Rapid Deployment of Model Predictive Control for Robotic Systems: From IMPACT to ROS 2 Through Code Generation

Alejandro Astudillo, Alvaro Florez, Wilm Decré and Jan Swevers
MECO Research Team, Department of Mechanical Engineering, KU Leuven, Belgium
Flanders Make@KU Leuven, Leuven, Belgium
{alejandro.astudillovigoya, alvaro.florez, wilm.decre, jan.swevers}@kuleuven.be

***Abstract*—This paper presents an approach for automatic code generation of nonlinear model predictive controllers (NMPC) for robotic systems as ROS 2 packages. The developed approach extends IMPACT – a framework for NMPC specification, prototyping and deployment – to expand its usability towards the robotics community. With such extension, the wide variety of numerical optimization solvers interfaced within IMPACT are made available to any project involving a ROS 2 architecture. The extensions to IMPACT are based on three submodules that automatically generate robot models from robot description files, generate ROS 2 packages for NMPC-based robot control, and benchmark the computational complexity of generated functions and algorithms. A hardware-in-the-loop simulation is presented for tunnel-following NMPC on a 7-degrees-of-freedom robot manipulator, illustrating the workflow from problem definition to deployment on embedded hardware.**

***Index Terms*—Model predictive control, robot operating system, code generation, robotic systems**

I. Introduction

Nonlinear model predictive control (NMPC) offers an optimization-based methodology to program automated robotic applications in a declarative way. Following a constrained-based task specification approach [1], complex tasks – e.g., in autonomous racing [2], [3] or robot manipulation [4] – can be programmed by formulating (i) measures of desirability as objective functions and (ii) task- and system-related restrictions as constraint functions of an underlying optimal control problem (OCP). Nevertheless, programming such robot-based solutions is no easy task.

The *workflow* from problem specification to NMPC-based solution deployment on robotic systems has a high engineering cost. Such *workflow* requires, among others, (i) the adequate formulation of robot-inherent constraints, e.g., system dynamics, kinematics, and actuation limits, (ii) the selection of methods to transcribe the OCP into a finite-dimensional nonlinear program (NLP) and to solve such NLPs, (iii) prototyping the selected solver in relevant simulation environments, and (iv) deploying such solution onto a real robot.

With a large community of programmers and users, the latest version of the Robot Operating System (ROS) – i.e., ROS 2 [5] – is becoming the *de facto* robotics middleware for robot programming since it can achieve low-latency executions (satisfying real-time constraints) unlike ROS 1. Optimization frameworks, such as acados [6], have been successfully manually integrated into ROS 1 architectures [7]. However, such manual integration adds engineering complexity to the prototyping and deployment process. To reduce such engineering complexity, other optimization frameworks, such as OpEn [8], allow the automatic code-generation of ROS 1 packages. Nevertheless, such frameworks restrict the user to a limited set of numerical optimization algorithms, lacking modularity and flexibility for prototyping and selecting the most adequate solver and settings for the OCP at hand.

Some robotics-oriented optimization frameworks such as Tasho [9] and Horizon [10] have interfaced the rigid-body dynamics library Pinocchio [11] to automatically generate robot models from robot description files (URDF), facilitating the problem specification step of the *workflow*. These frameworks, however, (i) offer no direct interface to ROS 1 or ROS 2, and (ii) are tailored for poly-articulated systems – e.g., robot manipulators and legged robots –, not being suitable for mobile robot applications, for instance.

IMPACT [12] is an in-house developed NMPC framework that aims to simplify the aforementioned *workflow* by abstracting away from the user the implementation effort required to formulate and transcribe OCPs underpinning NMPC, by interfacing multiple state-of-the-art optimization solvers, and by generating relevant artifacts for NMPC prototyping and deployment. Being a general NMPC framework, it still lacks elements in the *workflow* that are tailored for robotic systems: it does not provide a way to efficiently compute complex robot models, e.g., for poly-articulated systems, and it requires the user to (manually) interface the generated artifacts with robot drivers and robotics middlewares to complete the deployment procedure. This work aims to overcome the aforementioned drawbacks by orchestrating the state-of-the-art open-source tools IMPACT, Pinocchio and ROS 2.

