Simulating Interaction Movements via Model Predictive Control

MARKUS KLAR, FLORIAN FISCHER, and ARTHUR FLEIG, University of Bayreuth
MIROSLAV BACHINSKI, University of Bergen
JÖRG MÜLLER, University of Bayreuth

We present a Model Predictive Control (MPC) framework to simulate movement in interaction with computers, focusing on mid-air pointing as an example. Starting from understanding interaction from an Optimal Feedback Control (OFC) perspective, we assume that users aim at minimizing an internalized cost function, subject to the constraints imposed by the human body and the interactive system. Unlike previous approaches used in HCI, MPC can compute optimal controls for nonlinear systems. This allows to use state-of-the-art biomechanical models and handle nonlinearities that occur in almost any interactive system. Instead of torque actuation, our model employs second-order muscles acting directly at the joints. We compare three different cost functions and evaluate the simulation against user movements in a pointing study. Our results show that the combination of distance, control, and joint acceleration cost matches individual users’ movements best, and predicts movements with an accuracy that is within the between-user variance. To aid HCI researchers and designers in applying our approach for different users, interaction techniques, or tasks, we make our SimMPC framework, including CFAT, a tool to identify maximum voluntary torques in joint-actuated models, publicly available, and give step-by-step instructions.

CCS Concepts: • Human-centered computing → HCI theory, concepts and models;

Additional Key Words and Phrases: Simulation, model predictive control, optimal feedback control, biomechanics, interaction techniques, mid-air pointing, AR/VR environments, maximum voluntary torques

ACM Reference format:
Markus Klar, Florian Fischer, Arthur Fleig, Miroslav Bachinski, and Jörg Müller. 2023. Simulating Interaction Movements via Model Predictive Control. ACM Trans. Comput.-Hum. Interact. 30, 3, Article 44 (June 2023), pages 50.
https://doi.org/10.1145/3577016

1 INTRODUCTION

Movement during interaction can be understood from an Optimal Feedback Control (OFC) perspective [18]: During interaction, users aim at computing muscle control signals, which control the dynamical systems of the users’ bodies, which in turn control interactive systems. OFC states that users aim at minimizing an internal cost function subject to the constraints imposed by the users’ bodies and the interactive systems. They do so by observing the state of the interactive system and continuously adjusting their controls to further their goals.

Authors’ addresses: M. Klar, F. Fischer, A. Fleig, and J. Müller, University of Bayreuth, Bayreuth, Universitätsstraße 30, 95440, Germany; emails: {markus.klar, florian.j.fischer, arthur.fleig, joerg.mueller}@uni-bayreuth.de; M. Bachinski, University of Bergen, Bergen, P.O. Box 7800, 5020, Norway; email: miroslav.bachinski@uib.no.

This work is licensed under a Creative Commons Attribution-ShareAlike International 4.0 License.
© 2023 Copyright held by the owner/author(s).
1073-0516/2023/06-ART44
https://doi.org/10.1145/3577016

ACM Transactions on Computer-Human Interaction, Vol. 30, No. 3, Article 44. Publication date: June 2023.
It is this observed continuity and the adjustment of controls that drives the desire to model interaction beyond summary statistics, in order to predict movement along the entire interaction loop between human and computer, including, e.g., joint postures or cursor trajectories, on a moment-by-moment basis. Taking the OFC perspective allows us to accomplish these things, by modeling interaction as an optimization problem. Here, the optimization variable is a continuous muscle signal, i.e., a function of time, that drives the user’s movement, which “controls” the system. Thus, this optimization problem is usually referred to as an Optimal Control Problem (OCP). The feedback part of OFC is due to solving the OCP in a feedback manner to model the user’s ability to adjust their control during interaction, e.g., to react to unforeseen circumstances such as perturbations in the cursor movement.

Previous approaches of modeling interaction from the OFC perspective have employed linear optimal control theory, particularly the Linear Quadratic Regulator (LQR) [17], its stochastic extension LQG [18], and intermittent control methods [44]. These approaches differ fundamentally from our approach in that they considerably simplify the problem of computing the optimal control signals by using linear approximations to the human–computer system (e.g., a second-order spring-damper model) and quadratic cost functions. However, these limitations lead to unrealistic simplifications of the human–computer system. Typically, human movements are simulated only with simple point-mass models, since modeling the kinematic chain already leads to nonlinear dynamics. Other important nonlinear features, such as those of interactive systems (e.g., transfer functions), similarly cannot be modeled by this linear approach. Furthermore, quadratic cost functions cannot accurately reflect many tasks in Human–Computer Interaction, such as accurately hitting a button with abrupt boundaries.

