FrankaSim: A Dynamic Simulator for the Franka Emika Robot with Visual-Servoing Enabled Capabilities

Alexander Antonio Oliva, Fabien Spindler, Paolo Robuffo Giordano, François Chaumette

FrankaSim: A Dynamic Simulator for the Franka Emika Robot with Visual-Servoing Enabled Capabilities

Alexander Antonio Oliva[1], Fabien Spindler[1], Paolo Robuffo Giordano[2] and François Chaumette[1]

***Abstract*— We present in this paper a new open-source simulator based on CoppeliaSim and ROS for the popular Franka Emika Robot (FER) fully integrated in the ViSP ecosystem, a powerful library for Visual-Servoing. The simulator features a dynamic model that has been accurately identified from a real robot, leading to more realistic simulations. The C++ API closely follows the ViSP class of the real robot allowing to narrow the gap between simulation code and real control software deployment. Conceived as a multipurpose research simulation platform, it is well suited for visual servoing applications as well as, in general, for any pedagogical purpose in robotics. All the software, models and CoppeliaSim scenes presented in this work are publicly available under free GPL-2.0 license.**

I. Introduction

Software simulators are valuable proof-of-concept tools for validating methods, theories and ideas in a simple and cost-effective way. They are useful for testing and validating ideas and algorithms that would otherwise be hard to implement due to limited resources or restrictions [1]. For instance, robot simulators have recently been used to generate massive amounts of training data for training neural networks allowing to reach training levels that would be impossible with data collected on real hardware [2], [3].

Any robotic simulator must be able to simulate different kinds of robots, actuators and sensors with enough accuracy for narrowing the reality gap. The most common robotic system that is featured by virtually any robotic simulator is a manipulator arm for tasks involving motion control, pick-and-place, interaction with the environment, and so forth. An accurate dynamic modeling of a manipulator is of paramount importance in relation to problems of motion simulation, analysis of manipulation structures and control algorithms. Simulating the motion of a manipulator allows indeed to safely test control strategies and trajectory planning techniques in a low-risk environment without the need to refer to a physically available platform. Modern robotics simulators are flexible and dynamic and can handle complex physics simulations as well as a variety of sensors that a user may need to assess. In this respect, the goal of this work is to propose a *high-fidelity* and *openly available* simulator for a popular manipulator arm, with in addition, as case study, the integration and exploitation of the simulator for achieving Visual Servoing (VS) tasks [4]. Indeed, VS is a widely known set of techniques for controlling the motion of a robot from visual input provided by one or more cameras, and in this paper we show how an accurate dynamical simulation of the FER can be exploited to perform a realistic and non-trivial visual servoing schemes. In particular, Image-Based Visual-Servoing (IBVS) refers to the use of *visual features* directly extracted in the image plane as control data. In this approach, the visual error is minimized in the image plane, making the Cartesian space trajectories followed by a mounted camera unpredictable [4]. Another critical aspect in VS is the placement of the camera that needs to be appropriately designed to avoid possible occlusions preventing the extraction of such visual features.

The robot considered in this work is the popular Franka Emika Robot. It is a torque-controlled cobot widely used both in research and small-sized business because of its safety, versatility, reliability and relatively low price. Its total weight is about $18\,\mathrm{kg}$ and it is capable of handling payloads up to $3\,\mathrm{kg}$. It is a redundant manipulator with $n = 7$ revolute joints, each of which is equipped with a link-side torque sensor. This robot can be controlled with five different interfaces: at joint level by sending torque, position or velocity commands or by sending Cartesian pose or velocity commands to the Franka Control Interface (FCI) through the *libfranka*, its open-source C++ API. Through this library it is possible to send real-time control values at 1 kHz and have full access to the robot state. Robots with the Research Interface can also be controlled using the franka_ros metapackage [5], which integrates and exposes the complete *libfranka* API to the ROS ecosystem. This metapackage also provides hardware abstraction and model description in Universal Robotic Description Format (URDF) to simulate the robot in Gazebo [6].

II. Related Works

As far as we know, the manufacturer never released the official dynamic parameters of the robot, nor identified parameters were publicly available before the recent identification results reported in [7]. To overcome this limitation, a first attempt was made in [8] by taking the original URDF description files and the robot meshes from the official repository and adding the missing gazebo-specific tags and inertial parameters (mass, center of mass and inertia tensor) recovered from a CAD-based software. Once the identified dynamic parameters became available, some researchers started building their own simulators from the same repository and tweaking the URDFs [9], [10] to achieve more realistic simulations both in Gazebo and MuJoCo [11] as well. In [12], the authors built a FER simulation platform using Simscape Multibody MATLAB Toolbox based on the kinematic, dynamic and friction models identified in [7].

[1]Inria, Univ Rennes, CNRS, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France name.surname@inria.fr

[2]CNRS, Univ Rennes, Inria, IRISA, Rennes, France, prg@irisa.fr

Simscape is neither open nor free, and the simulation code developed cannot be transferred to the real implementation.

