

# A Framework for Motion Cueing Algorithm Development Demonstrated With a Nonlinear MPC-Based Approach

Thomas Bernhofer\*, Andreas Seefried<sup>†</sup> and Carsten Oldemeyer<sup>‡</sup>  
*German Aerospace Center (DLR), Weßling, 82234, Germany*

**Motion Cueing Algorithms (MCAs) convert simulated vehicle movements into commands for physical motion simulators, enabling the reproduction of realistic motion sensations for a pilot. Various MCA approaches have been developed over the past few decades, ranging from classic washout filters to optimization-based formulations and machine learning-based methods. While the MCA is typically the primary focus of development, significant effort is often devoted to implementing the associated infrastructure, including the interface to the motion simulator, reference trajectory generation, data processing, and visualization. This additional work is time-consuming and distracts from the core task of algorithm development. In this article, we present a modular framework that facilitates the development, implementation, and evaluation of MCAs on motion simulators. The framework provides standardized interfaces, integrated evaluation tools, and real-time communication with the motion simulator, thereby simplifying the overall integration effort and establishing a foundation for future motion cueing research and experimental studies. The framework is demonstrated using a nonlinear model predictive control-based MCA on the DLR Robotic Motion Simulator with a flight dynamics model of a Diamond DA42.**

## I. Introduction

Motion simulators are commonly used in pilot training, vehicle design, and human factors research to replicate the motion sensations of real-world vehicles. A key element is the Motion Cueing Algorithm (MCA), which converts desired vehicle movements into commands that can be executed within the limited workspace of the motion platform. In recent decades, various types of MCAs have been developed, ranging from filter-based approaches [1], to optimization-based methods [2–5], and machine learning-based methods [6].

The classical washout filter (CWF) is among the oldest MCA approaches, yet it remains widely used for its computational efficiency, ease of implementation, and stable behavior. The idea behind filter-based approaches is to separate the desired motion into high- and low-frequency components, where the high-frequency components are used to generate translational cues and the low-frequency components are used for tilt coordination. Tilt coordination is a technique in which the platform is slowly rotated so that a component of gravity creates a sensation of motion similar to sustained linear acceleration, without requiring any translational movement [7]. Although filter-based approaches are computationally efficient and simple to implement, a limitation is that additional objectives cannot be incorporated explicitly, but are instead considered indirectly by tuning the filter parameters. This limitation motivates the use of optimization-based methods, such as Model Predictive Control (MPC), or machine learning-based methods that can handle complex, nonlinear relationships. In driving simulation, MPC has become popular for developing MCAs because it can effectively integrate actuator constraints and perception-based objectives, as well as handle kinematic redundancy of motion simulators [8]. Furthermore, it considers a prediction horizon, allowing the system to anticipate upcoming movements when future trajectory data is accessible. However, applying MPC also presents challenges, including higher computational and implementation complexity, the need to tune many parameters, and the risk of instability if the parameters are poorly tuned.

While the design of the MCA itself is often the main scientific focus, its practical evaluation on a real motion simulator requires a substantial amount of additional infrastructure. Reference signals must be generated from a vehicle model, transformed into suitable input quantities, exchanged between distributed software components, visualized, evaluated, and finally transmitted to the motion platform under real-time constraints. In practice, implementing a new MCA therefore often requires adapting or reimplementing components such as the real-time control loop, the

---

\*Research Associate, Institute of Flight Systems, thomas.bernhofner@dlr.de, AIAA member

<sup>†</sup>Research Associate, Institute of Flight Systems, andreas.seefried@dlr.de

<sup>‡</sup>Research Associate, Institute of Robotics and Mechatronics, carsten.oldemeyer@dlr.de

motion-platform interface, logging mechanisms, visualization tools, and performance-evaluation routines. As a result, considerable development effort is spent on software integration rather than on the systematic design, tuning, and evaluation of the MCA. This motivates the development of a modular framework that separates the different components required for motion simulation into reusable modules and provides common interfaces that allow individual modules to be exchanged and different MCAs to be implemented, tested, and compared.

Some publications have successfully demonstrated the real-time implementation of MPC-based MCAs for driving simulators [2–5]. Typically, real-time capability for motion simulation requires control frequencies of around 100 Hz. The work by Beghi et al. [2] presents the first real-time implementation of a motion cueing algorithm based on linear MPC for a motion simulator with 6 degrees of freedom (DOF). They shortly describe a control chain in which signals from the simulation software are processed by a vestibular model and then passed to the MCA solved online with the C++ tool qpOASES at 100 Hz, before being sent to the platform control system. Building on this, Bruschetta et al. [3] proposed a nonlinear MPC approach for a 9-DOF platform in simulation, using simplified system kinematics to ensure computational tractability. For the numerical solution, they formulated the NMPC problem in the ACADO toolkit and solved it with a sequential quadratic programming method using a real-time iteration (RTI) scheme. They reported being capable of real-time operation at 100 Hz. Katliar et al. [4] demonstrated a real-time-capable nonlinear MPC approach on an 8-DOF robotic motion system. It was implemented in C++ using a RTI scheme with CasADi-generated C code and HPMPC, and operated with a sampling time of 12 ms. More recently, Lamprecht et al. [5] introduced a fully nonlinear MPC-based approach with driver prediction for a 7-DOF driving simulator. The paper outlines the overall signal flow of the MPC-based motion-cueing control scheme, whereas the implementation details focus on the GRAMPC-based solution of the MPC problem. The authors reported average computation times of 1.262 ms and a maximum of about 2 ms, remaining below the 4 ms real-time limit. While these studies have made substantial progress in MPC-based motion cueing, they have focused more on algorithm design than on the underlying software architecture or its integration with the motion simulator.

Despite its growing use in driving simulation, we are not aware of any publications documenting the implementation of an MPC-based MCA for flight simulation on a real motion platform. Existing work is largely restricted to simulation-based studies, in which several contributions compare CWF- and MPC-based motion cueing approaches [9–11]. The study by Tong et al. [9] concluded that MPC outperforms CWF by providing more accurate and phase-consistent motion cues, especially during highly dynamic upset scenarios. Bakker et al. [10] also found that MPC can achieve better tracking performance than CWF, provided that accurate future motion prediction is available. When no information about the future motion profile is available, as in most real-time motion simulator applications, the performance of MPC and CWF is comparable. Zhao et al. [11] also reported that MPC-based MCAs outperform CWFs in terms of tracking accuracy and motion fidelity because MPC optimizes the motion-perception error online rather than relying on fixed filter behavior.

Motivated by the lack of documented real-time implementations of MPC-based MCAs for flight simulation, and by the limited attention given in previous work to the software infrastructure required for such implementations, this paper presents a modular framework for the development, implementation, and evaluation of motion cueing algorithms on real motion simulators. The framework aims to reduce development and integration effort through reusable modules, standardized interfaces, integrated evaluation tools, and automatic parameter tuning. To demonstrate the proposed approach, a nonlinear MPC-based MCA is implemented on the DLR Robotic Motion Simulator (RMS) using a flight dynamics model of a Diamond DA42 aircraft. The remainder of the paper is organized as follows. Section II introduces the motion system, the human perception model, and the MPC formulation. Section III describes the proposed framework and its software architecture. Section IV presents and discusses the obtained results, and Section V concludes the paper.

## II. Motion Cueing Algorithm

### A. Motion System

The RMS is a motion simulator with 7 DOF based on an industrial robot arm mounted on a linear track with a length of 8 m. The robot arm has a payload capacity of 500 kg, holding a cockpit mockup with a pilot seat and a visual display system using two projectors. Alternatively, a head-mounted display can be used for visualization. The motion platform can generate a wide range of motion cues, and is therefore suitable for simulating various flight conditions, from normal flight to highly dynamic maneuvers. To control the movement of the robot, its joint angles are commanded every 12 ms via an interface called RSI (Robot Sensor Interface), which also provides feedback on the actual joint angles.

The motion platform is approximated for modeling as a double integrator in joint space, assuming that the



**Fig. 1 DLR Robotic Motion Simulator during operation.**

underlying robot controller accurately tracks the commanded joint motion over each sampling interval [5]. This yields a control-oriented model that is sufficiently accurate for motion cueing purposes while avoiding the complexity of detailed actuator and structural dynamics. The vector  $\mathbf{q} \in \mathbb{R}^7$  denotes the RMS joint coordinates, consisting of the six robot joint angles and the position of the linear axis as the last entry. Defining the motion system state as  $\mathbf{x}_s = [\mathbf{q}^\top, \dot{\mathbf{q}}^\top]^\top \in \mathbb{R}^{14}$ , and the motion system input as  $\mathbf{u}_s = \ddot{\mathbf{q}} \in \mathbb{R}^7$ , the double integrator simulator dynamics are given by

$$\dot{\mathbf{x}}_s = \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{u}_s \end{bmatrix}. \quad (1)$$

For the human perception model, introduced in Sec. II.B, the acceleration  ${}_P\mathbf{a}_P$  and the angular velocity  ${}_P\boldsymbol{\omega}_P$  acting on the pilot are required. The acceleration  ${}_P\mathbf{a}_P$  describes the acceleration of the pilot frame  $P$  expressed in frame  $P$ . The pilot frame is a body-fixed coordinate system centered at the pilot's head, with the  $x$ -axis pointing forward and the  $z$ -axis pointing down. Instead of deriving velocities and accelerations by differentiating the forward kinematics based on Denavit–Hartenberg parameters, the kinematic quantities are built up recursively from the base to the end-effector. Each element of the chain is treated as either a *revolute* or a *prismatic* element with a generalized coordinate  $q_i$  assigned to active, time-varying joints or a constant value assigned to structural elements.

