the lead in such approaches. Because of different varying parameters, e.g., objects shape, weight (or varying inertia), and nature (rigid, flexible)... it appears a huge endeavor to explore further this direction. Therefore, we started in parallel a more conventional model-based approach based on planning and numerical optimization. In a later stage, it may be interesting to investigate if both approaches can be merged to produce a more efficient impact-driven plans.

3 PLANNING-BASED POSTURE GENERATOR

3.1 Overview

Our work's long-term context is to enable efficient robotic fast grabbing, tossing, and boxing objects in automated industrial sorting chains. In such applications, robots shall reach, pick and toss or place objects of different sizes, shapes, and materials from one location to another as fast as possible. The proposed frameworks in [11, 12] and many other related works slow down drastically robot motion (up to zero relative velocity) when establishing contact with the environment. We instead aim to generate powerful impacts intentionally.
The previous Section 2 investigated a data-driven approach to generate impact-driven robot postures and was exemplified in tossing. In this section we approach the problem in a decoupled way. First we devise a planning brick that plans trajectories from initial robot end-effector pose/velocity to a desired pose/velocity. Then we connect this planning brick with another brick depending on the use-case of impact we address.
For the planning brick, our main contribution is to rely on a sampling based algorithm Bi-RTT [13]. We use two different approaches separately for the steering method mainly to compare their reliability in reaching intended impact contacts with desired velocities. The first approach relies on acceleration limited trajectory generation where a Double Integrator Minimum Time (DIMT) is the neighborhood criterion [14]. This allows us to have non-zero start and end velocities of the planned trajectories. The second one is jerk-limited [15] compared to the first we can add both starting and target accelerations and also impact point acceleration.
Compared to related works, the novelty of our approach lies in defining task coordinates (pick, toss and place) as a via-point which will be integrated in a time-constrained path. The latter is optimized of a generated samples' map, see Fig. 8. Joint limits are also considered within the steering method for the DIMT [14]. For long trajectories, the sample-map segments with limited duration would also prevent the numerical integration errors, thus there won't be any need to define an upper limit time for the whole trajectory [15].

3.2 Approach

The main framework is a sampling-based method (extended BiRRT algorithm) in the configuration space, see Fig 8. It grows two trees starting from the initial and final states, see Fig. 9. The steering method in Fig. 1 lines 12 and 18 is either the DIMT or the jerk-based one. It is evaluated to decide whether a new sample could be added to one of the trees.

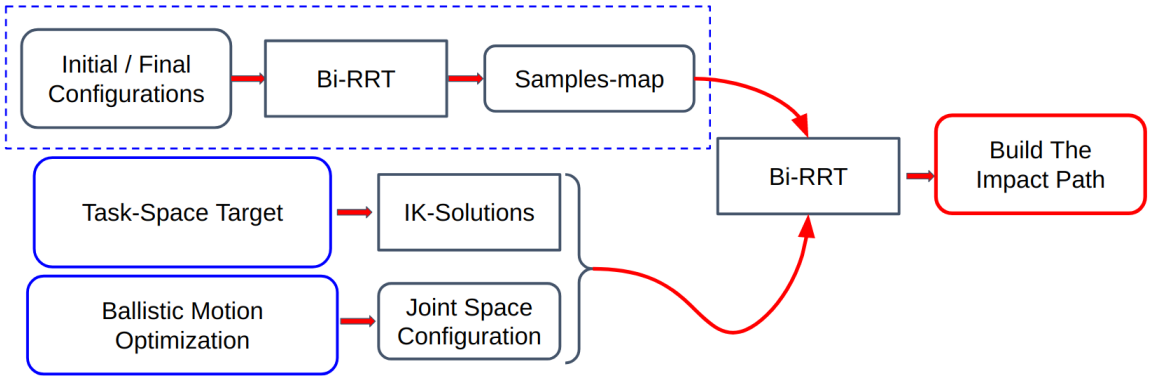


Figure 8: The planning approach relies mainly on Bi-RRT in order to get free collision path with a desired impact via-point.

This steering method, algorithm 2, has a neighboring function which determines the closest neighbor based on a a distance function. Different metrics were used as distance functions in the RRT algorithms [16].
In [17] for example, a complex parameter-dependent distance function links the euclidean distance between two robot configurations and corresponding rigid body transformation. Another distance function used in [18] adds velocity distance term to the euclidean configuration distance in order to count for desired target velocity. This latter can be replaced by the one we use in our approach, which based on go-to time between two states [14, 19]. Despite not being a metric because of its asymmetry, it is proven to be reliable, parameter-free, and accounts for both position and velocity.