As reported in the surveyed literature in [1], the most important criteria for choosing a simulator are: **small reality gap** (32%), **to be open-source** (24%), **light & fast** (11%), **simulation-to-real code transfer** (9%), **customization** (6%), **others** (5%), **no inter-penetration between bodies** (3%). Despite Gazebo is the most used simulator in research, CoppeliaSim (former V-REP) [13] scores the best in the previous criteria [1] while [14] pointed out that it offers a number of useful features, such as multiple physics engines (Bullet, ODE, Vortex and Newton), a comprehensive model library, the ability of a user to interact with the world during simulation and, most importantly, mesh manipulation and optimisation. Moreover, it automatically spawns new threads on multiple CPU cores and therefore utilises the full amount of CPU power when necessary. It is therefore suitable for high-precision modelling although it is the most demanding in terms of resources among the compared simulators. In the reality gap analysis conducted in [15] on a grasping task, CoppeliaSim with Newton and Vortex physics engines performed 1st and 3rd respectively among the compared simulators (Pybullet [16], CoppeliaSim and MuJoCo).

Most of the reviewed works are based on Gazebo, which has some limitations compared to CoppeliaSim which, despite its higher demand on resources, is more accurate and faithful to reality. To the best of our knowledge, the proposed simulator is the only dynamic simulator supporting CoppeliaSim. Besides offering the characteristics of other Franka Emika Robot simulators in terms of kinematics, dynamic model parameters and ROS integration, this simulator further offers full integration within the ViSP library [17], enabling it with VS capabilities. There exist some other libraries allowing for rapid prototyping of VS applications [18], [19], but ViSP is likely to be the most complete, open-source, hardware independent, simple and portable one. The above mentioned VS libraries, including ViSP, have very limited visualisation capabilities when simulating VS tasks, and this is another aspect the proposed simulator aims to overcome.

FrankaSim features a Lagrangian Dynamic model library that can provide the Coriolis matrix of the manipulator and the estimated joint dynamic friction, unlike dynamic models based on standard Newton-Euler algorithms. Furthermore, the FER is a redundant manipulator that allows for full 3D space control along/about all the Cartesian directions, unlike for the five degrees of freedom (DoF) robot proposed in [19]. Finally, being the code of the simulator implemented as a replica of the ViSP class that wraps the *libfranka*, it greatly facilitates the possibility to transfer verified code to the real robot. This simulator lends itself well to simulate Dynamic Visual-Servoing [20], [21] or Vision-Force control schemes [22], [23], [24], relieving the user from the burden of implementing his/her own simulation platform.

The rest of the paper is organized as follows: in Sec. III the FrankaSim simulator is presented detailing all its parts and then giving an architectural overview. In Sec. IV we present one experiment that illustrates the small gap between real and simulated robot and a second that highlights some of the salient features offered by the simulator. Sec. V states some conclusions and outlines future developments.

III. FrankaSim

The simulator proposed in this work, is a co-simulator that includes several conveniently integrated components allowing for rapid prototyping and testing of manipulator controllers. It is mainly based on ViSP *classes* that provides a bulk of utilities such as native support to linear algebra and transformation matrices, among others, greatly simplifying both the simulator implementation itself as well as the deployment of user defined controllers. ViSP further extends the simulator control skills by enabling it with visual servoing control capabilities. A physical simulation is performed in CoppeliaSim, in which one (or more) instance(s) of a FER with its identified dynamic model parameters [7] can be simultaneously controlled. The communication between CoppeliaSim and the C++ instance of a simulated robot takes place through ROS. All the software presented in this paper, the CoppeliaSim scenes and models as well as different experiments, for both the simulated and real robot, are available on `https://github.com/lagadic/visp_ros` while step-by-step tutorials can be found in the ROS wiki `https://wiki.ros.org/visp_ros`.

A high-level view of the architecture and operation of the simulator software will be provided below, after having detailed the various components.

A. Kinematics

A manipulator can be mechanically described as a kinematic chain consisting of rigid bodies (links) connected by means of revolute or prismatic joints, which constitute the DoF of the structure. One end of the chain is bounded to a base (or the floor) while at the other extreme a tool is usually attached (see Fig. 1). The overall motion of the structure is achieved through the composition of elementary motions of each link with respect to the previous one due to the motion of the $\boldsymbol{q}_i$-th joint ($\forall i = 1, ..., n$). In order to manipulate the tool end point (end-effector), the description of the position ($^f\boldsymbol{p}_e$) and orientation ($^f\boldsymbol{\varphi}_e$) of such point is required. This is referred to as direct kinematic problem $^f\boldsymbol{M}_e(^f\boldsymbol{p}_e, ^f\boldsymbol{\varphi}_e) = \mathcal{K}(\boldsymbol{q})$, being $^f\boldsymbol{M}_e \in SE(3) = \mathbb{R}^3 \times SO(3)$ the homogeneous transformation matrix representing the pose of the end-effector frame (Σ_e) expressed in floor frame (Σ_f):

$$^f\boldsymbol{M}_e(^f\boldsymbol{p}_e, ^f\boldsymbol{\varphi}_e) = \left[\begin{array}{c|c} ^f\boldsymbol{R}_e(^f\boldsymbol{\varphi}_e) & ^f\boldsymbol{p}_e \\ \hline 0\,0\,0 & 1 \end{array}\right]$$