#### *Revolute Element*

A revolute element describes the relative rotation about the unit axis  $\mathbf{e}_i$  by the angle  $\varphi_i$ . The corresponding relative rotation matrix  ${}_i\mathbf{R}_{i-1}$  maps vector components from frame  $i-1$  to frame  $i$  by rotating about the fixed unit axis  $\mathbf{e}_i$ . The recursive relations are then

$${}_i\mathbf{R}_0 = {}_i\mathbf{R}_{i-1} {}_{i-1}\mathbf{R}_0, \quad (2)$$

$${}_i\boldsymbol{\omega}_i = {}_i\mathbf{R}_{i-1} ({}_{i-1}\boldsymbol{\omega}_{i-1} + \mathbf{e}_i \dot{\varphi}_i), \quad (3)$$

$${}_i\dot{\boldsymbol{\omega}}_i = {}_i\mathbf{R}_{i-1} ({}_{i-1}\dot{\boldsymbol{\omega}}_{i-1} + \mathbf{e}_i \ddot{\varphi}_i + {}_{i-1}\boldsymbol{\omega}_{i-1} \times \mathbf{e}_i \dot{\varphi}_i), \quad (4)$$

$${}_0\mathbf{r}_i = {}_0\mathbf{r}_{i-1}, \quad {}_0\mathbf{v}_i = {}_0\mathbf{v}_{i-1}, \quad {}_0\mathbf{a}_i = {}_0\mathbf{a}_{i-1}. \quad (5)$$

For a fixed rotational offset,  $\dot{\varphi}_i = \ddot{\varphi}_i = 0$ , such that the element only changes the coordinate frame through the constant relative rotation matrix  ${}_i\mathbf{R}_{i-1}$ .

#### *Prismatic Element*

A prismatic element translates along the body-fixed unit axis  $\mathbf{e}_i$  by displacement  $s_i$ :

$${}_i\mathbf{R}_0 = {}_{i-1}\mathbf{R}_0, \quad {}_i\boldsymbol{\omega}_i = {}_{i-1}\boldsymbol{\omega}_{i-1}, \quad {}_i\dot{\boldsymbol{\omega}}_i = {}_{i-1}\dot{\boldsymbol{\omega}}_{i-1}, \quad (6)$$

$${}_0\mathbf{r}_i = {}_0\mathbf{r}_{i-1} + {}_{i-1}\mathbf{R}_0^\top (\mathbf{e}_i s_i), \quad (7)$$

$${}_0\mathbf{v}_i = {}_0\mathbf{v}_{i-1} + {}_{i-1}\mathbf{R}_0^\top (\mathbf{e}_i \dot{s}_i + {}_{i-1}\boldsymbol{\omega}_{i-1} \times \mathbf{e}_i s_i), \quad (8)$$

$${}_0\mathbf{a}_i = {}_0\mathbf{a}_{i-1} + {}_{i-1}\mathbf{R}_0^\top (\mathbf{e}_i \ddot{s}_i + {}_{i-1}\dot{\boldsymbol{\omega}}_{i-1} \times \mathbf{e}_i s_i + {}_{i-1}\boldsymbol{\omega}_{i-1} \times (\mathbf{e}_i \dot{s}_i + {}_{i-1}\boldsymbol{\omega}_{i-1} \times \mathbf{e}_i s_i)). \quad (9)$$

For a rigid translational offset,  $\dot{s}_i = \ddot{s}_i = 0$ , reducing velocity and acceleration to the centripetal and Euler terms caused by the rotation of the parent frame.

### End-Effector Acceleration

Starting from a stationary base,  ${}_0\mathbf{v}_0 = {}_0\mathbf{a}_0 = {}_0\boldsymbol{\omega}_0 = {}_0\dot{\boldsymbol{\omega}}_0 = \mathbf{0}$ , the two recursions above are applied for each element in the chain. The result after the full chain is the absolute acceleration  ${}_0\mathbf{a}_P$  of the pilot's head in the inertial frame 0. For further usage, it is expressed in the pilot frame  $P$ .

### B. Human Perception Model

One primary driver of human motion perception is the vestibular system, which includes the otolith organs for perceiving translational accelerations and the semicircular canals for perceiving angular velocity [7]. Therefore, MCAs aim to produce realistic inertial cues, which consist of specific force  $\mathbf{f}$  and angular velocity  ${}_P\boldsymbol{\omega}_P$ , rather than replicating the full vehicle trajectory. These quantities are collected in the inertial-cue vector

$$\mathbf{y}_P = \begin{bmatrix} \mathbf{f} \\ {}_P\boldsymbol{\omega}_P \end{bmatrix}. \quad (10)$$

Specific force is the apparent acceleration sensed by the otoliths, which is the combination of translational acceleration  ${}_P\mathbf{a}_P$  and gravity  ${}_I\mathbf{g}_I$  acting in the pilot frame [7]:

$$\mathbf{f} = {}_P\mathbf{a}_P - {}_P\mathbf{R}_{II}\mathbf{g}_I. \quad (11)$$

The rotation matrix  ${}_P\mathbf{R}_I$  maps vector components from the inertial frame  $I$  to the pilot frame  $P$ .

Linear time-invariant systems can be used to approximate the vestibular system. Including these systems in the MPC-based MCA only represents the perceived inertial cues, rather than the entire vehicle trajectory. According to Telban and Cardullo [7], the following transfer functions can be used component-wise to describe the relationship between the perceived force  $\hat{\mathbf{f}}$  and the specific force  $\mathbf{f}$ , and between the perceived angular velocity  $\hat{\boldsymbol{\omega}}$  and the angular velocity  $\boldsymbol{\omega}$ :

$$\frac{\hat{\mathbf{f}}}{\mathbf{f}} = 0.4 \frac{10s + 1}{(5s + 1)(0.016s + 1)}, \quad (12)$$

$$\frac{\hat{\boldsymbol{\omega}}}{\boldsymbol{\omega}} = 5.73 \frac{80s^2}{(80s + 1)(5.73s + 1)}. \quad (13)$$

These transfer functions can be reformulated equivalently as a linear state-space model. By introducing the perception state vector  $\mathbf{x}_P \in \mathbb{R}^{12}$ , the perception model and the perceived inertial-cue vector are given by

$$\dot{\mathbf{x}}_P = \mathbf{A}_P\mathbf{x}_P + \mathbf{B}_P\mathbf{y}_P, \quad \hat{\mathbf{y}}_P = \mathbf{C}_P\mathbf{x}_P + \mathbf{D}_P\mathbf{y}_P. \quad (14)$$

### C. MPC-based Motion Cueing Algorithm

The purpose of the MPC-based motion cueing algorithm is to generate trajectories that represent the inertial cues generated by a flight dynamics model within the workspace and the actuator constraints of the motion platform. MPC is a control approach in which an optimal control problem (OCP) is solved repeatedly over a finite prediction horizon [12]. At each time step, a model is used to predict the future behavior of the system and to determine the control action that best satisfies the chosen objective while respecting the system constraints. Only the first control input is applied before the optimization is repeated at the next time step using updated system information.