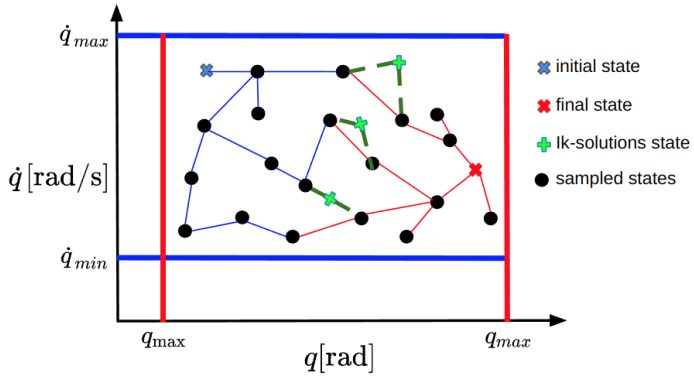


Figure 9: The sampled states are chosen within the feasible set for each DoF, defined by velocity limits (horizontal blue lines) and position limits (vertical red lines), the two grown trees are defined by blue and red edges, each corresponds to its starting point.

For the acceleration-limited trajectory generation, each state is represented by the joint position and velocity $[q, \dot{q}]$. This will be extended by adding acceleration $[q, \dot{q}, \ddot{q}]$ for jerk-based steering method.
As shown in the Algorithm 1, a connection between two trees can be made only when the randomly generated state Q_r is successfully added to both trees, but this doesn't guarantee that we would have a better path. Only a better-cost one is chosen (line 25).
Each task trajectory can be generated simply by skipping path selection at the first phase of building

sample-map for N iterations. After that the via-point represented in the joint space can be considered as the new random state. Then the algorithm 1 will extract the path that fulfills all the initial and final conditions. For redundant robots, we would have for the same task's via-point multiple joint-space configurations that will compete to generate the best path.

Algorithm 1 Extended BiRRT Algorithm

1: **procedure** BiRRT($q_{\max}, q_{\min}, \dot{q}_{\max}, \ddot{q}_{\max}, P_{\text{init}}, P_{\text{final}}$, N)
2: $\text{cost}_i \leftarrow 0$, $\text{cost}_f \leftarrow 0$
3: FinalPath $\leftarrow [\,]$; FinalCost $\leftarrow \infty$;
4: $V1 \leftarrow [P_{\text{init}}, Q_{\text{init}}, \text{cost}_i]$;
5: $V2 \leftarrow [P_{\text{final}}, Q_{\text{final}}, \text{cost}_f]$;
6: $E1 \leftarrow [\,]$; $E2 \leftarrow [\,]$;
7: d $\leftarrow$ true; % growing direction indicator
8: counter $\leftarrow 0$;
9: **while** counter $\leq$ N **do**
10: $Q_r \leftarrow$ SampleInJointSpace;
11: $P_r \leftarrow$ ForwardKinematics(Q_r);
12: $T_1, S_1, \text{cost}_{r_i}, \text{JointLimits} \leftarrow$ Steering$(Q_r, V1, \text{d})$;
13: **if** ¬JointLimits **then**
14: $Q_i, P_i \leftarrow$ SegmentsGeneration(T_1, S_1);
15: $E_{\text{temp}} \leftarrow$ BuildEdges(Q_i, P_i, S_1);
16: $V1 \leftarrow V1 \cup [P_r, Q_r, \text{cost}_r] \cup [P_i, Q_i, \text{cost}_i]$;
17: $E1 \leftarrow E1 \cup E_{\text{temp}}$;
18: $T_2, S_2, \text{cost}_{r_f}, \text{JointLimits} \leftarrow$ steering$(Q_r, V2, \bar{\text{d}})$;
19: **if** ¬JointLimits **then**
20: $Q_i, P_i \leftarrow$ SegementsGeneration(T_2, S_2);
21: $E_{\text{temp}} \leftarrow$ BuildEdges(Q_i, P_i, S_2) ;
22: $V2 \leftarrow V2 \cup [P_r, Q_r, \text{cost}_r] \cup [P_i, Q_i, \text{cost}_i]$;
23: $E2 \leftarrow E2 \cup E_{\text{temp}}$;
24: $\text{Cost}_{\text{Tmp}} = \text{cost}_{r_i} + \text{cost}_{r_f}$;
25: **if** $\text{Cost}_{\text{Tmp}} <$ FinalCost **then**
26: FinalCost $\leftarrow \text{Cost}_{\text{Tmp}}$;
27: FinalPath $\leftarrow$ Extractpath$(V_1, E_1, V_2, E_2, \text{d})$;
28: **end if**
29: **end if**
30: **end if**
31: Swap$([V_1, E_1], [V_2, E_2])$;
32: d $\leftarrow \bar{\text{d}}$;
33: counter $\leftarrow$ counter $+ 1$;
34: **end while**
35: **return** FinalPath, FinalCost;
36: **end procedure**

The steering method algorithm 2 would ensure that the nearest neighbor is found among the vertices of tree V (line 1) with a total cost to get to Q_r. Then, the trajectory S is made of time segments where each segment can be tested for joint-position limits (line 2) and eventually for collision. If a segment fails this test the new state is skipped and not added to the current neighbor tree.
The SegmentsGeneration() (line 14 and 20) method allows to generate a constant acceleration/jerk segments which their ends would also be added to sample-map with corresponding position and velocity (and acceleration for jerk steering). One of the main elements of RRT is the edges and that's what BuildEdges() is for (line 15 and 21). Where each element of E_{temp} has a constant acceleration/jerk segment and the duration, that could be used to build the complete path.
After checking the cost of the current random-state (or one of the corresponding solutions of the via-