In this subsection, under the term *kinematics* we encompass either direct, inverse and differential kinematics, being the inverse kinematics the problem of retrieving the joint configuration vector from a given end-effector pose $\boldsymbol{q} = \mathcal{K}^{-1}(^f\boldsymbol{M}_e)$, while the differential kinematics characterizes the link between the joint velocities $\dot{\boldsymbol{q}}$ and the corresponding linear ($^f\dot{\boldsymbol{p}}_e \in \mathbb{R}^3$) and angular ($^f\boldsymbol{\omega}_e \in \mathbb{R}^3$) velocities of the end-effector. This relationship is described by a configuration dependent transformation matrix named, the geometric

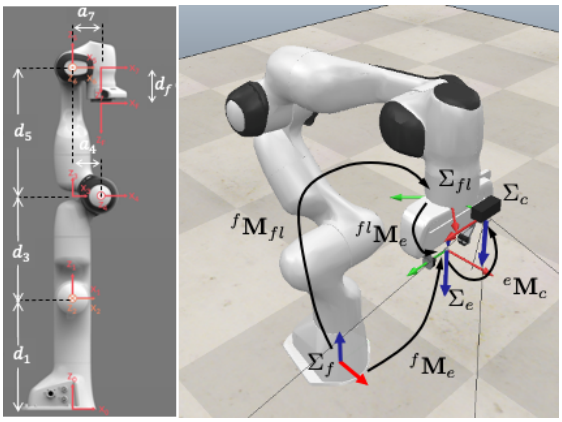

Fig. 1: Left: FER with its kinematic parameters according to the modified Denavit-Hartenberg convention: $d_1 = 0.333$ m, $d_3 = 0.316$ m, $d_5 = 0.384$ m, $d_f = 0.107$ m, $a_4 = 0.0825$ m, $a_5 = -a_4$ m, $a_7 = 0.088$ m. Right: An instance of the FER model in CoppeliaSim with its gripper and a wrist mounted camera. The floor Σ_f, flange Σ_{fl}, end-effector Σ_e and camera Σ_c frames are depicted as well as some transformation matrices between them.

Jacobian ($\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$) [25], and expressed as

$$ {}^f\mathbf{v}_e = \begin{bmatrix} {}^f\dot{\boldsymbol{p}}_e \\ {}^f\boldsymbol{\omega}_e \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} $$

The FER is kinematically described according to the modified Denavit-Hartenberg convention and its parameters are reported in Table I and Fig. 1.

TABLE I: Denavit-Hartenberg parameters.

i	1	2	3	4	5	6	7	8
a_i	0	0	0	a_4	a_5	0	a_7	0
α_i	0	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	0
d_i	d_1	0	d_3	0	d_5	0	0	d_f
θ_i	q_1	q_2	q_3	q_4	q_5	q_6	q_7	0

In FrankaSim, we delegate the issue of dealing with kinematics to the well-known kinematics and dynamics library OROCOS_KDL [26]. The Orocos project aims at providing kinematic and dynamic code usable in real time; it contains code for rigid body kinematic representation and calculations for structures and their direct and inverse kinematic solvers.

B. Dynamics

The dynamic model of a manipulator provides a description of the relationship between the actuation torques at the joints and the structure motion. With the Lagrange formulation, the equation of motion can be derived with an approach independent of the coordinate frame of reference. The dynamic model of a manipulator can be written as:

$$ \boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_f(\dot{\boldsymbol{q}}) + \boldsymbol{J}^\top \boldsymbol{h}_e = \boldsymbol{\tau} \qquad (1) $$

with $\boldsymbol{B}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ being the symmetric and positive definite inertia matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$ the matrix of centrifugal and Coriolis effects, $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$ the configuration dependent vector of gravitational forces, and $\boldsymbol{\tau}_f(\dot{\boldsymbol{q}}) \in \mathbb{R}^n$ the vector of joint torques due to friction effects. Finally, $\boldsymbol{\tau} \in \mathbb{R}^n$ is the vector of joint actuation torques and $\boldsymbol{h}_e \in \mathbb{R}^6$ is the external *wrench* acting on the end-effector frame.

In a previous work [7], an accurate dynamic model of the robot has been identified and a set of feasible dynamic parameters retrieved. These parameters have been adopted in the MATLAB Robotics toolbox[1] [27] alongside many hand-made URDFs of other simulators [9], [10] and, more recently, were also included in the official URDF model provided by the manufacturer, making *de facto* the dynamic model identified in [7] the standard model.

From the identified dynamic parameters in [7], the authors released both MATLAB and C++ libraries as open-source software[2] under GPLv3 license and a CoppeliaSim model of the FER (arm only), see Fig. 1. These libraries provide an estimate of the dynamic terms of the Lagrangian model (1) ($\hat{\boldsymbol{B}}(\boldsymbol{q}), \hat{\boldsymbol{C}}(\boldsymbol{q},\dot{\boldsymbol{q}}), \hat{\boldsymbol{g}}(\boldsymbol{q})$) computed from the symbolic expressions of the equation of motion and identified parameters as well as the joint viscous friction $\hat{\boldsymbol{F}}_v(\dot{\boldsymbol{q}})$. The main limitation of this library is that its API was not parameterized w.r.t. the attached payload; thus, if a payload is attached to the robot, such as the Franka Emika Hand, for which we have provided a model for CoppeliaSim within the simulator package[3] (see Fig 2), the library will not be able to account for it.