In this article, we extend the OFC approach to Human–Computer Interaction to nonlinear dynamics and non-quadratic cost functions by using Model Predictive Control (MPC) [23]. This allows us to investigate the simulation of human movement during interaction with computers using a state-of-the-art nonlinear biomechanical model of the human upper extremity in combination with nonlinear interaction dynamics such as pointer acceleration [46]. We evaluate our approach by simulating an ISO mid-air pointing task with two different interaction techniques, each in two different settings.

MPC as a method has various strengths, such as the easy inclusion of constraints and certain theoretical functionality guarantees to provide trust and reliability, but the main idea behind MPC is complexity reduction in time. It takes the above OCP, which can be computationally hard to solve for the whole interaction/movement duration, and breaks it down into iterative sub-problems of much smaller duration, which are thus considerably easier to solve. After solving a sub-problem, only the first part of the resulting optimal control sequence is applied to the system, resulting in a new system state. The horizon is then shifted by one step, i.e., the next sub-problem starts with this new state. This makes the MPC a closed-loop feedback controller, which is inherently robust against perturbations that may occur during the interaction.

In summary, the contribution of this work to the field of HCI is threefold:

1. We propose a framework that combines nonlinear biomechanical modeling and MPC to simulate human movement during interaction on a moment-by-moment basis (SimMPC), and a method to infer the maximum voluntary torques used in an interaction task (CFAT).
2. We evaluate our framework for the example use case of mid-air pointing, including comparisons between three different cost functions in their ability to generate biomechanically plausible movements, as observed in a new user study.
3. We make our approach accessible for HCI researchers by providing the sim-mpc and cfat python packages as well as a step-by-step guide on how to apply our framework to different tasks and interaction techniques.
The article is structured as follows. Related work is discussed in Section 2. The core of this article, our simulation approach using MPC (and practical advice on its use), is presented in Section 3. In Section 4, we introduce CFAT, a method to compute maximum voluntary torques for joint actuated models such as the one used in this article. Then follows an evaluation of our approach, applied to the use case of ISO pointing in VR, in Sections 5 and 6, where we show that our simulation is able to predict biomechanically plausible user movements. A discussion of the advantages and limitations of MPC ensues in Section 7. Section 8 explains step-by-step how to apply and extend our framework to different tasks and interaction techniques, using tracking a moving target via ray casting with a handheld VR controller as an example. Section 9 concludes the article.

The article is supplemented by our open-source project (https://github.com/mkl4r/sim-mpc), which consists of several components. The sim-mpc python package includes scripts to simulate mid-air interaction movements, compare simulation and user trajectories, generate plots, and optimize model parameters. The ISO-VR-Pointing Dataset contains marker, joint angle, and joint torque data for all mid-air pointing movements from our user study, and the SIM-MPC Dataset contains the corresponding simulation data produced for this work. The cfat python package provides an implementation of the CFAT method.

2 RELATED WORK

2.1 Forward Models of Interaction Movements

Forward models of movement during interaction with computers can predict variables such as movement duration, joint angles, or muscle activations. Depending on what they predict, they can be categorized as summary statistics (e.g., movement duration), end-effector models (e.g., end-effector position), or kinematic chain models (e.g., body joint trajectories).

The most widely used summary statistics model of the end-effector is Fitts’ Law [19]. It allows to predict the overall movement time $MT$ from the distance $D$ and width $W$ of the target as $MT = a + b \cdot \log_2(D/W + 1)$ (in the Shannon formulation [43]). It is important to note that Fitts’ Law has been developed to describe movement of the human hand. The fact that the same law can also be used to describe the movement of a virtual end-effector such as the mouse pointer, mediated by input devices and computer programs, is one of the great insights of HCI [8]. The parameters $a$ and $b$ must be identified for each user and type of movement (e.g., interaction technique) separately. Recently, more advanced models have been developed to predict, e.g., the failure rate and button press timing in moving-target acquisition tasks [37, 38].