Algorithm 2 Steering(Q_r, V, d)

1: $Q_{\mathrm{near}}, T, S, \mathrm{cost}_r \leftarrow$ NearestNeighbor(Q_r, V, d);
2: JointLimits $\leftarrow$ PositionLimitsViolation(S);
3: **return** T, S, cost$_r$, JointLimits;

point), An ExtractPath() method can be applied to the set of both trees edges (line 27).
To introduce both steering functions we need to define some notations:

- $(q_\star, \dot{q}_\star)$ position and velocity of one DoF at state $\star$;
- $(\ddot{q}_{\star_1}, \ddot{q}_{\star_2})$ acceleration values of a bang-bang profile;
- $(\dot{q}_{\max}, \ddot{q}_{\max}, \dddot{q}_{\max})$ velocity, acceleration and jerk limits of a DoF.

3.2.1 Double Integrator Minimum Time

The objective is to find a way to get from a state q_i to q_{i+1}, so this DIMT is based on bang-bang profile (2 segments) trajectory for each degree of freedom (DoF) separately, where the acceleration limits are used as velocity ramps. It is supposed to be the fastest straight-line motion possible [20]. This motion profile would transform to a trapezoidal one (3 segments) at each velocity limits violation.
Considering the velocity and acceleration limits of a 1 DOF mechanism, there are four extreme motion segments possible: the parabolas P^+ and P^- accelerating at maximum acceleration $\ddot{q}_{\max}$ and at minimum acceleration $-\ddot{q}_{\max}$, respectively, and the lines L^+ and L^- traveling with zero acceleration at maximum velocity $\dot{q}_{\max}$ and at minimum velocity $-\dot{q}_{\max}$, respectively. The sign of the first acceleration segment could be determined programmatically [14] using eq. (6):

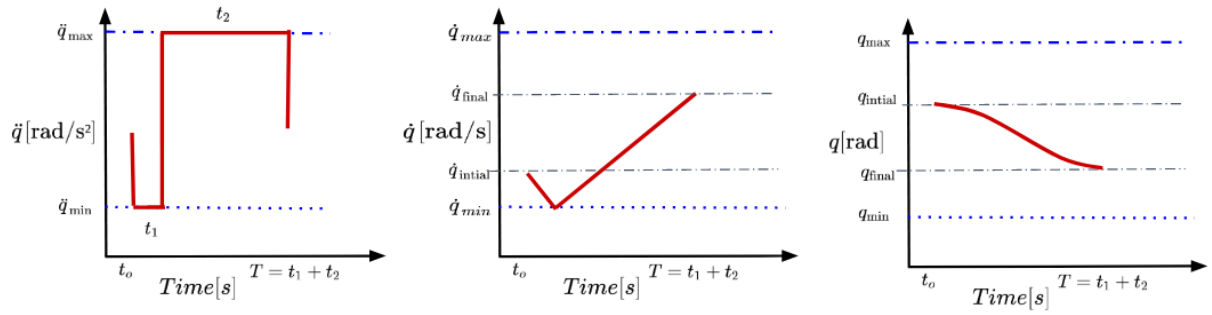
$$\sigma = \mathrm{sgn}(q_{i+1} - q_i - \delta q_{\mathrm{acc}}), \tag{6}$$

where δq_{acc} is distance crossed while accelerating at $\ddot{q}_{\max}$ defined as:

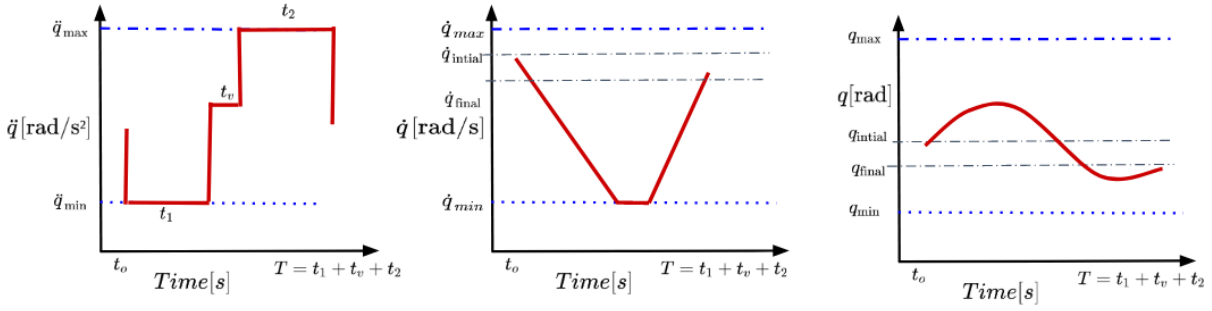
$$\delta q_{\mathrm{acc}} = \frac{1}{2}(\dot{q}_i + \dot{q}_{i+1})\frac{|\dot{q}_i - \dot{q}_{i+1}|}{\ddot{q}_{\max}}. \tag{7}$$

Thus we have:

$$\begin{aligned} \ddot{q}_{i_1} = -\ddot{q}_{i_2} &= \sigma \ddot{q}_{\max} \\ \dot{q}_{\mathrm{limit}} &= \sigma \dot{q}_{\max}. \end{aligned} \tag{8}$$



(a) In the case of extreme profile -two segments profile.