In this work we have further extended this library, parameterizing it with respect to both the payload parameters, as reported in [28], and the gravitational acceleration vector. In this way the library can provide the same features offered by Newton-Euler (N-E) algorithms, *i.e.* adding payloads to the chain or mounting the robot in a position different from vertical, while providing the full Coriolis matrix and the viscous friction terms. This constitutes an advantage w.r.t. standard N-E algorithms since they are not able to provide neither the Coriolis matrix $\hat{\boldsymbol{C}}(\boldsymbol{q},\dot{\boldsymbol{q}})$ -or any of its columns- nor the mixed velocity term $\hat{\boldsymbol{C}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r$, with $\dot{\boldsymbol{q}}_r$ a reference velocity different from the one used to compute $\hat{\boldsymbol{C}}(\boldsymbol{q},\dot{\boldsymbol{q}})$ [29], but only the resulting vector $\hat{\boldsymbol{c}}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \hat{\boldsymbol{C}}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$. These matrix is needed, for instance, to compute the momentum observer and evaluate the residual vector for collision detection and safe reaction applications [30] or to implement passivity-based trajectory tracking control laws [31]. Although [29] has proposed a modified N-E algorithm to overcome the aforementioned limitation of the standard N-E method, such solution is not implemented in OROCOS_KDL nor the full Coriolis matrix is provided by the *libfranka*. This motivates the use of our library both to avoid implementing this method in OROCOS_KDL from scratch and because this library can be easily integrated within the code of the real robot controller without having the dependency of the KDL library.

To add a tool to the real robot, one has to use its Desk web interface and specify both the new pose of the end-effector in flange frame (Σ_{fl}) and the dynamic parameters. Fig. 2 reports those values for the real gripper. Due to

[1] https://github.com/petercorke/robotics-toolbox-matlab/blob/master/models/mdl_panda.m

[2] https://github.com/marcocognetti/FrankaEmikaPandaDynModel

[3] The Franka Emika Hand is also available from CoppeliaSim V4.3.0

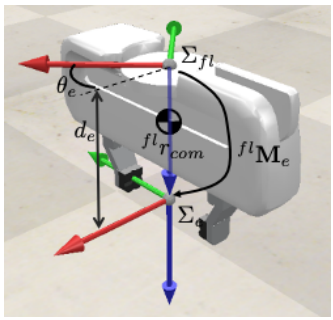

Fig. 2: CoppeliaSim model of the Franka Emika Hand. End-effector position ${}^{fl}\boldsymbol{r}_e = [0; 0; d_e]^\top\,[m]$ and orientation ${}^{fl}\boldsymbol{R}_e = R_z(\theta_e)$ and CoM position ${}^{fl}\boldsymbol{r}_{CoM} = [-0.01; 0; 0.03]^\top\,[m]$ in flange frame. Mass $m_L = 0.73\,[kg]$ and inertia tensor $\boldsymbol{I_L} = diag(0.001; 0.0025; 0.0017)\,[kg.m^2]$. $\theta_e = -45°$, $d_e = 0.1034[m]$.

the possibility of simulation malfunction with some physics engines, particularly Bullet and ODE, when the difference between connected masses is too large, it is preferable to distribute the total mass of the gripper uniformly among the fingers and the body in order to avoid "strange" behavior. By doing so, the total mass will stay the same but the inertia tensor and the location of the center of mass of the overall gripper will differ from the real one. This is not an issue since, as we will see in Section III-E, we automatically gather this information from CoppeliaSim and send it to the robot through ROS.

C. ViSP

ViSP[4] or **Vi**sual **S**ervoing **P**latform, is a modular C++ library developed and maintained by the Rainbow team (former Lagadic) at Inria/IRISA Rennes, France, that allows for fast development of visual servoing applications. This library was designed to be hardware independent, simple, portable and easily extendable. Furthermore, it features a large class of elementary tasks with various visual features ranging from image points, to image lines and moments, 3D pose estimation and so forth, as well as providing native support for linear algebra operations, visual trackers, plotters, and of course, VS controllers.

The ViSP's *vpRobotFranka* class is a wrap over the *libfranka* robot API. This is a multi-thread implementation that runs the robot control routine in a separated thread at the control rate (1 kHz) allowing the main program to continue its own execution; this means that from the same *main()*, one can read and process information coming from different sensors, like cameras and force/torque sensors, even at slower sampling rates. This implementation allows to simultaneously control more than one robot at a time.

D. Visp_ros

Visp_ros is a basket of generic ROS nodes that are based on the ViSP library and exposes it to the ROS ecosystem. It includes a C++ library of classes that allow ROS to be incorporated transparently and as conventional ViSP classes without the need to develop ROS-specific code. The nodes developed in Visp_ros exploit all the potential offered by both ViSP and ROS.