The main contribution of this work is a software module that automatically generates a ROS 2 package for prototyping and deploying NMPC for general robotic systems. In addition, this work presents a software module that facilitates the specification of dynamic and kinematic models for poly-

*The authors would like to thank Flanders Make SBO project DIRAC: "Deterministic and Inexpensive Realizations of Advanced Control" for supporting this research. Flanders Make is the Flemish strategic research centre for the manufacturing industry.

articulated robots and their use within an NMPC formulation, and the validation of the proposed *workflow* from problem definition to solution deployment through a hardware-in-the-loop simulation. These contributions are integrated within the NMPC toolchain IMPACT, extending its capabilities towards the robotics community. To the best of the authors knowledge, this is the first work that enables automatic code generation of NMPC solutions as easy-to-use ROS 2 packages.

The remainder of this work is organized as follows. Section II presents preliminary concepts and the background used in this work. Next, Section III describes the details of the software framework. Experiments and results are discussed in Section IV. We close this paper with concluding remarks.

II. Preliminaries

This section briefly discusses preliminaries that set the foundation of the contributions presented herein. We first describe concepts on NMPC, and then introduce the main frameworks related to this work.

A. Nonlinear model predictive control

The execution of NMPC requires the solution of an underlying OCP at each sampling time. Such OCPs can be formulated in continuous-time as follows:

$$\underset{x,\,u}{\text{minimize}} \quad \int_0^{t_\text{f}} V(x(t),u(t),p,t)\mathrm{d}t + V_f(x(t_\text{f}),p) \tag{1a}$$

$$\text{subject to} \quad B(x(0),x(t_f),p) \leq 0, \tag{1b}$$

$$\dot{x}(t) = f(x(t),u(t),p), \quad t \in [0,t_\text{f}], \tag{1c}$$

$$h(x(t),u(t),p) \leq 0, \quad t \in [0,t_\text{f}], \tag{1d}$$

where t_f is the time horizon, and $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$ and $p \in \mathbb{R}^{n_p}$ are the vectors of states, inputs and parameters, respectively. Moreover, $V(\cdot)$ and $V_\text{f}(\cdot)$ in (1a) represent Lagrange and Mayer terms of the objective function, $B(\cdot)$ in (1b) denotes initial and terminal constraints, $f(\cdot)$ in (1c) represents the system model, and $h(\cdot)$ in (1d) define general path constraints.

For an NMPC implementation, a feasible and optimal solution to OCP (1) must be computed in less than the sampling time to satisfy real-time constraints. Such computation requires, in general, using a numerical optimization solver to find a solution to a finite-dimensional NLP that is an equivalent transcription of the infinite-dimensional OCP. Pairing adequate combinations of (i) OCP formulations, (ii) transcription methods – e.g., single-shooting, multiple-shooting, direct collocation or B-spline methods –, and (iii) NLP solvers to achieve optimal performance is a complex task that requires the flexibility to easily prototype multiple options.

B. The NMPC Toolchain IMPACT

IMPACT is a flexible toolchain that aims to reduce the engineering complexity required to specify, prototype and deploy NMPC for dynamic systems. It is built on top of the optimization framework CasADi [13] and the optimal control framework Rockit [14].

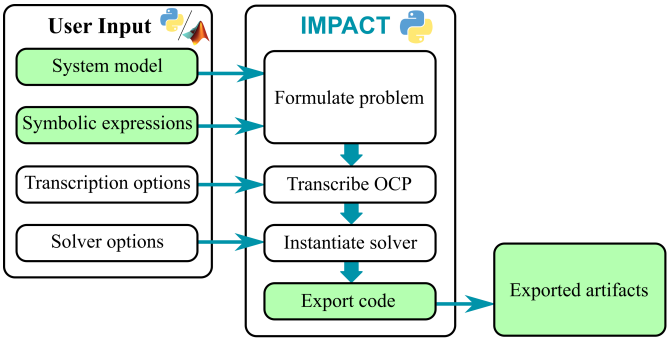

Fig. 1: Overview of the workflow of IMPACT toolchain.

The workflow represented in Fig. 1 shows the three main components that represent the procedure of using IMPACT: the user input, the inner toolchain processes and the exported artifacts. In green are highlighted the elements involved in the contributions of this work. IMPACT toolchain incorporates an MPC class, and all the elements from the user component should be defined through an instantiation of an object from the MPC class.