(b) Three segment profile .

Figure 10: Position-velocity-Acceleration profiles .

In order to determine the profile motion parameters, first, it's assumed that the motion is two segments profile, where t_1 and t_2 are to be defined as follows:

1. determine t_1 by solving the quadratic eq. (9)

$$\ddot{q}_{i_1} t_1^2 + 2\dot{q}_i t_1 + \frac{\dot{q}_{i+1}^2 - \dot{q}_i^2}{2\ddot{q}_{i_2}} - (q_{i+1} - q_i) = 0. \tag{9}$$

2. check for velocity limits violation,

$$\dot{q}_{t_1} = \ddot{q}_{i_1} t_1 + \dot{q}_i.$$

if $|\dot{q}_{t_1}| > \dot{q}_{\max}$ than we will have a trapezoidal profile with a constant velocity segment defined by the duration t_v:

$$t_1 = \frac{\dot{q}_{\text{limit}} - \dot{q}_i}{\ddot{q}_{i_1}} \tag{10}$$

$$t_v = \frac{\dot{q}_{i+1}^2 + \dot{q}_i^2 - 2\dot{q}_{\text{limit}}^2}{2\dot{q}_{\text{limit}}\ddot{q}_{i_1}} + \frac{q_{i+1} - q_i}{\dot{q}_{\text{limit}}}, \tag{11}$$

$$t_2 = \frac{\dot{q}_i - \dot{q}_{\text{limit}}}{\ddot{q}_{i_2}} \tag{12}$$

else:

$$t_2 = \frac{\dot{q}_{i+1} - \dot{q}_i}{\ddot{q}_{i_2}} + t_1.$$

3. the total time is $T = t_1 + t_2$ or $T = t_1 + t_v + t_2$ in case of constant velocity segment.

After calculating the extremal profile for each DoF with the duration T_i, where $i = \overline{1,n}$ and n is DoFs number, and Checking for blocked time interval for each one, the minimum-acceleration trajectory [14] [18] is applied then to synchronize T for all the DoFs by calculating new profiles parameters as follows:

$$T^2\ddot{q}_{i_1} + (2T(\dot{q}_{i+1} + \dot{q}_i) - 4(q_{i+1} - q_i))\ddot{q}_{i_1} - (\dot{q}_{i+1} - \dot{q}_i)^2 = 0, \tag{13}$$

the solution with greater absolute value is retained, then the rest of profile parameters are calculated as follows:

$$\ddot{q}_{i_2} = -\ddot{q}_{i_1} \tag{14}$$

$$t_1 = \frac{1}{2}\left(\frac{\dot{q}_{i+1} - \dot{q}_i}{\ddot{q}_{i_1}} + T\right) \tag{15}$$

$$t_2 = T - t_1 \tag{16}$$

if any velocity violation is present in the resultant profile than we calculate first the new

$$\dot{q}_{\text{limit}} = \text{sgn}(\ddot{q}_{i_1})\dot{q}_{\max},$$

then the new accelerations are calculated with eq. (17) and the duration with eq. (10),eq. (11) and eq. (12).

$$\ddot{q}_{i_1} = -\ddot{q}_{i_2} = \frac{(\dot{q}_{\text{limit}} - \dot{q}_i)^2 + (\dot{q}_{\text{limit}} - \dot{q}_{i+1})^2}{2(\dot{q}_{\text{limit}}T - (q_{i+1} - q_i))} \tag{17}$$

The profile produced is used to extract constant accelerations segments that would be used to check for position limits and collision violation, and form edges for the concerned tree if they are violations-free.

3.2.2 Jerk-limited trajectory generation

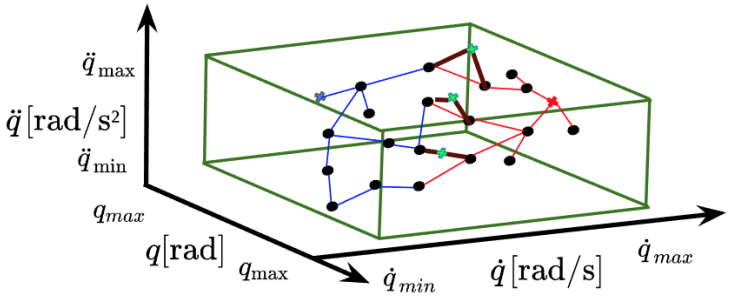


Figure 11: For the jerk-limited planner the states are chosen within the feasible set (green edged polytope) for each DoF, defined by velocity, position and acceleration limits.

The algorithm detailed in [15], is online, the resultant motion profile is an S-curve time scaling [20] that consists at maximum of seven segments. The approach goes mainly through these steps:

- an optional brake pre-trajectory where the DoF is brought to a safe kinematic state;
- For Each i-DoF, a valid extremal profile is established with a minimum duration $T_{i,\min}$;
- Check for the blocked time interval for each DoF;
- Synchronize arrival time for all the DoFs respecting the blocked time interval for each one.