FrankaSim is developed as part of Visp_ros. The simulator's *vpRobotFrankaSim* class replicates the *vpRobotFranka* class present in ViSP for the real robot, providing its same methods, and extending it with few others in order to keep the same interface and behavior on both the simulator and with the real robot. The ROS-specific code is implemented in the *vpROSRobotFrankaCoppeliasim* class which inherits from class *vpRobotFrankaSim*. It also contains CoppeliaSim-specific callbacks -passing through ROS- to manage e.g. the synchronization between CoppeliaSim and the C++ simulation code or the setting of the control mode. This is also a multi-threaded implementation featuring, other than the *main()* process in which the robot instance is declared, a "reading" thread, *listening* at the topic containing the robot state $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau})$, and a "writing" one, that continuously publishes the joint velocity $\dot{\boldsymbol{q}}_{cmd}$ or torque $\boldsymbol{\tau}_{cmd}$ commands depending on the selected control mode. Specific methods to get the simulation time from CoppeliaSim, for adding a tool, e.g. the Franka Emika Hand, as well as internal flags to select a synchronous or asynchronous simulation further extend the set of methods of the *vpROSRobotFrankaCoppeliasim* class. Preserving the same interface for the simulator classes compared to the one that controls the real robot allows the reusability of the code.

E. CoppeliaSim

CoppeliaSim is a versatile and scalable simulation framework offering a multitude of different programming techniques for the controllers, and allows to embed control functionalities in simulation models easing the programmers effort and reducing the deployment complexity. It integrates four free physics engines (Bullet 2.78 & 2.83, ODE, Newton) and a more precise closed-source engine (Vortex), for which it is possible to obtain a free non-commercial license. It also provides a large variety of ready to use sensors and support scripting via Lua. Using a Lua script attached to the robot, we have automated some procedures such as finding the dynamic and kinematic parameters of the payload (gripper), retrieve the absolute gravity vector in base frame or the choice of the control mode.

Thanks to the RosInterfaceHelper Lua script, it is possible to programmatically start/stop the CoppeliaSim scene simulation as well as to synchronize each simulation step between the C++ code and the physical simulation. In order to make the synchronization more transparent to the user, the triggers were embedded in the *vpROSRobotFrankaCoppeliasim* class, and a flag has to be selected to enable it.

Within the simulator we deployed the scenes of the experiments, the model of the Franka Emika Hand gripper (see Fig. 2) and some AprilTags.

F. Software Architecture

A block diagram of the co-simulation environment with some of the C++ classes and dependencies in a UML-like

[4]https://visp.inria.fr

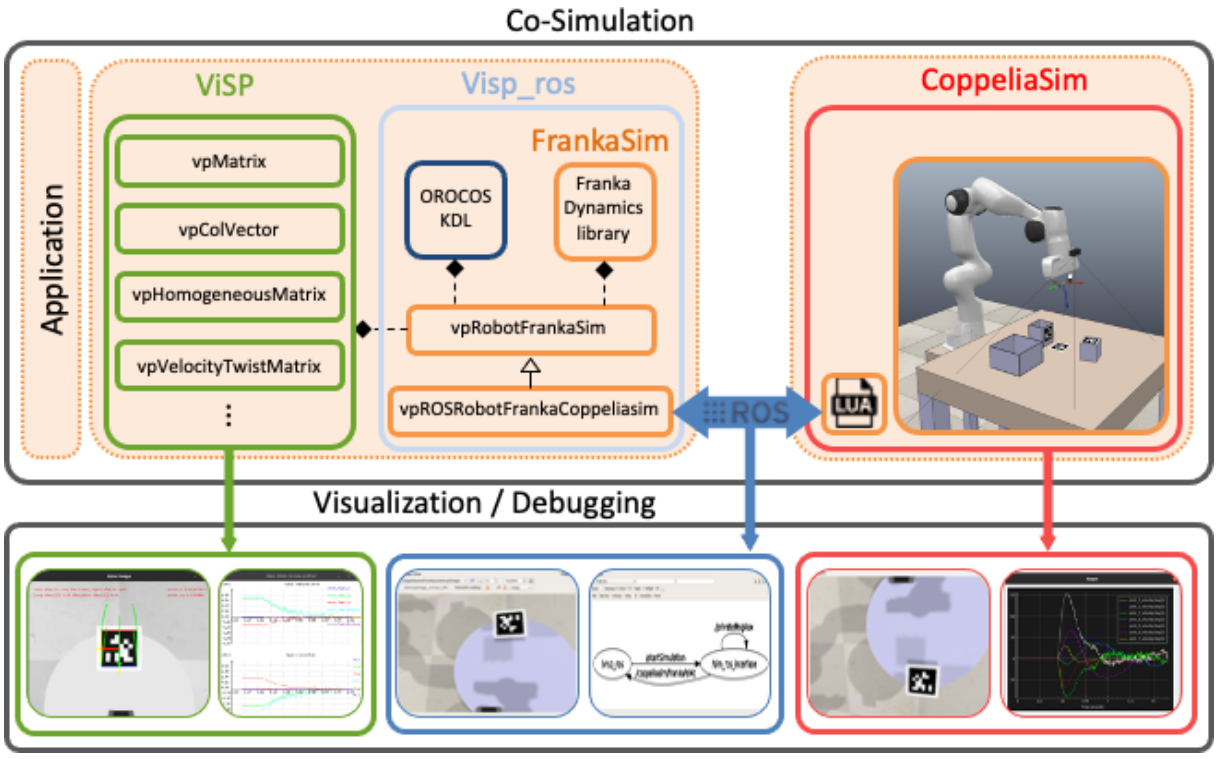

Fig. 3: Block diagram of the main components of the FrankaSim architecture. An UML-like diagram highlights the principal class relationships while a wide range of visualization and debugging possibilities at ViSP (green), ROS (blue) and CoppeliaSim (red) levels are depicted.

diagram is shown in Fig. 3. The diagram is complemented with some data visualization utilities provided by the various software components that help the user in analyzing the system performance and/or debugging.