The finite-horizon OCP is defined as

$$\begin{aligned} \min_{\mathbf{x}(\cdot), \mathbf{u}(\cdot)} \quad & \int_{t_k}^{t_k+T_h} l(\mathbf{x}(t), \mathbf{u}(t))dt + m(\mathbf{x}(t_k + T_h)) \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \\ & \mathbf{x}(t_k) = \mathbf{x}_k, \\ & \mathbf{x}(t) \in [\mathbf{x}_{min}, \mathbf{x}_{max}], \\ & \mathbf{u}(t) \in [\mathbf{u}_{min}, \mathbf{u}_{max}], \\ & t \in [t_k, t_k + T_h], \end{aligned} \quad (15)$$

where  $T_h$  is the prediction horizon. The state vector  $\mathbf{x}$  combines the motion system state and the perception state vector, and the input vector  $\mathbf{u}$  consists of the motion system input:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_s \\ \mathbf{x}_p \end{bmatrix} \in \mathbb{R}^{26}, \quad \mathbf{u} = \mathbf{u}_s \in \mathbb{R}^7. \quad (16)$$

Actuator constraints and mechanical limits of the motion platform are enforced through state constraints between  $\mathbf{x}_{min}$  and  $\mathbf{x}_{max}$  and through input constraints between  $\mathbf{u}_{min}$  and  $\mathbf{u}_{max}$ . The state constraints are applied only to the motion system states  $\mathbf{x}_s$ . The system dynamics  $\dot{\mathbf{x}}$  are obtained by combining the motion platform dynamics from Eq. (1) with the state-space representation of the perception model from Eq. (14):

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{x}}_s \\ \dot{\mathbf{x}}_p \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \\ A_p \mathbf{x}_p + B_p \mathbf{y}_p \end{bmatrix}. \quad (17)$$

The cost function consists of an integral cost

$$l(\mathbf{x}(t), \mathbf{u}(t)) = \Delta \mathbf{y}^\top(t) \mathbf{W}_y(t) \Delta \mathbf{y}(t) + \mathbf{u}^\top(t) \mathbf{W}_u \mathbf{u}(t) + \Delta \mathbf{q}^\top(t) \mathbf{W}_q \Delta \mathbf{q}(t), \quad (18)$$

and a terminal cost

$$m(\mathbf{x}(t_k + T_h)) = \Delta \mathbf{x}^\top(t_k + T_h) \mathbf{W}_x \Delta \mathbf{x}(t_k + T_h). \quad (19)$$

The integral cost is composed of three terms with different objectives. First, the tracking error between the actual perceived inertial-cue vector from Eq. (14) and the desired perceived inertial-cue vector  $\hat{\mathbf{y}}_{p,des}$  is penalized, with

$$\Delta \mathbf{y} = \hat{\mathbf{y}}_p - \hat{\mathbf{y}}_{p,des}. \quad (20)$$

This first term represents the primary objective of the OCP, since the MCA aims to reproduce the desired perceived inertial cues as closely as possible. Second, the control inputs are penalized in order to reduce control effort and avoid unnecessarily aggressive platform motion. And Third, deviations from a neutral platform position  $\mathbf{q}_0$  are penalized, with

$$\Delta \mathbf{q} = \mathbf{q} - \mathbf{q}_0, \quad (21)$$

which keeps the simulator close to its central workspace and supports washout behavior by creating freedom for subsequent motion commands.

Finally, the terminal cost penalizes the deviation of the simulator state at the end of the prediction horizon from a desired terminal state  $\mathbf{x}_{s,0}$ , which corresponds to the neutral joint configuration  $\mathbf{q}_0$  and zero joint velocities, defined as

$$\Delta \mathbf{x} = \mathbf{x}_s(t_k + T_h) - \mathbf{x}_{s,0}. \quad (22)$$

The terminal cost encourages the platform to return to a neutral state at the end of the horizon and thereby supports the stability of the MPC-based MCA.

To maintain real-time capability while maintaining sufficient look-ahead, the OCP is discretized using a non-uniform prediction horizon. The discretization is finer at the beginning of the horizon and becomes progressively coarser toward the end. This discretization reduces the number of prediction steps while still allowing a sufficiently long horizon for the terminal cost to encourage the motion simulator to return to a standstill toward its neutral state.

All terms of the cost function can be parameterized separately by the diagonal weight matrices  $\mathbf{W}_y(t)$ ,  $\mathbf{W}_u$ ,  $\mathbf{W}_q$ , and  $\mathbf{W}_x$ . In pilot-in-the-loop experiments, the future reference trajectory is generally not known in advance because it depends on the future control actions of the pilot. A common procedure for this case is to keep the pilot input constant over the entire prediction horizon. The same constant-horizon assumption is used in this work. However, the farther a prediction step lies in the future, the more uncertain this assumption becomes. To account for the increasing uncertainty of the predicted reference, the weighting matrix  $\mathbf{W}_y(t)$  is defined as the initial weighting matrix  $\mathbf{W}_{y,0}$  multiplied by a scaling factor that gradually decreases over the horizon according to

$$k_y(t) = k_{y,0} - (k_{y,0} - k_{y,f}) \left( \frac{t - t_k}{T_h} \right)^p, \quad t \in [t_k, t_k + T_h]. \quad (23)$$

Here,  $k_{y,0}$  denotes the initial scaling factor at the beginning of the prediction horizon,  $k_{y,f}$  denotes the final scaling factor at the end of the horizon, and  $p$  is a decay exponent that determines the shape of the reduction over time. In this way, the OCP places greater emphasis on near-future tracking accuracy, where the predicted reference is more reliable.

#### D. Reference Signal Scaling

In motion cueing, the desired translational accelerations and angular velocities that the motion platform needs to reproduce may exceed its physical capabilities due to actuator limits. To ensure feasibility, these reference signals are scaled to a realizable range. Although this scaling introduces an error in the magnitude of the inertial cues, such an error is unavoidable whenever the desired cues exceed the platform limits. For the MPC-based motion cueing algorithm, this scaling is not strictly required, but it simplifies the parameterization. Therefore, a smooth norm-based scaling is used, which reduces the magnitude of the input vector while preserving its direction. The scaling factor is defined as

$$w_{\text{scale}} = \frac{w_{\text{max}}}{\|\mathbf{r}\|} \tanh\left(\frac{\gamma \|\mathbf{r}\|}{w_{\text{max}}}\right), \quad (24)$$

and is applied uniformly to all components of the input vector  $\mathbf{r}$ , which represents either the desired translational acceleration or the desired angular velocity. For small  $\|\mathbf{r}\|$ , the mapping is approximately linear with gain  $\gamma$ , which determines both the small-signal gain and the smoothness of the transition to saturation. For larger magnitudes, the vector is smoothly compressed such that the magnitude of the scaled vector asymptotically approaches  $w_{\text{max}}$ , which specifies the maximum output magnitude.

#### E. Performance Metrics

Typical performance metrics for motion cueing algorithms are the root-mean-square error (RMSE) and the Pearson correlation coefficient (PCC) [8, 10]. In this work, both metrics are evaluated for each component of the perceived inertial-cue vector in order to assess how well the desired perceived motion cues are reproduced. For the hyperparameter tuning process described in Section III.E, modified versions of these metrics are used to obtain comparable contributions from the different cue channels, as well as a suitable scalar objective function for optimization. The RMSE quantifies the average magnitude of the deviation between two vectors  $\mathbf{v}_1$  and  $\mathbf{v}_2$  and is defined as

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{k=1}^N (v_{1,k} - v_{2,k})^2}. \quad (25)$$

The Pearson correlation coefficient is the ratio of the covariance of two vectors  $\mathbf{v}_1$  and  $\mathbf{v}_2$  to the product of their standard deviations:

$$\text{PCC} = \frac{\sum_{k=1}^N (v_{1,k} - \bar{v}_1)(v_{2,k} - \bar{v}_2)}{\sqrt{\sum_{k=1}^N (v_{1,k} - \bar{v}_1)^2} \sqrt{\sum_{k=1}^N (v_{2,k} - \bar{v}_2)^2}}. \quad (26)$$

Here,  $\bar{v}_1$  and  $\bar{v}_2$  denote the mean values of the elements of the vectors  $\mathbf{v}_1$  and  $\mathbf{v}_2$ , respectively. The PCC therefore represents a normalized measure of their linear relationship and is bounded between  $-1$  and  $1$ .

For use in the hyperparameter tuning cost function in Eq. 29, the RMSE of each cue channel is normalized in order to improve comparability between channels with different physical units and amplitudes. The normalized RMSE is defined as

$$\text{NRMSE} = \frac{\text{RMSE}}{s}, \quad (27)$$

where  $s$  denotes a normalization factor computed from the corresponding reference signal. We used the standard deviation as a normalization factor in this work. The channel-wise normalized RMSE values are collected in the vector  $\mathbf{e}_{\text{NRMSE}} \in \mathbb{R}^6$ .