3.2.3 Arrival-Time Synchronization and Infeasible Time

In order to obtain both the desired task-space pose with the desired velocity, all the DoFs have to reach their target states at the same time. It might appear that choosing the maximum of $T_{i,\min}$ would allow the robot to reach the desired state, but that's not true.
In [21], [14] and [15] this issue has been investigated for different profiles, blocked intervals a.k.a inoperative/ infeasible time intervals for each DoF, in which this latter can't reach the desired state in a desired duration. These blocked intervals are linked to the desired state, where a target state that has only position can be reached at each time with no blocked intervals. But for targets with velocity and acceleration desired states there will be at least one inoperative interval. The latter is always between two extreme profiles.
In order to determine the blocked interval [14] in this case, it comes to solve eq. (9) using the following values of the profile parameters:

$$\begin{aligned} \ddot{q}_{i_1} &= -\ddot{q}_{i_2} = -\sigma \ddot{q}_{\max} \\ \dot{q}_{\text{limit}} &= -\sigma \dot{q}_{\max} \end{aligned} \tag{18}$$

The smallest solution is the lower bound of the blocked interval and its upper one is defined by the biggest solution. If the velocity limits are violated than we proceed as explained before for the same case.

3.2.4 Trajectory Tracking

The trajectory tracking is formulated as a QP posture task [22]. Let the triplet $\{q_{\text{ref}}, \dot{q}_{\text{ref}}, \ddot{q}_{\text{ref}}\}_{\text{t}} \in \mathbb{R}^{3n}$ be position, velocity, acceleration references at t time-step within the generated trajectory, The posture task error is then defined as:

$$\eta = \begin{bmatrix} e \\ \dot{e} \end{bmatrix} = \begin{bmatrix} q - q_{\text{ref}} \\ \dot{q} - \dot{q}_{\text{ref}} \end{bmatrix}. \tag{19}$$

Described by its state-space dynamics as:

$$\dot{\eta} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \eta + \begin{bmatrix} 0 \\ I \end{bmatrix} \mu, \; \mu = \ddot{q} - \ddot{q}_{\text{ref}}. \tag{20}$$

Choosing control input μ in (20) as

$$\mu = -\begin{bmatrix} K_s & K_d \end{bmatrix} \eta, \tag{21}$$

with K_s, K_d are diagonal positive-definite that represent the stiffness and damping gains, the new acceleration is:

$$\ddot{q} = -\begin{bmatrix} K_s & K_d \end{bmatrix} \eta + \ddot{q}_{\text{ref}}, \tag{22}$$

Having as desired task acceleration $\ddot{q}_{\text{d}} = \ddot{q}_{\text{ref}} - K_s e - K_d \dot{e}$, the QP task formulation is defined as:

$$\min_{\ddot{q} \in \mathbb{R}^n} \|\ddot{q} - \ddot{q}_{\text{d}}\|^2 \tag{23a}$$

$$\text{s.t: } \mathcal{C}(\dot{q}, q)\ddot{q} + d \leq 0 \tag{23b}$$

where $\mathcal{C}(\dot{q}, q)$ is the constraints vector related to the robots limits (position, velocity, acceleration, torque, jerk...); the jerk is included in the constraints by deriving generated acceleration. When jerk based method is used, it is (jerk) the key stone to calculate the acceleration, velocity, and position references. Yet in the task formulation it's not explicitly written; d is the bounds values vector of all these constraints.

3.3 Ballistic Throws

3.3.1 Robot Tossing Workspace

To make reasonable tossing operations, one should investigate the toss reachable workspace of the used robot, so that targets should be inside the tossing reachable workspace (Fig. 12). Hence, an iterative algorithm (based on Monte Carlo) is used along with a ballistic motion model in order to highlight the limits that can be reached while throwing objects. At first, the reachable workspace is investigated, than the Tossing workspace is estimated from solving simple ballistic motion discarding any other post-impact phenomena. The surrounding volume is divided into horizontal layers of a specified jumping step. At each layer, the tossing-workspace is formed by a surface where all points have the same z.

Algorithm 3 Tossing-Workspace $(\text{Robot}, Z_{\max}, Z_{\min}, \text{step})$

1: $\text{RW}_{\text{space}} \leftarrow \text{ReachableWorkspace}(\text{Robot}, V, \text{d})$;
2: $\text{z} = Z_{\min}$;
3: **while** $\text{z} \leq Z_{\max}$ **do**
4: $\quad \text{TW}_{\text{space}} \leftarrow \text{TossingWorkspace}(\text{Robot}, \text{RW}_{\text{space}}, \text{z})$;
5: $\quad \text{z} = \text{z} + \text{step}$;
6: **end while**
7: **return** TW_{space};