End-effector models describe the entire trajectory of the end-effector during the movement. A classical end-effector model of hand movement is the minimum jerk model [20]. In HCI, only few works investigate the motion of the end-effector, although Bootsma et al. [6] demonstrate the importance of understanding movement in HCI beyond summary statistics. Müller et al. [47] give an introduction to end-effector models in HCI. They investigate the kinematics of mouse movements and compare four models from manual control theory. Quinn and Zhai [53] demonstrate how the minimum jerk model can be used to model finger movements during gesture typing. Jokinen et al. [33] frame touchscreen typing as a visuomotor coordination task and show that optimal supervisory control allows to generate human-like eye-hand movement patterns. Fischer et al. [17, 18] compared the applicability of different optimal control methods to simulate and predict mouse pointing trajectories, and introduced a general optimal control framework for Human–Computer Interaction. The focus there lies on controllers, such as the Linear-Quadratic Gaussian (LQG) Regulator, which are able to describe mouse pointing while also incorporating signal-dependent noise via a linear-quadratic OCP. The limitation to linear system dynamics rules out its application to more complex models of human biomechanics. Moreover, all of the above works have only analyzed motion in 1D or 2D, although they are in general not limited to 1D or...
2D motion. The only end-effector model that is evaluated with 3D mid-air movements in HCI that we are aware of is the recent work of Bachinski et al. [2], who investigate a 2nd and a 3rd order lag for modeling mid-air movements.

In this work, we are aiming to model not only end-effector movements in 3D but using a biomechanical model of the human upper body. This allows us to obtain joint angles, velocities, and even aggregated muscle activation, each observed during interaction. Therefore, we consider kinematic chain models that, in contrast to pure end-effector models, also make predictions about the underlying causes of the movement by modeling the entire kinematic chain. In particular, this allows to predict ergonomic variables such as joint angles and joint moments. Most of the previous work on biomechanical models of human movement outside of HCI has concentrated on the substantially simpler 2D case and simple linked-segment models [28, 41, 61, 67]. Linked-segment models use simplified bones as sticks and hinge joints, usually without movement constraints. In movement science, the minimum torque change model [67] has been proposed, transferring the idea of the minimum jerk model (i.e., maximization of “smoothness”) to a simple 2D linked-segment model. This model requires the exact movement time as well as all joint angles, velocities, and torques of the initial and final postures as input, and yields the kinematics and dynamics of the movement between initial and final state as output. Li and Todorov [41] present a control method for a 2D linked-segment model using the iterative Linear Quadratic Regulator (iLQR), which minimizes the difference between current and target posture plus quadratic control costs. However, this assumes that the final body posture is known in advance, which is not necessarily the case when the only goal is to move an end-effector (i.e., the fingertip or a virtual cursor) to a target. Moreover, the model has not yet been extended to the 3D case.

2.2 Inverse Biomechanical Simulation in HCI

In contrast to forward models, inverse biomechanical simulation takes as input human movement data and performs inverse estimations of how a specific movement was created. The method stems from the fields of biomechanics and rehabilitation and allows to compute accurate physiological indices of movements [10, 54]. Given motion capture data, it allows to estimate multiple internal variables such as joint angles, joint moments, muscle forces and activation, and neural excitation signals. At the core of the biomechanical simulation is a musculoskeletal model, which represents the kinematic, inertial, dynamic, force generation, and neural control properties of the human body [56]. Biomechanical simulation has been introduced and validated for HCI tasks as a method for ergonomic and fatigue evaluation of post-desktop user interfaces [3, 4]. It has also been used as a data generation method to develop summarization models of performance and ergonomics for arm movements [5]. Simplified biomechanical models were adapted as components of simulations for fatigue assessment tools [30, 31]. Although one current weakness of biomechanical simulations is its necessity for motion capture data collection in user experiments, these simulations have a large potential in the field of HCI, in particular for the analysis and development of AR, VR, and ubicomp user interfaces.