Since the PCC lies in the interval  $[-1, 1]$ , it is shifted to the interval  $[0, 1]$  in order to obtain a nonnegative quantity that is more convenient for use in the cost function:

$$\text{SPCC} = \frac{1 + \text{PCC}}{2}. \quad (28)$$

The shifted PCC values of the different cue channels are grouped together in the vector  $\mathbf{e}_{\text{SPCC}} \in \mathbb{R}^6$ . For both the NRMSE and SPCC vectors, weighted mean values over all channels are computed, yielding  $\text{NRMSE}_{\text{wmean}}$  and  $\text{SPCC}_{\text{wmean}}$ .

respectively. The corresponding weighting vectors are defined as  $\mathbf{w}_{\text{NRMSE}} \in \mathbb{R}^6$  and  $\mathbf{w}_{\text{SPCC}} \in \mathbb{R}^6$ . These two weighted mean values are then used to form the scalar objective for hyperparameter tuning

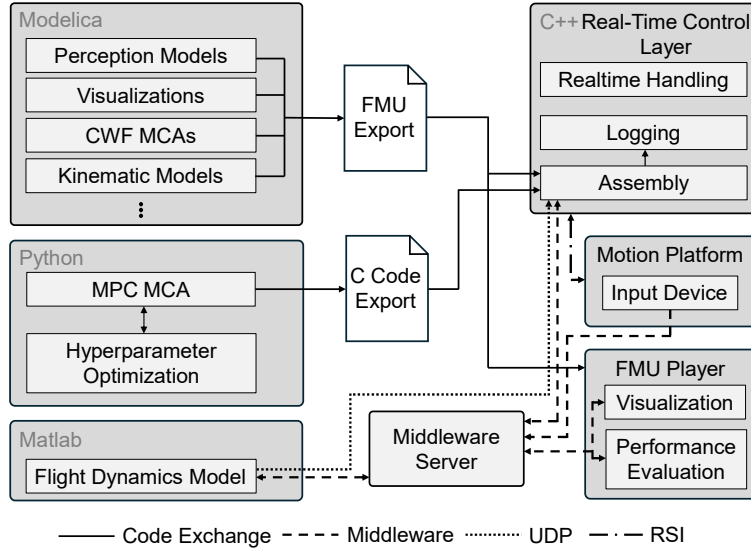
$$J = \alpha \text{NRMSE}_{\text{wmean}} + \beta (1 - \text{SPCC}_{\text{wmean}}). \quad (29)$$

The weights  $\alpha$  and  $\beta$  determine the relative influence of the tracking-error term and the correlation term. In this way, the optimization objective is intended to favor parameter sets that reduce the tracking error while maintaining the temporal shape of the desired perceived inertial cues.

The values of the parameters used are summarized in Table 1 in the Appendix.

### III. Framework Architecture

This section presents a modular framework for the development, implementation, and evaluation of MCAs. The main objective of the framework is to reduce the integration effort required when developing new MCAs for use with an existing motion simulator, or when individual components are exchanged. To achieve this, the framework is separated into reusable modules with clearly defined interfaces. This allows the different modules, such as the flight dynamics model, the MCA, visualization components, or the motion platform, to be replaced without significant changes to the software stack.



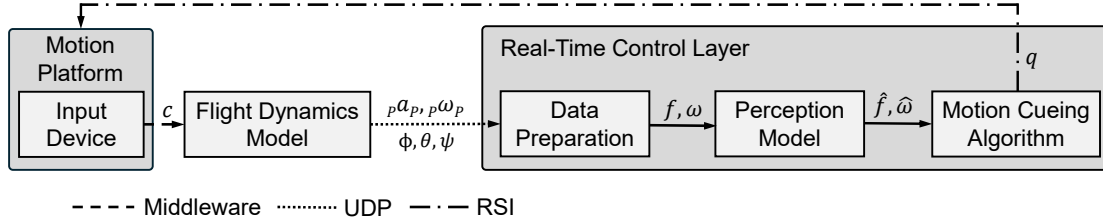
**Fig. 2 Overview of the motion cueing framework.**

An overview of the framework is shown in Fig. 2. It combines modules implemented in different programming languages and software environments that exchange data via middleware and a real-time control layer. This architecture can be used for interactive pilot-in-the-loop experiments, scripted playback of predefined trajectories, and simulation-only operation without the real motion platform.

#### A. Distributed Execution and Interfaces

The modules of the framework are executed in a distributed manner on multiple machines running different operating systems. All of these machines are interconnected via a local network. The communication between the modules within the framework can be divided into a real-time and a non-real-time part. Non-real-time communication is primarily used for visualization, performance evaluation, and pilot input and must provide sufficiently low latency so that no perceptible delay is introduced to the pilot during operation. Occasional missed updates at individual time steps are acceptable in this part of the framework, as they do not compromise overall functionality. In contrast, the real-time communication must satisfy the timing requirements of the robot control cycle, because missing a control cycle can trigger a fault and cause the motion platform to stop.

The non-real-time communication between modules is handled by ModelNet [13], a lightweight middleware developed for low-latency communication across distributed systems and different software environments. It is based on a publish–subscribe pattern in which clients publish data under predefined names and subscribe to the signals they require. A central server distributes the signals to all subscribed clients. This architecture enables flexible and efficient signal exchange without relying on a static configuration. Although ModelNet is designed for low latency, it is not the appropriate tool for the real-time critical path-planning tasks or the communication with the robot. For this reason, we employed a real-time control layer, which is presented in Sec. III.C.



**Fig. 3 Signal flow for flight simulation of the proposed motion-cueing framework.**

Fig. 3 provides an overview of the main signal flow for motion cueing and shows the modules with their signal interfaces that convert the pilot input  $c$  into motion commands  $q$  for the motion platform.

The first block is the *Input Device*, which represents the cockpit input devices of the motion platform. From there, the pilot commands  $c$  are transmitted to the *Flight Dynamics Model* via ModelNet. This model computes the translational acceleration  ${}_p a_P$  and the angular velocity  ${}_p \omega_P$  of the airplane. The flight dynamics model itself can be implemented in any software environment, provided that it is able to receive the pilot commands and it offers a UDP interface through which the computed acceleration  ${}_p a_P$ , angular velocity  ${}_p \omega_P$ , and Euler angles  $\phi$ ,  $\theta$ , and  $\psi$  of the pilot frame with respect to an inertial reference frame are transmitted.

The first module in the real-time control layer is the subsequent *Data Preparation* block, which transforms the translational acceleration using the Euler angles into the specific force  $f$  as defined in Eq. (11). The Euler angles are used to compute the rotation matrix  ${}_p R_I$  that transforms the gravity vector from the inertial frame to the pilot frame. This block also scales the specific force and angular velocity to a range that can be represented on the motion platform, as explained in Sec. II.D, and may perform additional preprocessing steps, such as coordinate transformations or unit conversions. This ensures the compatibility of existing flight dynamics models, if they cannot be modified to the desired interface.

The *Perception Model* receives the specific force and angular velocity and calculates the perceived force  $\hat{f}$  and the perceived angular velocity  $\hat{\omega}$  according to Eqs. (12) and (13). If the *Motion Cueing Algorithm* is designed to use the specific force  $f$  and the angular velocity  $\omega$  as inputs, the *Perception Model* block can be omitted.

The *Motion Cueing Algorithm* then computes trajectories that aim to reproduce the received perceptual reference signals as accurately as possible, within the physical limitations of the platform. The trajectory is then forwarded to the motion platform as joint angles  $q$ . Depending on the motion platform’s kinematic structure, the interface also changes, so the *Motion Cueing Algorithm* must be designed for a specific type of motion platform.

## B. Functional Mock-up Units

Several modules of the framework have been implemented in Modelica and exported as Functional Mock-up Units (FMUs) in accordance with the Functional Mock-up Interface (FMI) standard. Modelica is a declarative, equation-based modeling language for physical and technical systems that supports both textual and graphical modeling. The FMI standard provides a standardized way to package such models as executable units with defined parameters, inputs, and outputs. In this work, we used FMUs for the perception model, data preparation, performance evaluation, and visualization. They can also be set up for modules such as a CWF or a kinematic model of a motion platform.

The perception model and the signal processing modules were implemented using the Modelica Standard Library. For the implementation of the visualization and performance evaluation modules, the DLR Visualization 2 Library [14] was utilized. Visualization includes a third-person view of the aircraft, the outside view presented to the pilot in the cockpit, and the cockpit flight instruments. It also covers operator control panels for the motion simulator and the visualization of the motion simulator itself for simulation purposes. The performance evaluation module displays the

desired perceived force and perceived angular velocity and compares them with the actual values, allowing direct assessment of the MCA's behavior.

To execute FMU-based modules on the distributed systems, the framework uses an FMU Player. The FMU Player is a command-line tool implemented as a Python executable based on the `fmpy` package, which enables flexible execution of FMUs on a target system. However, FMUs are executed not only with the FMU Player but also directly in the real-time control layer, which is described in the following subsection.

### C. Real-Time Control Layer

The real-time backbone of the framework is the Modular Robot Controller (MoRC) [15], which was extended in this work to fit the requirements for motion cueing and flight simulation. MoRC was developed as a vendor-agnostic high-performance robot control platform for industrial robots. Its software is built around interchangeable control modules that follow the Functional Mock-up Interface (FMI) standard and can be dynamically connected. Each module exposes defined input and output ports that allow data exchange with other modules. The connections between modules are specified in a Lua script, forming what is referred to as an Assembly. Multiple Assemblies can be defined and executed sequentially if needed. This makes it possible to configure complete control loops by instantiating modules and linking their ports, without modifying the internal implementation of the modules themselves. MoRC supports real-time execution of Assemblies both on physical hardware and in a fully simulated environment, enabling thorough testing without requiring hardware activation.