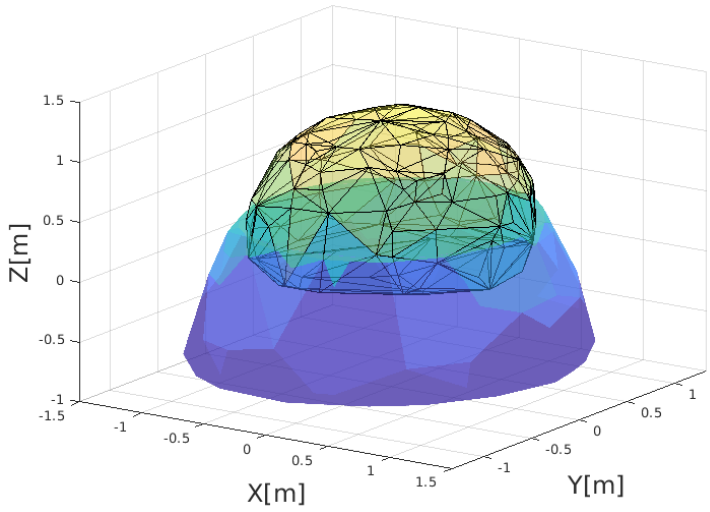


Figure 12: Panda Reachable Workspace (sphere-like volume with black edges) and Tossing Workspace, for $Z_{\max} = 0.5m$ and $Z_{\min} = -0.7m$.

3.3.2 Ballistic Optimization

The application would include picking an object from a known position and tossing it from a release point so it would reach a desired final state by performing a ballistic motion. But this doesn't limit the approach to only this since it's task-independent.
The release state is determined by a ballistic optimization process considering the robot's capabilities to generate a feasible one. This latter plays the role of the via-point in our planning approach. Other phenomena could be modeled in this optimization as aerodynamics and bouncing.

we define (P_d, quat_d) as the desired position and quaternion orientation of the box and $(P_{t_f}, \text{quat}_{t_f}) \in \mathbb{R}^7$ the ones at time-step t_f of the ballistic motion defined by:

$$P_{t_f} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} v_x t_f + x_o \\ v_y t_f + y_o \\ -g t_f^2 + v_z t_f + z_o \end{bmatrix}, \tag{24}$$

and

$$\begin{aligned} \text{quat}_{t_f} &= \int_0^{t_f} \frac{1}{2} \mathcal{Q}(\omega) * \text{quat}\, dt \\ \dot{\omega} &= \mathcal{I}^{-1}\tau - \mathcal{I}^{-1}(\omega \times \mathcal{I}\omega) \end{aligned} \tag{25}$$

where $\mathcal{Q}(\omega)$ is a skew-matrix defined as:

$$Q(w) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \tag{26}$$

where $[x_o, y_o, z_o] \in \mathbb{R}^3$ is position coordinates of the COM of the object at the moment of release, $(v, \omega) \in \mathbb{R}^6$ are the linear and angular velocities of the end-effector that would be inherited by the picked object (under the assumption of rigid connection between bodies), $\mathcal{I} \in \mathbb{R}^{3\times3}$ is the object inertia matrix and $\tau \in \mathbb{R}^3$ is the external torques for a free ballistic motion it's assumed to be absent ($\tau = 0_{3\times1}$). Considering now that quaternion $\text{quat}_* = \{\nu_*, \epsilon_*\}$, we can express e_O, the orientation error [23], as:

$$e_\text{O} = \Im[\text{quat}_\text{d} \ominus \text{quat}_{t_f}^{-1}] = \nu_{t_f} \cdot \epsilon_\text{d} - \nu_\text{d} \cdot \epsilon_{t_f} + \epsilon_\text{d} \times \epsilon_{t_f} \tag{27}$$

The problem can be formulated as a weighted optimization as follows:

$$\min_{q,\dot{q}} \; w_1 \left\| P_{t_f} - P_d \right\|^2 + w_2 \left\| e_\text{O} \right\|^2 + w_3 \left\| v_\text{desired} - v_\text{impact} \right\|^2 \tag{28a}$$

$$\text{s.t: } \dot{q}_\text{min} \leq \dot{q} \leq \dot{q}_\text{max} \tag{28b}$$

$$q_\text{min} \leq q \leq q_\text{max} \tag{28c}$$

$$[v, \omega] = J(q)\dot{q} \tag{28d}$$

$$\left\| v \right\|^2 \leq v_\text{max} \tag{28e}$$

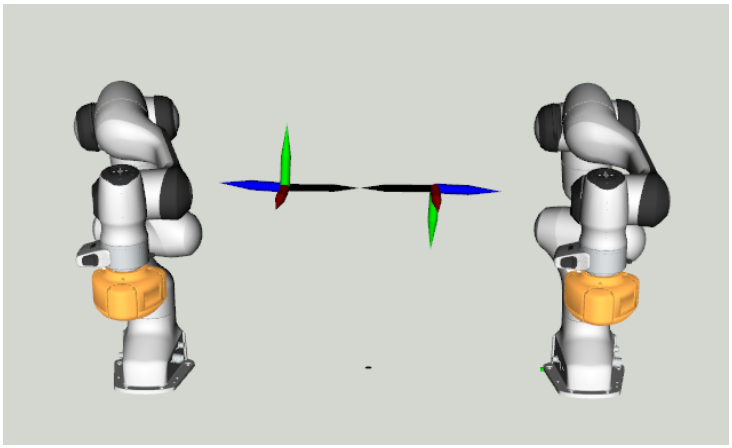
$$\left\| \omega \right\|^2 \leq \omega_\text{max} \tag{28f}$$