In the implementation of an NMPC, it is essential to define the system model. In the IMPACT toolchain the model can be defined with multiple methods. The first method consists of explicitly defining the state space model: Create variables for the states and inputs, and then designate the derivative expressions of the states, this means, the definition of the first order ordinary differential equations (ODE). Instead of defining the model directly, it can be imported from a yaml file. It contains the equations and parameters or includes the path to a serialized CasADi function that contains an expression that defines the system dynamics. The following code snippet shows the structure of the yaml file with a two equivalent simplified pendulum examples, in the left column inline expressions explicitly define the model, in the right column a serialized CasADi function `ode.casadi` is incorporated.

```yaml
equations:
  inline:
    ode:
      phi: dphi
      dphi: L*cos(phi)*sin(phi) ...
differential_states:
  - name: phi
  - name: dphi
controls:
  - name: F
constants:
  inline:
    L: 2
```

```yaml
equations:
  external:
    type: casadi_serialized
    file_name: ode.casadi
differential_states:
  - name: phi
  - name: dphi
controls:
  - name: F
```

Once the yaml file is defined, the method `add_model` of the `MPC` class is used to include the system model, e.g., as `rob = mpc.add_model('rob', 'rob.yaml')`.

The objective and constraints of the OCP are nonlinear functions that can be defined by using symbolic expressions of the states and inputs, allowing the user to easily customize any type of problem.

Selecting a solver is also an important user task. The solver should be suitable to handle the complexity of the problem and to find a solution in a feasible time. The IMPACT toolchain

interfaces multiple widely used and state-of-the-art solvers, e.g., `IPOPT` [15], `GRAMPC` [16], `FATROP` [17]. Information on diverse solvers available is presented in IMPACT documentation[1].

Once the NMPC is defined, the export method is called, then, IMPACT toolchain generates the artifacts that allow us to deploy the NMPC solution. They are the third component in the workflow presented in Fig. 1. It provides a C application programming interface (API) that sets the parameter values and retrieves the OCP solution by using a unified syntax – i.e., a transparent syntax that is independent of the used solver. One of the artifacts created is a Simulink block. In this block, the parameters p of the OCP (1) are set as input ports, while the states and control variables x and u are set as output ports. Thus, the Simulink block represents the OCP solver and, connected in a closed loop with the actual or simulated system, represents the whole NMPC implementation. This artifact can be used for simulation or for deployment of the strategy in a real-time execution interfaced with the real system.

C. Robotic operating system (ROS)

ROS is a software designed to easily interface sensors, actuators, and control systems in a modular architecture. ROS interfaces a wide range of robot components, providing large flexibility in the robot structure and tasks to be performed. A ROS implementation has a graph structure where each node represents a component and executes specific tasks. The nodes can share information using topics and messages. The version ROS 2 has been widely adopted by the robotics community due to the improvements with respect to the ROS 1 version. Among the improvements are: the extension of development language including C, the incorporation of security features, and one of the most important improvements, the better support for real-time systems making it suitable for systems with execution time constraints.

III. Software Developments

In this section, we introduce the general architecture of the proposed software modules. First, we present a `RobotModel` module, which facilitates the generation of robot models – i.e., robot dynamics, kinematics and inherent system constraints – and their integration within NMPC specifications. Then, we present a `ROS2PackageGenerator` module, which extends IMPACT's capabilities of artifact generation towards its use within robotics applications. Finally, we introduce the `Benchmark` module, which facilitates the evaluation of the computational burden of the generated artifacts.

A. Robot models

Adequate robot models are essential for the correct specification of NMPC for robotic systems. Defining such models is, however, not an easy procedure. It requires deep knowledge of the robot description including, e.g., actuation limits, kinematics and inertial parameters. The aforementioned robot description can be obtained from Unified Robotics Description Format (URDF) files, which are usually provided by robot vendors.