For the present work, MoRC was extended by a User Datagram Protocol (UDP) interface and an RSI interface. The UDP interface is used to receive the reference trajectory from the flight-dynamics model, while the RSI interface provides the real-time connection to the RMS. In this way, MoRC serves as the execution layer that links the motion-cueing algorithm to the physical motion platform.

```

1  -- create modules
2  reference = ReferenceTrajectoryUDP(6)
3  dataPreparation = FMU("dataPreparation.fmu", 6)
4  MCA = MPC_MCA(12)
5  motionSimulator = RMS()
6  log = Logger("LogFile.h5", 6)
7
8  vis = FMU("sendVisualizationData.fmu", 6)
9  vis:setParameter("port", 1234)
10
11 -- connect modules
12 connect(reference, "a_x", reference, "a[1]")
13 connect(reference, "f_perceived[1]", MCA, "f_ref[1]")
14 connect(MCA, "q1", motionSimulator, "q1_d")
15 connect(motionSimulator, "q1", log, "q1_rms")
16 connect(motionSimulator, "q1", vis, "q1_rms")

```

**Listing 1 Simplified configuration file for MPC-based motion cueing.**

Listing 1 shows a simplified configuration for an MPC-based motion-cueing pipeline, as shown in Fig. 3. The function arguments indicate the update times of the corresponding modules in milliseconds. In the upper part of the listing, several modules are instantiated, including a UDP interface receiving the reference trajectory from the flight-dynamics model (`reference`), a data preparation FMU including a perception model (`dataPreparation`), the MCA module (`MCA`), a data logger (`log`), the motion simulator (`motionSimulator`), and a visualization module (`vis`). For the visualization module, it is shown how a parameter for the FMU can be set. In the lower part of the listing, the signal routing between these modules is defined through output-to-input connections. The reference acceleration is first passed through the perception model, the resulting perceived force is provided to the MCA, and the computed joint command is forwarded to the motion simulator. The simulator output is then logged and simultaneously sent to the visualization module. This simplified example demonstrates how an entire real-time motion-cueing pipeline can be assembled from modular components within a single Lua configuration script.

#### D. MPC-Based MCA Implementation

The MPC-based MCA used to demonstrate the framework is implemented using the acados software package [16]. The acados software provides a collection of solvers for fast embedded optimal control. Implemented in C, it provides Python and Matlab interfaces, enabling the rapid prototyping and deployment of optimal control problems. The system models are formulated using CasADi [17], and high-performance numerical backends such as BLASFEO [18] and the quadratic-programming solver HPIPM [19] are used to achieve real-time performance, making it well-suited for motion-cueing applications.

In order to integrate the MCA into the framework, the equations necessary for setting up the OCP are presented in Sec. II are formulated in CasADi. The CasADi expressions are then used by the Python interface of acados to define the OCP. In acados, the OCP is formulated using a multiple-shooting approach, where the prediction horizon is divided into discrete shooting intervals, with the system dynamics defined as constraints between each interval. The resulting nonlinear MPC problem is solved using the RTI scheme, which performs one sequential quadratic programming iteration per sampling instant, rather than carrying out multiple SQP iterations until convergence, thereby enabling fast online optimization [20]. HPIPM is used to solve the quadratic programming (QP) subproblem arising within each RTI step. Based on this description, acados generates C code containing a set of functions that initialize the OCP, update its parameters, solve it, and return the solution. To enable execution in MoRC, we implemented a dedicated module that integrates the C code generated by acados into the framework and uses it to execute the MPC controller in real time. This procedure provides the flexibility to formulate and modify the entire OCP in Python, eliminating the need to manually implement solver logic and low-level numerical routines in C.

The numerical values of the parameters used in acados are summarized in Table 2 in the Appendix.

#### E. Hyperparameter Tuning

Hyperparameter tuning is carried out offline using the Optuna framework [21]. The goal is to identify suitable values for the diagonal entries of the weighting matrices in the cost functions from Eqs. (18) and (19) without relying on time-consuming manual tuning. For automated tuning, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) sampler [22] is used, because MPC weight tuning is a non-convex optimization problem with strongly coupled parameters, that can be treated as a black-box problem. The CMA-ES sampler is an evolutionary optimization method for continuous black-box problems that samples candidate solutions from a multivariate normal distribution and iteratively updates its mean, covariance matrix, and step size based on the best-performing samples. This means that, for each generation, a set of parameter vectors with cost weights is sampled and evaluated in an MCA simulation. For the simulation of the MCA, a representative reference trajectory must be used that covers all the maneuvers to be performed during the flight simulation. Failure to cover all maneuvers can lead to the MCA becoming unstable. Using the simulation results, the scalar objective in Eq. (29) is calculated for each parameter vector. These objectives are then used to update the CMA-ES sampler for the next generation. In addition to the objective, the implementation includes a small regularization term that penalizes an excessively large spread of cost weight parameter values across the four diagonal matrices. This discourages highly unbalanced parameter sets with similar performance but unnecessarily large differences in weight magnitude.

The sampling procedure is implemented with the ask–tell interface of Optuna. This separates the generation of new parameter sets from their numerical evaluation and thereby facilitates parallel execution. A single coordinator process owns the study and samples new trials, while a pool of worker processes evaluates the sampled parameter sets independently. After each worker has returned its result, only the coordinator performs the corresponding tell step and updates the Optuna study. To avoid race conditions during solver generation and loading, the acados solver artifacts are prepared once before the worker processes are started. The sampled trials are processed in batches that are aligned with the population size of the CMA-ES sampler.

The optimization variables are parameterized in logarithmic space. Optuna samples the base-10 logarithms of the diagonal entries, and the corresponding weight matrices are obtained by elementwise exponentiation. This enables an efficient search over several orders of magnitude.

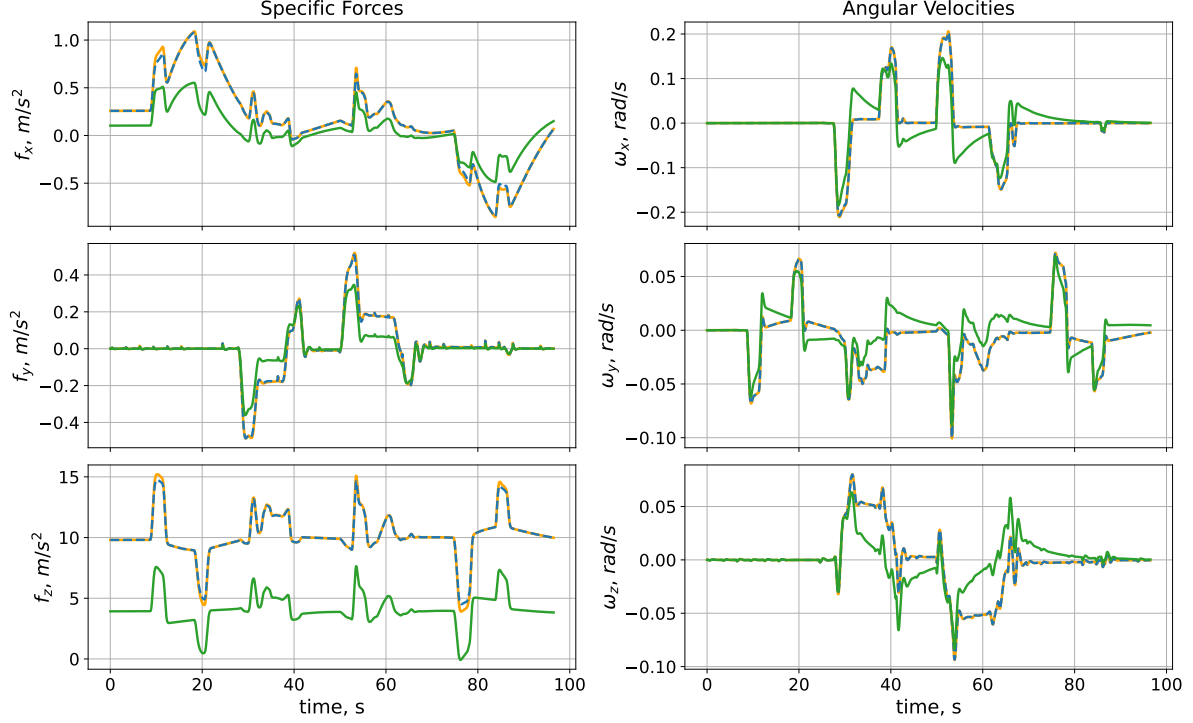
The numerical values used for Optuna are summarized in Table 3 in the Appendix.

### IV. Results and Discussion

To demonstrate the proposed framework, a nonlinear MPC-based motion cueing algorithm was evaluated on a flight maneuver consisting of a short climb phase, followed by a left turn, a subsequent right turn, and a short descent phase. The maneuver was generated with a flight dynamics model of the Diamond DA42 and executed on the DLR Robotic

Motion Simulator using the framework described in Sec. III.