$$\tag{28g}$$

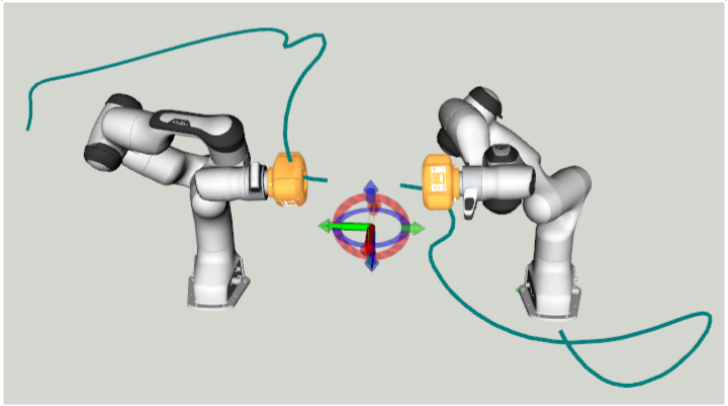
where $(q, \dot{q}) \in \mathbb{R}^{2n}$ are robot's joint position and velocity with n as the number of DoFs, J is the Jacobian matrix, and w_i the associated weight to each term.
The object impact velocity v_impact is considered in the ballistic optimization, so the generated tossing posture would induce a desired impact velocity v_desired at the arrival time t_f at the end of ballistic motion trajectory.

3.4 Impact Grabs

In this section we extend this planner to handle grabbing with impact, and also for dual robots systems. The planner is asked to find feasible collision-free configuration paths for both robots in order grab by means of a dual-arm robotic system an object characterized by its desired impacting position and velocity vectors Fig. 13; each impacting contact position is within the reachable workspace of each designated robot.
The planner would expect that both position and rotation are known for both robots, and since the impact-planning is executed offline, all the obstacles to be avoided are included in the scene.
The problem can be handled in many ways, and since we tend to optimize the running time of a robot and always minimize trajectory duration, the synchronization would be handled during the launching of paths-to-impact of both robots. As it can be done to optimize the overall running time of both robots which would push one of them to operate in slower pattern than the other one (generally farther configurations would maximize time-to-arrival of the generated paths).



(a) Dual panda grabbing scenario, with a nonzero zero impact-grab velocity (black arrows).



(b) Dual panda grabbing scenario, after executing the planned trajectories (blue lines).

Figure 13: Dual panda grabbing scenario.

As this planner can handle both non zero-velocity start and final configuration, many configurations can

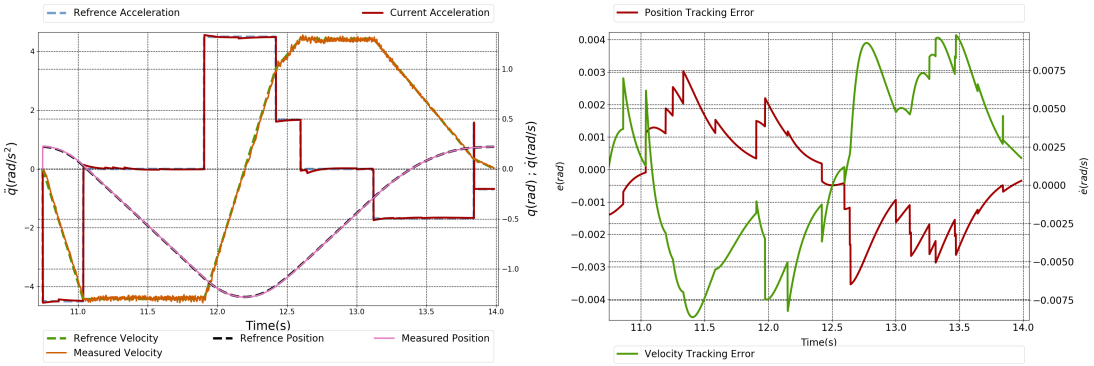
be put into cascade to form a full defined series of tasks, i.e dual-grab, dual-toss or dual-place, dual-re-grab.

3.4.1 Synchronizing dual robots for grabbing

As already said, synchronizing both robots could be done either on the planning level, by adding time constraints on the planned path for each robot to match the arrival time for both of them , or on control level, since the we are using same tracking strategy in section 3.2.4, by introducing the posture task of the fastest robot with a delay that corresponds to time difference between the planned paths.

3.5 Real-Robot Experiments

The experiments on the tossing framework are conducted on the 7 DOF Panda Robot from Franka-Emika with an attached suction cup as a picking tool. The picking process is also handled as a via point. The object at the beginning of the trajectory tracking is considered rigidly attached to the suction pump. The robot's both reachable and tossing workspace are estimated to provide a reasonable testing target and enhance the role of tossing on the whole workspace volume, see Fig. 12.
The generated trajectories by algorithm 1 are passed to a trajectory tracking task (section 3.2.4) implemented on the open-source QP controller `mc_rtc`[5] that runs at 1 kHz frequency using the necessary dependencies [4] `mc_franka`[6] and `mc_panda`[7] corresponding to the robot used in the trials.