Robot description files – e.g., URDF – can be parsed in order to extract the necessary information to generate functions for robot kinematics, robot dynamics, and meta-data related to constraints inherent to the system. To achieve this, we present a `RobotModel` module for IMPACT, which facilitates the generation of system models and symbolic expressions (see Fig. 1) that can be used to specify OCPs underpinning NMPC for robotic systems. To achieve this, this module relies on the state-of-the-art rigid-body dynamics library Pinocchio [11] and its interface with the optimization framework CasADi presented in [9], [18]. As CasADi is the back-end of IMPACT, functions and symbolic expressions generated with CasADi are directly usable within IMPACT.

Given a URDF file (for instance `robot.urdf`) describing a poly-articulated robot – e.g., legged robots or robot manipulators –, by initializing a `RobotModel` object as `robot = RobotModel('robot.urdf')`, this module generates CasADi functions for the forward kinematics of every link in the kinematic tree, forward dynamics, inverse dynamics, and joint torque regressor, among others. It also generates efficient analytical derivatives of kinematic and dynamic functions, which accelerate the solution of OCPs relying on such functions. In addition, it extracts information regarding lower and upper bounds on joint positions, velocities, accelerations and torques. The interested reader is referred to [18] for further details on the advantage of analytical derivatives of robot kinematics and dynamics.

The generated `RobotModel` object can be used directly by the user to specify an NMPC. However, in an effort to standardize the definition of models that depend on ODEs and DAEs, the generated robot models can be also exported as a compressed file – which has been labelled with the extension `.impz` – to facilitate sharing such models with collaborators. This procedure is performed by calling the `export_model` method of the `RobotModel` object as `robot.export_model('rob')`, which generates a file `rob.impz`. This file contains all the generated CasADi functions in a serialized format, a definition of the ODE or DAE that represents the system dynamics, and a `yaml` file that defines the state, control and algebraic variables of the system (as explained in Section II-B).

The generated robot model `rob.impz` can be directly imported by IMPACT by using the `add_model` method of the `MPC` class (as explained in Section II-B), by using `robot = mpc.add_model('rob', 'rob.impz')`.

B. ROS 2 package generator

Until now, IMPACT was able to code-generate three artifacts for NMPC prototyping and deployment, namely the numerical back-end that interfaces numerical optimization solvers, an IMPACT library that exposes the IMPACT C API, and a Simulink block that can be directly imported into a Simulink model and deployed by using the Simulink Coder (see Section II-B).

[1]https://meco-software.pages.gitlab.kuleuven.be/impact/

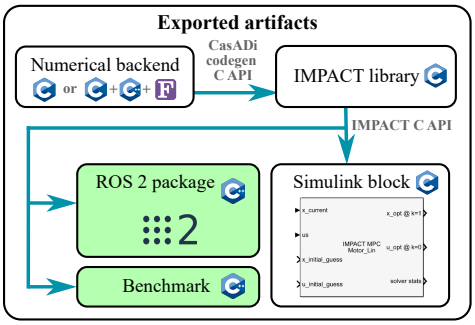

Fig. 2: Overview of the artifacts that the user can export by using IMPACT.

The robotics community makes extensive use of ROS, and of its newest version ROS 2, to prototype and deploy – both in simulation and on real robots – strategies for motion planning and control, among others. This calls for the development of a `ROS2PackageGenerator` module for IMPACT to allow automatic code generation of ROS 2 packages that can easily be integrated within architectures that use ROS 2.

The `ROS2PackageGenerator` module lies within the export code procedure of IMPACT (see Fig. 1). This module generates a complete ROS 2 package that can be used within a ROS 2 workspace without further user script manipulation.

This module automatically generates adequate `package.xml` and `CMakeLists.txt` files that consider the dependencies of the numerical backend of the NMPC solver. For instance, in case the NLP solver IPOPT is set by the user, the `CMakeLists.txt` file will include directives to find IPOPT in the system of the user, and to link the generated nodes to the IPOPT library.

The generated package contains two ROS 2 C++ nodes: (i) an NMPC node, and (ii) a simulation node. The NMPC node imports the IMPACT library artifact, and makes use of the IMPACT C API to create an interface with the numerical backend artifact (see Figure 2). This node mimics the input and outputs of the generated Simulink block (see Section II-B), and for each input/output it creates a ROS 2 topic based on messages containing float arrays. The simulation node, on the other hand, allows the user to rapidly prototype the generated NMPC by simulating the system by means of a discretized version of the system dynamics based on the ODE or DAE defined, e.g., from the `yaml` or `impz` files. Such discretization is achieved by using a fourth-order Runge-Kutta integrator.