Figure 4 shows how the reference cues are transformed before they are forwarded to the MCA. The inertial cues generated by the flight dynamics model are shown in orange. As a first step, these signals are scaled to a feasible range to limit cues beyond the capabilities of the motion platform. The resulting scaled cues are shown in blue. Because the reference maneuver is already close to the scaling limits, mainly a few peaks of the specific-force signals are limited, whereas large parts of the trajectory remain unaffected. In a second step, the perception model transforms the scaled signals into perceived inertial cues that better reflect human motion perception. The resulting perceived inertial cues, shown in green, are then used as the reference for the MPC-based MCA.



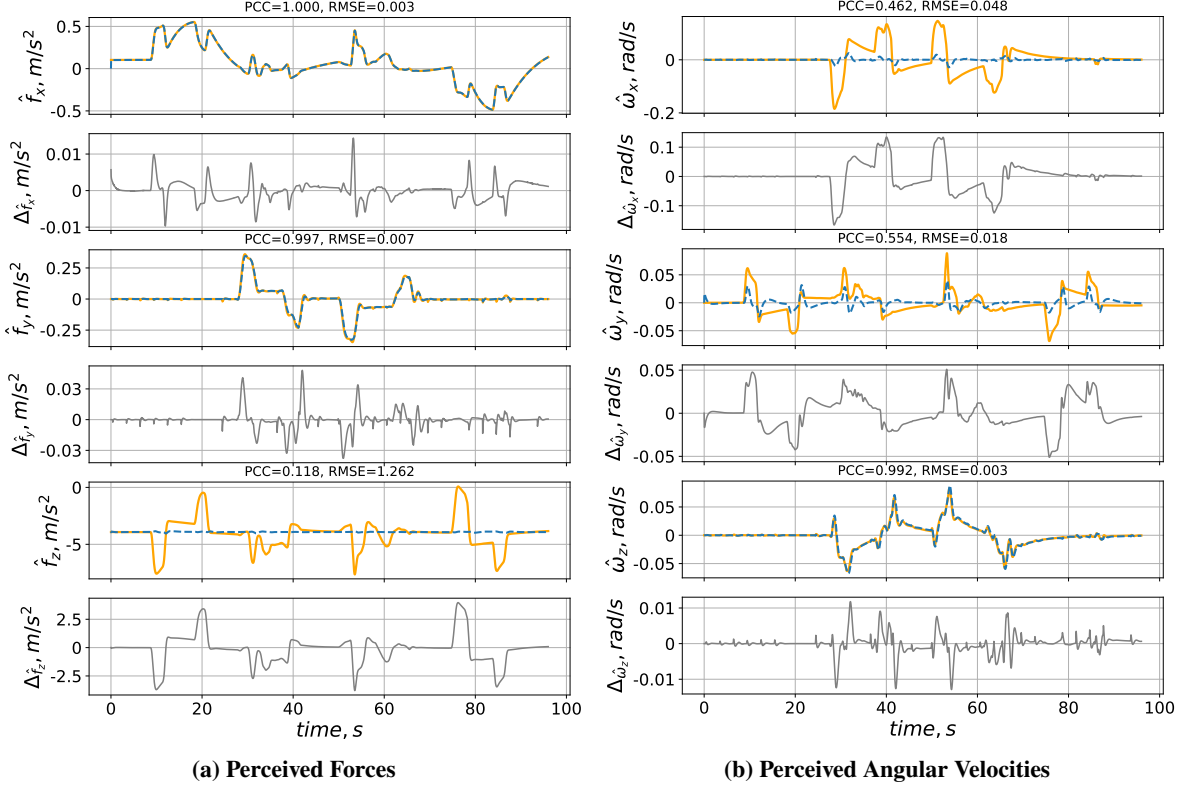
**Fig. 4** Comparison of original inertial cues (—), scaled inertial cues(---), and perceived inertial cues(—), resulting from the perception model. The perceived inertial cues are used as reference trajectory by the MPC-based MCA.

In order for the MCA to accurately track the reference, the cost weights must be appropriately tuned. These weights were obtained through the hyperparameter tuning process described in Sec. III.E. The reference trajectory used for tuning was based on the same maneuvers as those shown in Fig. 4, but it was generated separately and therefore was not identical. The tuning process involved evaluating approximately 2600 trials with 32 parallel workers, taking roughly 3 h.

To assess the tracking performance of the proposed MCA, Fig. 5 compares the desired and actual perceived inertial cues obtained using the framework and shows the corresponding tracking errors. The Pearson correlation coefficient (PCC) and the root-mean-square error (RMSE) are shown as well.

Overall, the plots show good tracking of the perceived forces in the  $x$ - and  $y$ -directions, as well as the perceived angular velocity about the  $z$ -axis. The remaining channels show weaker performance, in particular, the specific force in the  $z$ -direction. This can be explained by the fact that sustained vertical acceleration is hard to generate on most motion platforms. Unlike in the lateral and longitudinal axes, no tilt-coordination mechanism exists that can reorient gravity to emulate a steady acceleration in the vertical direction. As a result, vertical specific force cues must be generated primarily by real heave motion, which is limited to very short cues by the available workspace and actuator constraints of the platform. Although increasing the weight of vertical cue tracking in the cost function can improve  $z$ -direction tracking, it leads to a substantial decline in tracking of most other cue components. With the current hyperparameter-tuning objective, it was not possible to identify a compromise parameter set that modestly improves  $z$ -direction tracking without significantly degrading overall tracking performance. Potential improvements could be

achieved by modifying the cost function or by using a different hyperparameter-tuning objective. Another possible improvement would be to avoid a constant prediction horizon with a fixed assumed input and instead use a known future reference trajectory [10]. In pilot-in-the-loop flight simulation, however, this would require a pilot predictor. The weaker tracking of the perceived angular velocity about the x- and y-axes can be explained by the fact that larger roll and pitch rotations introduce cross-coupled inertial cues. The MCA must therefore trade off the reproduction of these angular motions against the tracking of the other cue components and the platform constraints.



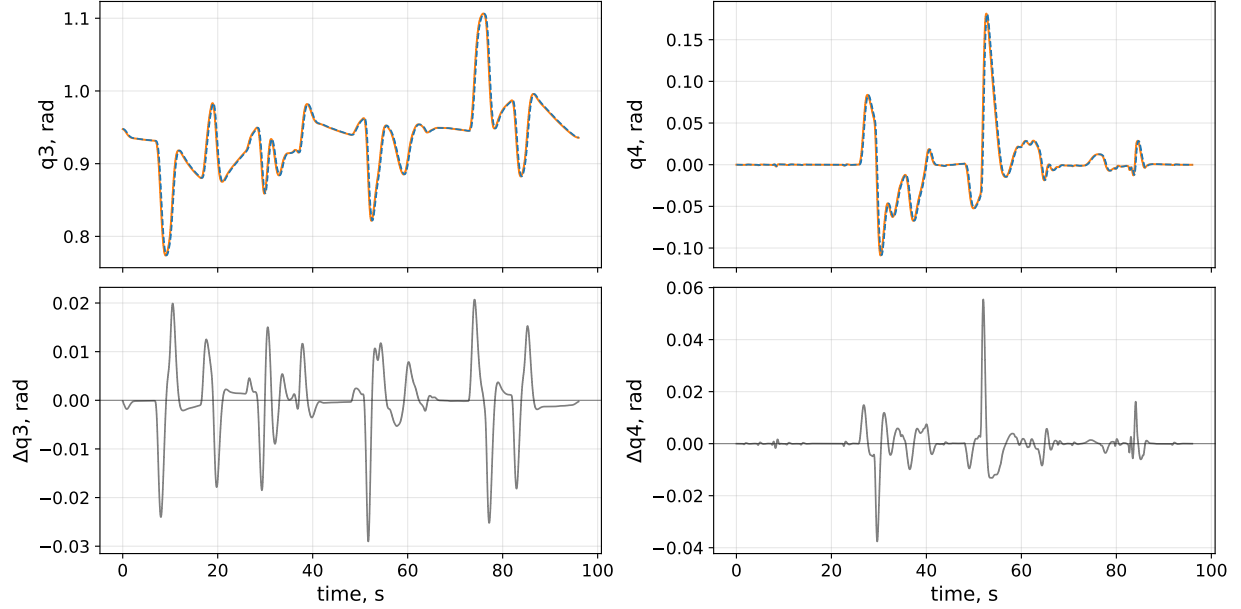
**Fig. 5** The plots compare the perceived forces and perceived angular velocities generated by the MCA (---) with the reference signals from the flight dynamics model (—). The corresponding tracking errors (—) are shown below.

To demonstrate how the desired MCA motion is reproduced on the RMS, Fig. 6 compares the desired and measured trajectories for axes 3 and 4 of the industrial robot. The results show a small, consistent phase lag in the actual joint positions. This lag is mainly caused by the internal robot position controller, whose dynamics are not captured by the simplified system model used in the OCP. As a result, a certain model mismatch between the commanded and realized joint motion is unavoidable. However, the tracking error, quantified by the RMSE, remains within 1.5%-3.2%, indicating that the motion simulator is able to track the desired joint motion with sufficient accuracy for the considered motion-cueing experiment.