2.3 Deep Learning and Muscle Control

The above works from Sections 2.1 and 2.2 have investigated the control of the human body or a virtual object using either outcome-specific relationships such as Fitts’ Law, or optimization methods such as iLQR. Another approach incorporates recent tools and methods from the field of Deep Learning. Most notably, Cheema et al. [9] recently presented a method to estimate cumulative fatigue during mid-air interaction, in terms of Borg CR10 ratings. They use a 3D linked-segment arm model and reinforcement learning (RL) to learn a control policy for a Fitts’ law type task. They propose a novel reward function—the analogy of a cost function in OCPs, based on effort.
estimated through the *Three Compartment Controller*, and show that this generates faster and more “ergonomic” movements compared to a baseline reward of summed normalized instantaneous joint torques. Their model is shown to be able to predict the Borg CR10 ratings of the movements performed in [31] with good accuracy. However, Cheema et al. [9] did not analyze the realism of the movements generated by their approach in terms of end-effector trajectories or joint angles, but rather in terms of predicted cumulative fatigue, averaged over 12 models.

Following the work of Cheema et al., in [16] a state-of-the-art RL algorithm was used to learn to move the finger to arbitrary targets within reach. The resulting end-effector trajectories follow both Fitts’ Law [19] and the $\frac{1}{2}$ Power Law [36]. Lately, Hetzel et al. have extended the model from [16] to simulate mid-air keyboard typing, using the same RL method [29].

Beside these works, the objective of many research works making use of Deep Learning methods is not to model or understand human motion, but rather to create interesting and realistic animations for movies or computer games. We are not aware of any works from this research area that compare the synthesized movements to actual human movements on a biomechanical level. Similarly to the works from movement science, most works have controlled the torques at the joints (e.g., [26, 49]). Control on a muscle-level has traditionally been considered to be computationally infeasible. This is due to the fact that the computation time increases exponentially with the dimensionality of the control problem, called the *curse of dimensionality*.

Recently, however, two approaches to create movements of muscle-actuated characters have been presented. Lee et al. [39] propose a two-level *imitation learning* algorithm for musculoskeletal models. A high-level controller follows a reference motion and generates target joint angles. A low-level controller then controls the muscles to generate the appropriate forces. Imitation learning assumes that reference motions from humans are available. Whether and how imitation learning approaches can generate novel interaction movements that are not available as recordings will be an important question for future research.

In contrast to imitation learning, *reference-free approaches* can synthesize novel movements based only on the model description and reward function. Jiang et al. [32] present an approach to circumvent the muscle control problem by controlling the character in joint space, while determining maximum joint torques and energy costs from a neural network, learned from a realistic model in OpenSim [10]. Jiang et al. demonstrated their technique on a leg model. Whether and how this approach can work for a significantly more complex arm model, especially taking the shoulder into account, remains open.

Control of muscles is particularly necessary when the movements are big and cover very different joint angles, as moment arms change significantly during the movement in such cases. One example used by Jiang et al. [32] is a jump for maximum height, where the joint torque network prevents overbending of the knees as an optimal strategy. However, during most interactions, the movements are small and the moment arms change only minimally during the execution. In this case, actuating the joints either based on simplified muscle dynamics (as we do in this work) or direct torque control can be a good approximation and is substantially simpler to use than complex musculotendon models. For small movements, passive forces created by ligaments and musculotendon units also play a smaller role.

### 3 Modeling Interaction as Model Predictive Control

In this section, we describe our approach to model and simulate human movement during interaction with the computer using MPC. We lay the theoretical foundation and provide concrete but extensible models and practical advice. A concrete use case is described in detail in Section 5.

The defining aspect of movement in Human–Computer Interaction is that users move their bodies *in order to change the state of an interactive system* such that their virtual representation,
(e.g., an avatar or a cursor), reaches a desired state, e.g., selecting a button or dragging a virtual object.

When modeling this interaction, it is important to be able to deal with nonlinearities, as user movement and interaction techniques are in general nonlinear. On top of that, for most interaction techniques, there exists an infinite number of body movements that result in the same state-change in the interactive system. For example, using the mouse as input device, movements to the left result in the same cursor movement as movements to the top after a clockwise rotation of the mouse by 90 degrees. This complicates the simulation of movements, as it is unclear how the model should move in order to plausibly replicate human behavior during interaction. Therefore, in order to understand the entire interaction loop on a moment-by-moment basis, the actual state of the user’s body and the input device needs to be taken into account, which can be achieved by utilizing a unifying and mathematically rigorous optimal control framework of interaction [18].