To use the `ROS2PackageGenerator`, the user must instantiate an object of the class `ROS2Generator` from IMPACT. This object is initialized with multiple arguments including the `mpc_specification` which corresponds to the specified MPC object from IMPACT, `package_name` and `nmpc_node_name` which gives a name to the generated package and NMPC node, respectively, `version` and `author` that allow the user to set meta-data of the generated package, among others. Finally, the ROS 2 package is actually generated after calling the `export_package` method of the instantiated object of `ROS2Generator`. An example of this procedure is shown in the code snippet below.

```
ROS2gen = ROS2Generator(
  mpc_specification = mpc
  package_name = 'my_ROS2_package',
  nmpc_node_name = 'nmpc_node',
  version = '0.0.1',
  author = {'name': ..., 'email': ...}
)
ROS2gen.export_package()
```

C. Benchmark generator

The `Benchmark` module facilitates the assessment of the computational efficiency of any `CasADi` function in terms of evaluation time and number of atomic operations. For `CasADi` functions involving instances of numerical optimization solvers – as the NMPC solvers generated by IMPACT –, the evaluation time is interpreted as the total solution time required to solve the optimization problem. This module includes methods to (i) automatically generate C code related to the `CasADi` function subject of the benchmark and a benchmark script, (ii) compile the code-generated files with a user-defined selection of compiler and compilation flags, and (iii) execute the benchmark with a defined number of repetitions. The execution of the benchmark generates a text file including the benchmark results.

IV. Hardware-in-the-Loop illustration

In this section, we present an implementation example that illustrates the use of the software modules introduced in Section III. We present a Hardware-in-the-Loop simulation described using IMPACT and deployed on an NVIDIA Jetson AGX Orin employing ROS 2 packages.

A. Problem formulation

The problem is a tunnel-following task with a 7-degrees-of-freedom Kinova Gen3 robot manipulator. The robotic system dynamics are characterized by $\dot{\boldsymbol{x}} = \xi(\boldsymbol{x}(t), \boldsymbol{u}(t))$, where the state vector is $\boldsymbol{x}(t) \coloneqq \begin{bmatrix} q^\top(t) & \dot{q}^\top(t) \end{bmatrix}^\top$ the joint angles and angular velocity vectors, the input $\boldsymbol{u}(t)$ is the torque vector τ . The objective is to guide the end-effector along a parameterized trajectory while maintaining position and rotation within boundaries defined as the tunnel. To introduce the trajectory in the problem formulation we first define the path progress parameter $s \in [0, 1]$. Additionally, the reference position vector $\boldsymbol{p}_{\text{ref}}(s)$, reference rotation matrix $\mathcal{R}_{\text{ref}}(s)$, and reference path velocity $\dot{s}_{\text{ref}}(s) \geq 0$, all are known a priori. Let's define the path-related system with states $\mathcal{X} = [s, \dot{s}]$, and input $\mu = \ddot{s}$, whose dynamics correspond to a double integrator. Now, we consider the augmented dynamics $\hat{\dot{\boldsymbol{x}}} = \hat{\xi}(\hat{\boldsymbol{x}}(t), \hat{\boldsymbol{u}}(t))$, where $\hat{\boldsymbol{x}} = [\boldsymbol{x}^\top, \mathcal{X}^\top]^\top$, and $\hat{\boldsymbol{u}} = [\boldsymbol{u}^\top, \mu^\top]^\top$.