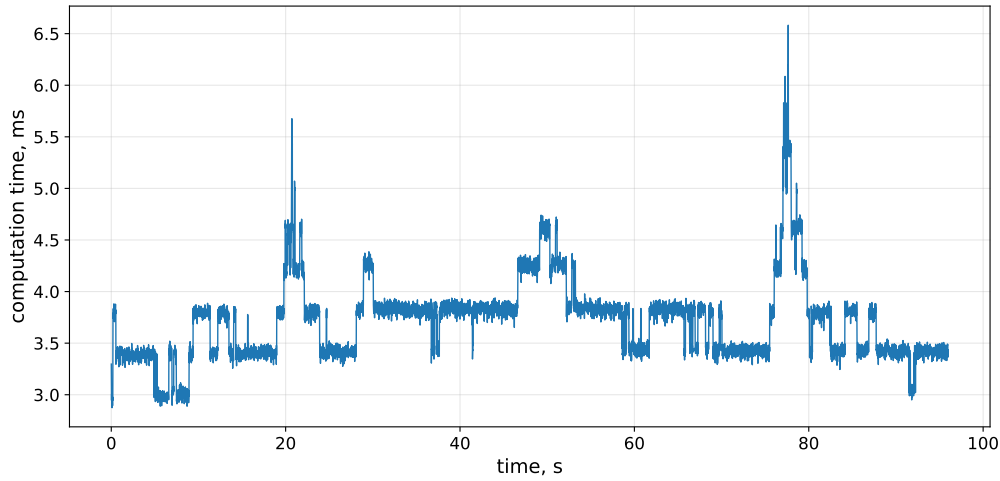
In addition to cue-tracking performance, Fig. 7 shows the computation time of the MPC-based MCA along the maneuver. The controller was executed within the 12 ms robot cycle time of the RMS. Over the complete trajectory, the average computation time was 3.7 ms, while the maximum computation time was 6.6 ms. The computational load varies over the maneuver, but remains well below the available cycle time in all shown phases. This means that the framework is capable of executing the MPC-based MCA in real time on the RMS.

The results highlight the remaining challenges of MPC-based motion cueing for flight simulation, especially with respect to the reproduction of vertical specific force and cross-coupled rotational cues. It is planned to carry out experiments with human pilots to further evaluate the performance of the MPC-based MCA and to investigate how the remaining tracking errors affect the pilot's perception and performance during flight simulation. Additionally, we plan to implement a mechanism that takes the perception threshold of rotational cues into account, which is expected to

improve the finding of optimal trade-offs between reproducing rotational cues and tracking of other cue components.



**Fig. 6** Desired (—) and measured (---) joint angles of axes 3 and 4 of the RMS, together with the corresponding tracking errors (—) shown in the lower panels.



**Fig. 7** Computation time required for one execution step of the MCA over the simulation time.

## V. Conclusion

In this paper, we presented a modular framework for the development, implementation, and evaluation of motion cueing algorithms on real motion simulators. The framework combines reusable software modules, standardized interfaces, integrated visualization and performance evaluation, automatic hyperparameter tuning, and a real-time execution layer for the motion platform. The modular architecture supports interactive pilot-in-the-loop experiments, scripted trajectory playback, and simulation-only operation, making it applicable throughout the entire development cycle.

The framework was demonstrated using a nonlinear MPC-based motion cueing algorithm on the DLR Robotic Motion Simulator, with a flight dynamics model of the Diamond DA42 aircraft. The results showed that the proposed

framework is capable of executing the nonlinear MPC-based MCA on the motion platform in real time. The desired joint motions were tracked with sufficient accuracy, and the cycle time of the controller remained well within the available robot cycle time of 12 ms throughout the considered maneuver. The achieved cue-tracking performance indicates that relevant aspects of the maneuver can be reproduced successfully, although limitations remain in reproducing vertical specific force and cross-coupled rotational cues. To the best of our knowledge, this constitutes the first documented real-time implementation of a nonlinear MPC-based motion cueing algorithm for flight simulation on a physical motion platform.

The limitations in the reproduction of motion cues highlight important areas for future research, where the proposed framework can be used as a platform for implementing and testing improved MCAs. Potential improvements include modifications to the cost function, alternative hyperparameter-tuning objectives, and the integration of pilot prediction to improve motion fidelity. Future work will also include pilot-in-the-loop experiments to evaluate the perceptual quality of the generated motion cues and to investigate how the remaining tracking errors influence pilot performance and workload. Overall, the proposed framework provides both a practical environment for real-time MCA development and a foundation for further research on advanced motion-cueing methods for flight simulation.

## Appendix

### Parameters

**Table 1 OCP parameters of the MPC-based motion cueing algorithm.**

| Symbol                      | Value                                                                                                                                               |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| $\mathbf{W}_{y,0}$          | diag(26.779554, 7.305349, 0.051689, 6.061932, 45.495957, 65.494682)                                                                                 |
| $\mathbf{W}_u$              | diag(3.027078, 0.821319, 1.214957, 0.131568, 0.068770, 0.695079, 1.384661)                                                                          |
| $\mathbf{W}_q$              | diag(0.0056258, 0.0062690, 0.0054964, 0.0051905, 0.007207, 0.0037284, 0.0066842)                                                                    |
| $\mathbf{W}_x$              | diag(0.004835, 0.089077, 0.018335, 0.029767, 0.014940, 0.185587, 0.011287,<br>0.330238, 0.486399, 0.254843, 0.201334, 0.213412, 0.136533, 0.191480) |
| $q_{\min}$                  | (−6.283185, −2.234021, −0.523599, −1.553343, −1.012291, −3.141593, −8.380000)                                                                       |
| $q_{\max}$                  | (6.283185, −0.733038, 1.343904, 1.553343, 1.012291, 3.141593, −0.580000)                                                                            |
| $\dot{q}_{\min}$            | (−1.221730, −0.977384, −1.204277, −1.326450, −1.326450, −2.094395, −1.500000)                                                                       |
| $\dot{q}_{\max}$            | (1.221730, 0.977384, 1.204277, 1.326450, 1.326450, 2.094395, 1.500000)                                                                              |
| $\ddot{q}_{\min}$           | (−1.692969, −1.256637, −2.234021, −0.733038, −1.658063, −1.169371, −0.800000)                                                                       |
| $\ddot{q}_{\max}$           | (1.692969, 1.256637, 2.234021, 0.733038, 1.658063, 1.169371, 0.800000)                                                                              |
| $T_h$                       | 6 s                                                                                                                                                 |
| $k_{y,0}$                   | 1.0                                                                                                                                                 |
| $k_{y,f}$                   | 0.4                                                                                                                                                 |
| $p$                         | 0.5                                                                                                                                                 |
| $w_{\max,f}$                | 10.0                                                                                                                                                |
| $\gamma_f$                  | 1.0                                                                                                                                                 |
| $w_{\max,\omega}$           | 1.0                                                                                                                                                 |
| $\gamma_\omega$             | 1.0                                                                                                                                                 |
| $\mathbf{w}_{\text{NRMSE}}$ | (1.0, 1.0, 5.0, 1.0, 1.0, 1.0)                                                                                                                      |
| $\mathbf{w}_{\text{SPCC}}$  | (2.0, 3.0, 1.0, 10.0, 10.0, 5.0)                                                                                                                    |
| $\alpha$                    | 1.0                                                                                                                                                 |
| $\beta$                     | 0.5                                                                                                                                                 |

**Table 2 Parameters of acados for the MPC-based MCA.**

| Manuscript quantity                | Acados parameter                      | Value                                                                                                                     |
|------------------------------------|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| Dimension of $\mathbf{x}$          | model.nx                              | 26                                                                                                                        |
| Dimension of $\mathbf{u}$          | model.nu                              | 7                                                                                                                         |
| Dimension of parameters            | model.np                              | 40                                                                                                                        |
| Stage cost representation          | ocp.cost.cost_type                    | EXTERNAL                                                                                                                  |
| Terminal cost representation       | ocp.cost.cost_type_e                  | EXTERNAL                                                                                                                  |
| Prediction horizon $T_h$           | ocp.solver_options.tf                 | 6.0 s                                                                                                                     |
| Number of shooting intervals $N$   | ocp.solver_options.N_horizon          | 30                                                                                                                        |
| Nonuniform discretization of $T_h$ | ocp.solver_options.shooting_nodes     | First 10 intervals: 0.024 s, next 10: 0.048 s, next 9: 0.096 s, final interval adjusted to 4.416 s to reach $T_h = 6.0$ s |
| Nonlinear-program solver           | ocp.solver_options.nlp_solver_type    | SQP_RTI                                                                                                                   |
| Quadratic-program solver           | ocp.solver_options.qp_solver          | PARTIAL_CONDENSING_HPIPM                                                                                                  |
| HPIPM configuration                | ocp.solver_options.hpipm_mode         | ROBUST                                                                                                                    |
| Hessian approximation              | ocp.solver_options.hessian_approx     | EXACT                                                                                                                     |
| Regularization                     | ocp.solver_options.regularize_method  | PROJECT                                                                                                                   |
| QP solver tolerance                | ocp.solver_options.qp_tol             | $10^{-4}$                                                                                                                 |
| Maximum QP iterations              | ocp.solver_options.qp_solver_iter_max | 20                                                                                                                        |
| Step size                          |                                       | 12 ms                                                                                                                     |