A simulation is built as a ROS node that benefits from both FrankaSim and ViSP capabilities to implement one or multiple FER controllers. The scene rendering and the physics engine are supported by CoppeliaSim. Communication between FrankaSim and CoppeliaSim is performed using ROS communication level. The ROS node may be thought of as a *main()* routine that first initializes the simulation (e.g., connecting the robot and camera, initializes the position, and so on) and then enters an endless *while* loop. At each iteration of the loop, from the robot state measurements (position, velocity or torque) and, potentially any other sensor measurement like an image acquired by a virtual camera, a new joint/Cartesian velocity/torque command is computed and sent back to CoppeliaSim that updates the scene rendering. There exists a mechanisms that allows to perform a synchronous execution between the simulation step in CoppeliaSim and the control program. Typically for a visual servoing simulation, one iteration of the loop could be synchronized at 20 ms (like for a real camera), while for an impedance control it is recommended to reduce this time step between 1 and 3 ms (the simulation will be more accurate with shorter time steps but will last longer). FrankaSim has been designed in such a way as to hide ROS from the user, making its use transparent. This approach makes it very easy to prototype an application in simulation first, and then to deploy it on a real robot by just changing few lines of code, as we will see in Section IV-A.

IV. Experiments

In this section we present two experiments to assess both the gap between the simulated and real robot and to demonstrate the great simulation potential of FrankaSim, which can easily handle highly complicated scenarios.

A. Single-Arm Experiment: Real vs Simulated

In this experiment we consider the following joint space torque controller

$$\boldsymbol{\tau}_t = \boldsymbol{B}(\boldsymbol{q})\left(\boldsymbol{K}\boldsymbol{e} + \boldsymbol{D}\dot{\boldsymbol{e}} + \boldsymbol{I}\int \boldsymbol{e}dt + \ddot{\boldsymbol{q}}_d\right) + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{F}_v(\dot{\boldsymbol{q}}) - \boldsymbol{\tau}_0 e^{-\mu t} \quad (2)$$

with $\boldsymbol{\tau}_t$ the joint torque command at time t, $\boldsymbol{q}_d(t)$ a desired joint trajectory, $\boldsymbol{e} = \boldsymbol{q}_d - \boldsymbol{q}$, and $\boldsymbol{K}$, $\boldsymbol{D}$, $\boldsymbol{I} \in \mathbb{R}^{7\times 7}$ are the proportional, derivative and integral diagonal gain matrices respectively. The exponential term allows for a smooth start of the commands making the computed joint torques to start from zero having set $\boldsymbol{\tau}_0 = \boldsymbol{\tau}_t(0)$ equal to the computed torque at $t = 0$ while μ determines the decay rate. A FER arm equipped with its gripper and a camera (not used) attached to the flange is controlled using (2). Figures 4 and 5 compares the joint positions and torques measured on a real FER versus the simulated one using FrankaSim while applying a desired sinusoidal joint trajectory on joints 1, 3

and 4. The signals obtained in the simulation closely resemble those recorded on the real robot, despite the latter being clearly noisier, as we expected. In the paper's video (https://www.youtube.com/watch?v=Kxn3pXsK9h4) we also show a Pose-Based Visual-Servoing (PBVS) and an IBVS experiment with an AprilTag target where the visual-servo behaviors are very close between the real and simulated cases. More than 95% of the C++ code is common between real and simulated experiments. Only about ten lines of code are simulator-specific to initiate communication with the robot and the camera if the latter is used.

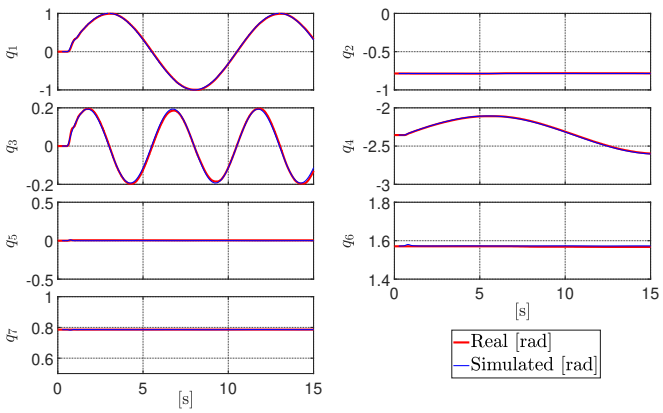

Fig. 4: Real vs Simulated measured joint positions while tracking a desired joint trajectory using controller (2) . The total RMS position Error along the whole trajectory ($[0.5; 15]$ [s]) for all the joints is $RMSE_q = 0.006$ [rad].