The position error is $e_{\text{pos}} = \boldsymbol{P}_{\text{ee}} - \boldsymbol{P}_{\text{ref}}$, representing the difference between the end-effector position and reference position, the path progression error is $e_{\dot{s}} = \dot{s} - \dot{s}_{\text{ref}}$. Considering the end-effector rotation matrix as $\mathcal{R}_{\text{ee}}$, the orientation error $e_{\text{rot}} = (\mathcal{R}^a_{\text{ref}} \times \mathcal{R}^a_{\text{ee}} + \mathcal{R}^b_{\text{ref}} \times \mathcal{R}^b_{\text{ee}} + \mathcal{R}^c_{\text{ref}} \times \mathcal{R}^c_{\text{ee}})/2$, where a

rotation matrix consists of three column vectors with the form $\mathcal{R}_j = [\mathcal{R}_j^a \ \mathcal{R}_j^b \ \mathcal{R}_j^c]$. The tunnel is defined as constraining the position and orientation error

$$\begin{bmatrix} \|e_{\text{pos}}\ (\hat{x}(t))\|^2 \\ \|e_{\text{rot}}\ (\hat{x}(t))\|^2 \end{bmatrix} \leq \begin{bmatrix} \rho_{\text{pos}}^2 \\ \rho_{\text{rot}}^2 \end{bmatrix} + \begin{bmatrix} l_{\text{pos}}(t) \\ l_{\text{rot}}(t) \end{bmatrix},$$

where ρ_{pos} and ρ_{rot} are the user defined tolerance (tunnel radius), $l(t) = [l_{\text{pos}}^\top(t)\ l_{\text{rot}}^\top(t)]^\top$ is a slack variable that relaxes the path constraints. we define the following OCP that makes the robot end-effector follow a position and orientation tunnel along the trajectory

$$\min_{\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, l} \int_0^T \left\| \begin{bmatrix} e_{\text{pos}}(t) \\ e_{\text{rot}}(t) \\ e_{\dot{s}}(t) \\ \hat{\boldsymbol{x}}(t) \\ \hat{\boldsymbol{u}}(t) \end{bmatrix} \right\|_{\mathbf{Q}}^2 + \|l(t)\|_{\mathbf{Q}_l}^2 \ \mathrm{d}t + \left\| \begin{bmatrix} e_{\text{pos}}(T) \\ e_{\text{rot}}(T) \\ \dot{q}(T) \\ s(T) - 1 \\ \dot{s}(T) \end{bmatrix} \right\|_{\mathbf{Q}_T}^2 \tag{2a}$$

$$\text{s.t.} \quad \hat{\boldsymbol{x}}(0) = \hat{\boldsymbol{x}}_{\text{est}}, \tag{2b}$$

$$\dot{\hat{\boldsymbol{x}}}(t) = \hat{\xi}(\hat{\boldsymbol{x}}(t), \hat{\boldsymbol{u}}(t)), \qquad t \in [0, T], \tag{2c}$$

$$\|e_{\text{pos}}\|^2 \leq \rho_{\text{pos}}^2 + l_{\text{pos}}(t), \ t \in [0, T], \tag{2d}$$

$$\|e_{\text{rot}}\|^2 \leq \rho_{\text{rot}}^2 + l_{\text{rot}}(t), \quad t \in [0, T], \tag{2e}$$

$$[\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}, l] \in \mathcal{W}, \qquad t \in [0, T] \tag{2f}$$

The initial state $\hat{\boldsymbol{x}}(0)$ corresponds to $\hat{\boldsymbol{x}}_{\text{est}}$ which represents the estimated or measured state vector. The space $\mathcal{W}$ is defined by considering the following constraints: $q_{\min} \leq q(t) \leq q_{\max}$, $\dot{q}_{\min} \leq \dot{q}(t) \leq \dot{q}_{\max}$, $0 \leq s(t) \leq 1$, $\dot{s} \geq 0$, $\tau_{\min} \leq \tau(t) \leq \tau_{\max}$, and $l(t) \geq 0$. The reference trajectory is a lemniscate-shaped path, an eight-shape curve that starts and finishes at the same point. For additional details about the formulation and weight values, please refer to [19].

Remark 1: Setting $\rho_{\text{pos}} = 0$ and $\rho_{\text{rot}} = 0$ would bring the tunnel-following NMPC scheme back to a path-following NMPC implementation.