**Table 3 Optuna parameters for hyperparameter tuning.**

| Explanation                        | Optuna setting         | Value                                                                                                                                                                                                                                                                                           |
|------------------------------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Search interval for $\mathbf{W}_y$ |                        | $[-2, 2]$                                                                                                                                                                                                                                                                                       |
| Search interval for $\mathbf{W}_u$ |                        | $[-3, 1]$                                                                                                                                                                                                                                                                                       |
| Search interval for $\mathbf{W}_q$ |                        | $[-3, 1]$                                                                                                                                                                                                                                                                                       |
| Search interval for $\mathbf{W}_x$ |                        | $[-3, 1]$                                                                                                                                                                                                                                                                                       |
| Initial parameters                 |                        | diag( $\mathbf{W}_y$ ) = (1.0, 0.5, 0.1, 1.0, 1.0, 1.0)<br>diag( $\mathbf{W}_u$ ) = (0.5, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)<br>diag( $\mathbf{W}_q$ ) = (0.4, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)<br>diag( $\mathbf{W}_x$ ) = (0.4, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1) |
| Sampler                            |                        | CmaEsSampler                                                                                                                                                                                                                                                                                    |
| Initial standard deviation         | sigma0                 | 0.65                                                                                                                                                                                                                                                                                            |
| Population size                    | popsiz                 | 32                                                                                                                                                                                                                                                                                              |
| Random seed                        | seed                   | 42                                                                                                                                                                                                                                                                                              |
| Pruned-trial handling              | consider_pruned_trials | False                                                                                                                                                                                                                                                                                           |
| Storage database                   | storage                | PostgreSQL                                                                                                                                                                                                                                                                                      |

## References

- [1] Reid, L. D., and Nahon, M. A., “Flight Simulation Motion-Base Drive Algorithms: Part 1. Developing and Testing Equations,” UTIAS Report No. 296, University of Toronto, 1985.
- [2] Beghi, A., Bruschetta, M., and Maran, F., “A Real Time Implementation of MPC Based Motion Cueing Strategy for Driving Simulators,” *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, IEEE, Maui, HI, USA, 2012, pp. 6340–6345. <https://doi.org/10.1109/CDC.2012.6426119>.
- [3] Bruschetta, M., Maran, F., and Beghi, A., “A Nonlinear, MPC-Based Motion Cueing Algorithm for a High-Performance, Nine-DOF Dynamic Simulator Platform,” *IEEE Transactions on Control Systems Technology*, Vol. 25, No. 2, 2017, pp. 686–694. <https://doi.org/10.1109/TCST.2016.2560120>.
- [4] Katliar, M., Drop, F. M., Teufel, H., Diehl, M., and Bülthoff, H. H., “Real-Time Nonlinear Model Predictive Control of a Motion Simulator Based on a 8-DOF Serial Robot,” *2018 European Control Conference (ECC)*, IEEE, Limassol, 2018, pp. 1529–1535. <https://doi.org/10.23919/ECC.2018.8550041>.
- [5] Lamprecht, A., Steffen, D., Nagel, K., Haecker, J., and Graichen, K., “Online Model Predictive Motion Cueing With Real-Time Driver Prediction,” *IEEE Transactions on Intelligent Transportation Systems*, Vol. 23, No. 8, 2022, pp. 12414–12428. <https://doi.org/10.1109/TITS.2021.3114003>.

- [6] Scheidel, H., Asadi, H., Bellmann, T., Seefried, A., Mohamed, S., and Nahavandi, S., “A Deep Reinforcement Learning Based Motion Cueing Algorithm for Vehicle Driving Simulation,” *IEEE Transactions on Vehicular Technology*, Vol. 73, No. 7, 2024, pp. 9696–9705. <https://doi.org/10.1109/TVT.2024.3375941>.
- [7] Telban, R. J., and Cardullo, F. M., “Motion Cueing Algorithm Development: Human-Centered Linear and Nonlinear Approaches,” *New York*, Vol. NASA TechReport CR-2005-213747, 2005.
- [8] Jacumet, R., Kolff, M., Venrooij, J., Schwienbacher, M., Wagner, S., Wollherr, D., Leibold, M., Pool, D. M., and Mulder, M., “A Review on Optimization-Based Motion Cueing Algorithms for Driving Simulation,” *IEEE Transactions on Intelligent Transportation Systems*, 2026, pp. 1–25. <https://doi.org/10.1109/TITS.2026.3661819>.
- [9] Tong, Y., Zhou, H., Wu, Z., Li, Q., and Lu, B., “Model Predictive Control Based Washout Algorithm Design for Flight Simulator Upset Prevention and Recovery Training,” *Aerospace*, Vol. 10, No. 10, 2023, p. 886. <https://doi.org/10.3390/aerospace10100886>.
- [10] Bakker, R. M., Stroosma, O., Pool, D. M., Van Paassen, M. M., and Mulder, M., “Comparison of Classical and Optimization-Based Motion Cueing for Simulating Aircraft Upset Maneuvers,” *AIAA SCITECH 2025 Forum*, American Institute of Aeronautics and Astronautics, Orlando, FL, 2025. <https://doi.org/10.2514/6.2025-0978>.
- [11] Zhao, J., Xu, Z., Wu, D., Cao, Y., and Xie, J., “Six-DoF Stewart Platform Motion Simulator Control Using Switchable Model Predictive Control,” , 2025. <https://doi.org/10.48550/ARXIV.2503.11300>.
- [12] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M., *Model Predictive Control: Theory, Computation, and Design*, 2<sup>nd</sup> ed., Nob Hill Publishing, 2017.
- [13] Seefried, A., Pollok, A., Kuchar, R., Hellerer, M., Leitner, M., Milz, D., Schallert, C., Kier, T., Looye, G., and Bellmann, T., “Multi-Domain Flight Simulation with the DLR Robotic Motion Simulator,” *2019 Spring Simulation Conference (SpringSim)*, IEEE, Tucson, AZ, USA, 2019, pp. 1–12. <https://doi.org/10.23919/SpringSim.2019.8732921>.
- [14] Kümper, S., Hellerer, M., and Bellmann, T., “DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning,” *14th Modelica Conference 2021*, 2021. <https://doi.org/10.3384/ecp21181197>.
- [15] Oldemeyer, C., Hellerer, M., Reiner, M., Thiele, B., Weber, P., and Bellmann, T., “MoRC—A Modular Robot Controller,” *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Yokohama, Japan, 2024, pp. 17643–17649. <https://doi.org/10.1109/ICRA57147.2024.10611553>.
- [16] Verschueren, R., Frison, G., Kouzoupis, D., Frey, J., Duijkeren, N. V., Zanelli, A., Novoselnik, B., Albin, T., Quirynen, R., and Diehl, M., “Acados—a Modular Open-Source Framework for Fast Embedded Optimal Control,” *Mathematical Programming Computation*, Vol. 14, No. 1, 2022, pp. 147–183. <https://doi.org/10.1007/s12532-021-00208-8>.
- [17] Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M., “CasADi: A Software Framework for Nonlinear Optimization and Optimal Control,” *Mathematical Programming Computation*, Vol. 11, No. 1, 2019, pp. 1–36. <https://doi.org/10.1007/s12532-018-0139-4>.
- [18] Frison, G., Kouzoupis, D., Sartor, T., Zanelli, A., and Diehl, M., “BLASFEO: Basic Linear Algebra Subroutines for Embedded Optimization,” *ACM Transactions on Mathematical Software*, Vol. 44, No. 4, 2018, pp. 1–30. <https://doi.org/10.1145/3210754>.
- [19] Frison, G., and Diehl, M., “HPIPM: A High-Performance Quadratic Programming Framework for Model Predictive Control,” , 2020. <https://doi.org/10.48550/ARXIV.2003.02547>.
- [20] Diehl, M., Findeisen, R., Allgöwer, F., Bock, H., and Schlöder, J., “Nominal Stability of Real-Time Iteration Scheme for Nonlinear Model Predictive Control,” *IEE Proceedings - Control Theory and Applications*, Vol. 152, No. 3, 2005, pp. 296–308. <https://doi.org/10.1049/ip-cta:20040008>.
- [21] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M., “Optuna: A Next-generation Hyperparameter Optimization Framework,” *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, Anchorage AK USA, 2019, pp. 2623–2631. <https://doi.org/10.1145/3292500.3330701>.
- [22] Hansen, N., “The CMA Evolution Strategy: A Tutorial,” , 2016. <https://doi.org/10.48550/ARXIV.1604.00772>.