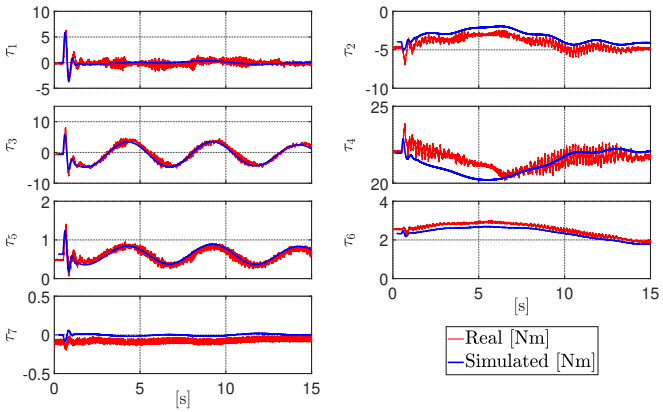

Fig. 5: Real vs Simulated measured joint torques while tracking a desired joint trajectory using controller (2) . The total RMS torque error along the whole trajectory ($[0.5; 15]$ [s]) for all the joints is $RMSE_\tau = 0.5546$ [Nm].

The results of the experiment emphasise the small reality gap between the simulated and real robots, which is crucial for accurately forecasting the outcomes of experiments conducted on a real setup. The discrepancies observed in the signals reported in Figs. 4 and 5 between real and simulated robots, are mostly due to non-simulated friction effects.

B. The Dual-Arm Experiment

This example was designed to showcase together most of the salient features of the simulator, *i.e.*, the possibility to handle multiple robots in the same program, the ability to control the robot in velocity or in torque, the capacity of the presented Lagrangian dynamic model library to fully compensate for the gravity and the payload (gripper) even when the robot is not vertically mounted, the synchronous execution with the physics simulator, as well as simulating a lower camera rate and a Visual-admittance [24] PBVS in interaction with the environment (right arm).

The simulation platform consists of a dual arm manipulator named "Franko", equipped with two FER arms with their grippers and a wrist-mounted camera and force/torque sensor on the right arm. The left arm is torque controlled; it implements a computed torque Cartesian controller and holds an AprilTag of the family 36h11. The right arm implements an Extended External Hybrid controller [24] to visually follow the AprilTag. At start, both arms are not servoed and the left one is only compensating for the gravitational effects (the arm itself + the gripper), then it is commanded to reach a desired pose in the workspace while the right arm is commanded to make sure that the AprilTag is centered in the image at a distance of 20 cm. Once the VS converges, the left arm starts tracking a circular trajectory with the end-effector, causing the right arm to follow. There is an obstacle on the worktable to which the gripper of the right arm collides with while chasing the AprilTag, resulting in a "D" shaped trajectory (see Fig. 6 and paper's video for more in depth and clear understanding of this experiment).

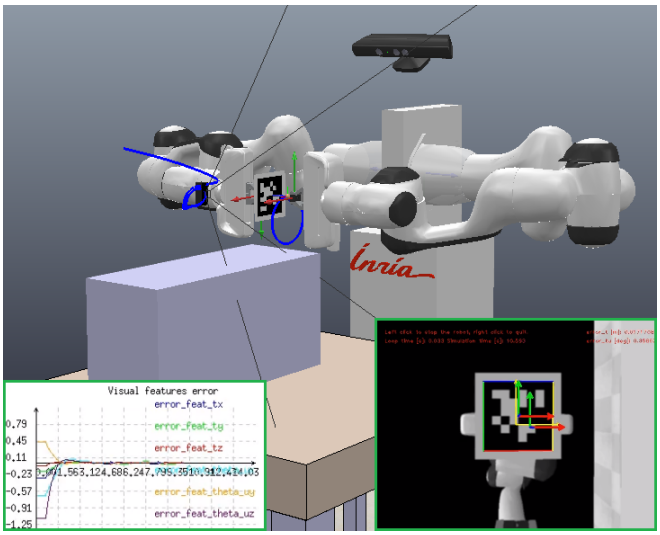

Fig. 6: Dual-Arm Experiment: two Franka Emika Robots constitute the arms of "Franko". The torque controlled left arm holds an AprilTag while following a circular trajectory. The right arm implements an Extended External Hybrid vision-force controller [24] to follow the AprilTag while dealing with the collision with the environment. The circular and "D" shaped path (due to the impact with the obstacle) of both arms are drawn (blue lines). The camera view and some plotted curbs are shown at the bottom of the figure.

V. Conclusions And Future Works

In this paper we presented our dynamic simulator for the popular Franka Emika Robot. The simulator has been

designed with most of the criteria of importance for the users in mind (small reality gap, open-source, simulation-to-real code transfer). It has a number of features that make it unique compared to other dynamic simulators for this robot; in fact, to date, it is the only one to support CoppeliaSim and to integrate a library providing the Lagrangian dynamic model of the robot that provides the full Coriolis matrix and the estimated dynamic friction as well as being fully incorporated into the ViSP ecosystem that enables it with VS capabilities. It has proven to be versatile and remarkably truthful to reality. Furthermore, its API allows to easily transfer verified code from simulation to the real implementation.