(a) Position-velocity-Acceleration tracking for joint 1.

(b) the corresponding errors between measured and reference position and velocity .

Figure 14: Position-velocity-Acceleration tracking.

The trajectory is well tracked as shown in Fig. 14 where we have a limited position and velocity error ($|e_{\text{position}}| \leq 4.10^{-3}$rad, $|e_{\text{velocity}}| \leq 5.10^{-3}rad/s$) for all joints.

[5] `https://jrl-umi3218.github.io/mc_rtc/index.html`
[6] `https://github.com/jrl-umi3218/mc_franka`
[7] `https://github.com/jrl-umi3218/mc_panda`

4 CONCLUSION

This report describes two different proposed solutions to generate impact-postures or more precisely, impact plans. One is data-driven based on learning to find a suitable release state, which would be handled by a Dynamical System to generate a trajectory to this point, see also deliverable D5.3. The other would rather plan a trajectory that includes this configuration as via point to ensure safety return to a rest configuration or as a finale impact-posture.
Furthermore we continued by applying these solutions on different use-cases:

1. generate a tossing-configuration using one robot that picks an object from a known position and throw it to a desired target pose in the tossing reachable workspace with both Learning-DS and Planning based methods;

2. generate impact-configurations for grabbing task using a dual arm-robots based on planning.

The Data-Driven posture generator is integrated along with the Dynamical System from EPFL, in the Robot-Toss-Scenario, to generate acceleration references that are fed to a mc_rtc `mc_rtc` robot QP controller.
The Model-based Planning posture generator defines full paths with a desired impact task at a known step time and it was applied on both Robot-Toss-Scenario and Dual-Grab-Robots.
However, up to now, the planning approach is not real-time and this is problematic in many ways. We shall be able to plan fast so as to estimate the objects inertia on-the-fly using knowledge from WP3 and subsequently plan the right trajectories subsequent to each object in the tossing case scenario. We shall also integrate the suction cup in the planning part for the tossing or boxing, and this is clearly an open challenge.

5 REFERENCES

[1] S. Brossette, A. Escande, J. Vaillant, F. Keith, T. Moulard, and A. Kheddar, "Integration of non-inclusive contacts in posture generation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 933–938.

[2] A. Escande, S. Brossette, and A. Kheddar, "Parametrization of catmull-clark subdivision surfaces for posture generation," in *2016 IEEE International Conference on Robotics and Automation*, 2016, pp. 1608–1614.

[3] S. Brossette, A. Escande, and A. Kheddar, "Multicontact postures computation on manifolds," *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1252–1265, 2018.

[4] N. Dehio and A. Kheddar, "Robot-safe impacts with soft contacts based on learned deformations," in *IEEE Int. Conf. on Robotics and Automation*, 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02973947

[5] T. Bretl and S. Lall, "Testing static equilibrium for legged robots," *IEEE Transactions on Robotics*, vol. 24, no. 4, pp. 794–807, 2008.

[6] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[8] G. Schwarz, "Estimating the Dimension of a Model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461 – 464, 1978.

[9] S. S. M. Salehian, M. Khoramshahi, and A. Billard, "A dynamical system approach for softly catching a flying object: Theory and experiment," *IEEE Transactions on Robotics*, vol. 32, no. 2, pp. 462–471, 2016.

[10] K. Bouyarmane, K. Chappellet, J. Vaillant, and A. Kheddar, "Quadratic programming for multirobot and task-space force control," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 64–77, 2019.

[11] H. Pham and Q.-C. Pham, "Critically fast pick-and-place with suction cups," *IEEE Int. Conf. on Robotics and Automation*, pp. 3045–3051, 2019.

[12] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "TossingBot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[13] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning." in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001.

[14] T. Kunz and M. Stilman, "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits," in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 3713–3719.

[15] L. Berscheid and T. Kröger, “Jerk-limited real-time trajectory generation with arbitrary target states,” in *Robotics: Science and Systems*, 2021.

[16] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[17] P. Lertkultanon, “Planning algorithms for complex manipulation task,” Ph.D. dissertation, Nanyang Technological University, Singapore, 2017.

[18] C. Lau and K. Byl, “Smooth RRT-connect: An extension of RRT-connect for practical use in robots,” in *IEEE International Conference on Technologies for Practical Robot Applications*, 2015, pp. 1–7.

[19] K. Hauser and V. Ng-Thow-Hing, “Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts,” in *IEEE international conference on robotics and automation*, 2010, pp. 2493–2498.

[20] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. USA: Cambridge University Press, 2017.

[21] T. Kröger and F. M. Wahl, “Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events,” *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94–111, 2010.

[22] K. Bouyarmane and A. Kheddar, “On weight-prioritized multitask control of humanoid robots,” *IEEE Transactions on Automatic Control*, vol. 63, no. 6, pp. 1632–1647, 2017.

[23] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2009, ch. Differential kinematics and statics, pp. 105–160.