B. Implementation details

Using the IMPACT toolchain, we easily described the problem (2) by incorporating the Kinova Gen3 robot model (`.impz` file). To solve the OCP at every sampling instant, we use CasADi's Sequential Quadratic Programming (SQP) solver, and the Quadratic Programming (QP) solver QRQP to solve the QP subproblems. These solvers are completely code-generated in the numerical back-end artifact, ensuring dependency-free operation. During the exportation procedure, we obtained the `ROS2PackageGenerator` module. Subsequently, we instantiate the class `ROS2Generator` to export the ROS 2 package. From the package, we get the system simulation node that was configured in a Desktop PC, while the NMPC node was set up on an NVIDIA Jetson AGX Orin, as illustrated in Fig. 3. The execution of NMPC within a ROS 2 architecture was performed on an NVIDIA Jetson AGX Orin, connected via Ethernet to a Desktop PC with an Intel Core i9-9900X CPU and 16 GB RAM, where a simulation node

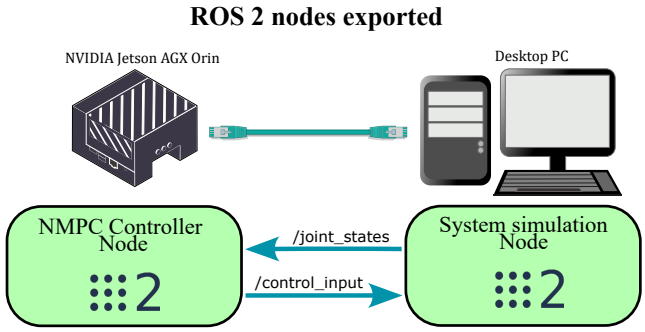

Fig. 3: ROS 2 node diagram. This diagram shows the hardware devices interconnected and the nodes created with IMPACT, the arrows show the topics published/read between nodes.

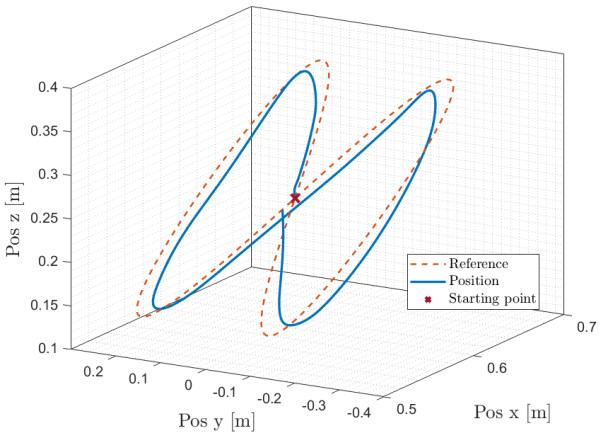

Fig. 4: End-effector position result of NMPC executing a time optimal tunnel-following task with the Kinova Gen3

is executed. In Fig.3 the diagram illustrating the implemented node scheme is presented. Both topics, `/joint_states` and `/control_input` handle messages of type JointState. For the `/joint_states`, the position and velocity elements are set, while for `/control_input`, only joint velocities are provided. In the OCP model dynamics, the system has torque inputs that require updates faster than 1 ms. Since the OCP takes a longer time to solve, joint velocities are sent to the simulated robot, and the low-level torque controllers are set by using internal PID controllers.

C. Results

The end-effector trajectory results, depicted in Fig. 4, demonstrate that the end-effector follows the reference trajectory while remaining within the tunnel. The average NMPC solution time is 7.5 ms, with a maximum and minimum of 8.4 ms and 7.1 ms, respectively.

In Fig. 5, the error trajectory illustrates the position deviation of the end-effector from the reference along the trajectory. The color indicates the progress along the trajectory, while the sphere represents the tunnel, which is the maximum allowed

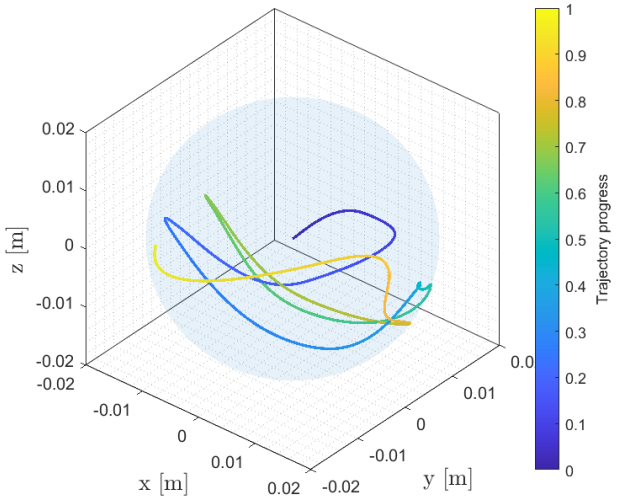

Fig. 5: Trajectory deviation. Position error along the trajectory, the sphere represents the tunnel (maximum deviation), and the color represents the progress in the trajectory.