The simulator is currently being used by researchers in the Rainbow team for prototyping and testing novel sensor-based control strategies, to study dynamic visual servoing[5] [21] and the combined use of vision and force sensing for precision assembly tasks. We expect that making the code available to the community will increase the number of end-users and applications addressed.

As future work we aim at extending the support to other environments like MuJoCo that can also simulate deformable objects and Gazebo which, despite its limitations, is the most used simulator in research and could be ran with other physics engines like DART [32]. Since ROS is arriving to the end-of-life support, we are currently migrating towards ROS2.

REFERENCES

[1] M. Santos Pessoa de Melo, J. Gomes da Silva Neto, P. Jorge Lima da Silva, J. M. X. Natario Teixeira, and V. Teichrieb, "Analysis and comparison of robotics 3d simulators," in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, 2019, pp. 242–251.

[2] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 1082–10 828.

[3] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[4] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *IEEE Robotics Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[5] Franka Emika, "Ros integration for franka emika research robots," https://github.com/frankaemika/franka_ros.

[6] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[7] C. Gaz, M. Cognetti, A. Oliva, P. Robuffo Giordano, and A. De Luca, "Dynamic identification of the Franka Emika Panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robotics and Automation Lett.*, 2019.

[8] M. Križmančić, "Franka_gazebo," https://github.com/mkrizmancic/franka_gazebo.

[9] S. Sidhik, "Panda_simulator: Gazebo simulator for franka emika panda robot supporting sim-to-real code transfer," https://github.com/justagist/panda_simulator, DOI: 10.5281/zenodo.3818280.

[10] ——, "Mujoco panda," https://github.com/justagist/mujoco_panda.

[11] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[12] A. Trabelsi, J. S. Sandoval Arévalo, G. Moncef, and m. a. Laribi, *Development of a Franka Emika Cobot Simulator Platform (CSP) Dedicated to Medical Applications*, 05 2021, pp. 95–103.

[13] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. [Online]. Available: www.coppeliarobotics.com

[14] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, "Feature and performance comparison of the v-rep, gazebo and argos robot simulators," in *Towards Autonomous Robotic Systems*, M. Giuliani, T. Assaf, and M. E. Giannaccini, Eds. Cham: Springer International Publishing, 2018, pp. 357–368.

[15] J. Collins, D. Howard, and J. Leitner, "Quantifying the reality gap in robotic manipulation tasks," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6706–6712.

[16] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016. [Online] Available: https://pybullet.org.

[17] E. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics and Automation Magazine*, vol. 12, no. 4, pp. 40–52, December 2005.

[18] P. Corke, "The machine vision toolbox: a matlab toolbox for vision and vision-based control," *IEEE Robotics Automation Magazine*, vol. 12, no. 4, pp. 16–25, 2005.

[19] P. M. Khiabani, B. S. Aghdam, J. Ramezanzadeh, and H. D. Taghirad, "Visual servoing simulator by using ros and gazebo," in *2016 4th International Conference on Robotics and Mechatronics (ICROM)*, 2016, pp. 308–312.

[20] S. Vandernotte, A. Chriette, P. Martinet, and A. S. Roos, "Dynamic sensor-based control," in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2016, pp. 1–6.

[21] A. A. Oliva, E. Aertbeliën, J. de Schutter, P. Robuffo Giordano, and F. Chaumette, "Towards Dynamic Visual Servoing for Interaction Control and Moving Targets," in *ICRA 2022 - IEEE International Conference on Robotics and Automation*. Philadelphia, United States: IEEE, May 2022, pp. 1–7.

[22] G. Morel, E. Malis, and S. Boudet, "Impedance based combination of visual and force control," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 2, 1998, pp. 1743–1748 vol.2.

[23] Y. Mezouar, M. Prats, and P. Martinet, "External hybrid vision/force control," *Intl. Conference on Advanced Robotics*, 2007.

[24] A. A. Oliva, P. R. Giordano, and F. Chaumette, "A general visual-impedance framework for effectively combining vision and force sensing in feature space," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4441–4448, 2021.

[25] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modeling, Planning and Control*, 3rd ed. London: Springer, 2008.

[26] R. Smits, "KDL: Kinematics and Dynamics Library," http://www.orocos.org/kdl.

[27] P. Corke, "Robotics, Vision & Control," Springer 2017, ISBN 978-3-319-54413-7.

[28] C. Gaz and A. De Luca, "Payload estimation based on identified coefficients of robot dynamics — with an application to collision detection," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3033–3040.

[29] A. De Luca and L. Ferrajoli, "A modified newton-euler method for dynamic computations in robot fault detection and control," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 3359–3364.

[30] A. De Luca, A. Albu-Schaffer, S. Haddadin, and G. Hirzinger, "Collision detection and safe reaction with the dlr-iii lightweight manipulator arm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 1623–1630.

[31] W. Chung, L.-C. Fu, and S.-H. Hsu, *Motion Control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 133–159. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_7

[32] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, Feb 2018. [Online]. Available: https://doi.org/10.21105/joss.00500

[5]Further experimental videos of dynamic visual servoing using FrankaSim can be found at this link: `https://www.youtube.com/watch?v=fOSRnxd1dlE`