deviation from the reference. The end-effector's initial position is on the reference, so the error trajectory starts at zero, the sphere origin (blue side). Subsequently, the error trajectory evolves within the sphere, indicating that the end-effector remains within the tunnel boundaries.

V. Conclusions

The modules introduced in this work extend the IMPACT toolchain with new features to generate ROS 2 packages. The new tools facilitate the specification of system models for NMPC and simplify the generation of NMPC ROS 2 packages. These tools allow the users to effortlessly generate ROS 2 nodes, enabling the deployment of NMPC for robotics systems across processing systems compatible with ROS 2. To illustrate the practical application of these modules, we conducted a Hardware-in-the-Loop simulation for a tunnel-following NMPC using the Kinova Gen3 robot model showing an adequate performance of all modules.

References

[1] J. D. Schutter, T. D. Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, May 2007.

[2] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3340–3356, Dec. 2022.

[3] P. Scheffe, T. M. Henneken, M. Kloock, and B. Alrifaee, "Sequential convex programming methods for real-time optimal trajectory planning in autonomous vehicle racing," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 661–672, Jan. 2023.

[4] N. Dehio, Y. Wang, and A. Kheddar, "Dual-arm box grabbing with impact-aware MPC utilizing soft deformable end-effector pads," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5647–5654, Apr. 2022.

[5] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, May 2022.

[6] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, no. 1, pp. 147–183, Oct. 2021.

[7] B. B. Carlos, T. Sartor, A. Zanelli, G. Frison, W. Burgard, M. Diehl, and G. Oriolo, "An efficient real-time NMPC for quadrotor position control under communication time-delay," in *16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2020, pp. 982–989.

[8] P. Sopasakis, E. Fresk, and P. Patrinos, "OpEn: Code generation for embedded nonconvex optimization," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6548–6554, 2020.

[9] A. S. Sathya, A. Astudillo, J. Gillis, W. Decre, G. Pipeleers, and J. Swevers, "Tasho: A Python toolbox for rapid prototyping and deployment of optimal control problem-based complex robot motion skills," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022.

[10] F. Ruscelli, A. Laurenzi, N. G. Tsagarakis, and E. M. Hoffman, "Horizon: A trajectory optimization framework for robotic systems," *Frontiers in Robotics and AI*, vol. 9, Jul. 2022.

[11] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, Jan. 2019.

[12] A. Florez, A. Astudillo, W. Decré, J. Swevers, and J. Gillis, "IMPACT: A toolchain for nonlinear model predictive control specification, prototyping, and deployment," in *Proceedings of the 22nd IFAC World Congress*, Jul. 2023.

[13] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[14] J. Gillis, B. Vandewal, G. Pipeleers, and J. Swevers, "Effortless modeling of optimal control problems with rockit," in *39th Benelux Meeting on Systems and Control, Date: 2020/03/10-2020/03/12, Location: Elspeet, The Netherlands*, 2020.

[15] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, May 2006, copyright: Copyright 2008 Elsevier B.V., All rights reserved.

[16] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (GRAMPC)," *Optim. Eng.*, vol. 20, no. 3, pp. 769–809, Sep. 2019.

[17] L. Vanroye, A. Sathya, J. D. Schutter, and W. Decré, "Fatrop : A fast constrained optimal control problem solver for robot trajectory optimization and control," 2023.

[18] A. Astudillo, J. Carpentier, J. Gillis, G. Pipeleers, and J. Swevers, "Mixed use of analytical derivatives and algorithmic differentiation for NMPC of robot manipulators," *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 78–83, 2021.

[19] A. Astudillo, J. Gillis, W. Decré, G. Pipeleers, and J. Swevers, "Towards an open toolchain for fast nonlinear mpc for serial robots." *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9814–9819, 2020, 21st IFAC World Congress.