Our framework is depicted in Figure 1. Our model takes into account the target user group, the interaction technique, and the interaction task. The first influence is the User Model, see Section 3.1, where we match the physical properties of the target users by using state-of-the-art biomechanical models. The interaction technique consists of two parts in our framework. First, modeling of the Input/Output device (e.g., motion capture tracking of the index finger, or the combined use of a HTC Vive controller and head-mounted display (HMD)) is described in Section 3.2. Second, we define Interface Dynamics that determine how the user input is transferred to the virtual system (e.g., to a virtual cursor) in Section 3.3.

Readers that are already familiar with the above concepts and are mostly interested in the core method used to simulate movements may directly skip to Section 3.4. There we introduce the notation of a nonlinear OCP, which augments the former discussed models of the human body
and the interaction technique with a \textit{Cost Function} that formalizes the user-specific objectives for a given interaction task.

In Section 3.5, we show how the OCP can be solved via MPC, resulting in a simulation of the complete biomechanical chain during interaction. We thus obtain trajectories of joint angles, angular velocities, and accelerations; trajectories of cursor positions, velocities, and accelerations; and biomechanical data such as aggregated muscle recruitment.

### 3.1 User Model

The biomechanical properties of the user model should match those of the considered users, and the range of possible movements must be sufficient to fulfill the task. To fit in our simulation pipeline (Figure 1) and act as a part of the system dynamics, the user model should be able to be forward-simulated, i.e., to map the current body state $\mathbf{x}_{\text{user}}$ (including, e.g., joint angles and angular velocities, and the internal state of the muscles), and the aggregated muscle control signals $\mathbf{u}$ to the next (i.e., updated) body state $\mathbf{x}_{\text{user}}^+$ in a realistic way.\footnote{Here and throughout this work, we denote states that contain a variety of different quantities in bold font, and the individual quantities in regular font.} Formally, this mapping can be defined via a function $f_{\text{user}}$:

$$\mathbf{x}_{\text{user}}^+ = f_{\text{user}}(\mathbf{x}_{\text{user}}, \mathbf{u}).$$

In this work, a biomechanical, joint-actuated model implemented in a physics engine, coupled with second-order muscle dynamics, will take the role of $f_{\text{user}}$. Of course, it is possible to exchange our user model with other models of human motion.

#### 3.1.1 Upper Extremity Model in MuJoCo

We make use of the fast physics simulation MuJoCo [64] to handle the complex biomechanics of human motion. In [16], a MuJoCo model from the state-of-the-art OpenSim [58] musculoskeletal model from Saul et al. [56] was derived. We use this MuJoCo model for two reasons: (i) limitations in OpenSim’s ability to simulate contacts—it is very difficult in OpenSim to allow a model to interact with input devices and environmental objects such as a chair or table, while preventing the model from reaching through its torso or legs--; and (ii) computation speed.

The biomechanical model has seven independent joints (i.e., seven DOFs)\footnote{Some human joints are reflected by multiple model joints. Therefore, throughout the article, we use the term \textit{joint} synonymously for a hinge joint in our model.} and 13 coupled joints, representing a shoulder, an elbow, and a wrist. The shoulder is the centerpiece of the model and is connected to a torso—which is made immovable during interaction for simplicity—through a set of three independent and eleven coupled joints. The three independent joints set up the angle and extent of the elevation, as well as the rotation of the upper arm. Ten of the eleven coupled joints are used to accurately describe the motion of clavicle and scapula with respect to the shoulder elevation. Since the joints build on each other, another coupled joint is used to revert the elevation angle before applying rotation. The elbow is composed of two independent joints allowing flexion-extension and pronation-supination movements. For the wrist, we use four joints, two independent and two coupled, which allow accurate flexion-extension and abduction-adduction movements of the hand. The finger joints are locked in a pointing posture, since they are less important for our example tasks and omitting them considerably simplifies the user model. The complete model is depicted in Figure 2; joint angle ranges can be found in Appendix B.1.

To avoid the \textit{curse of dimensionality}, i.e., the exponential growth of computation time with the number of variables to be optimized, we refrain from including muscles in our MuJoCo model. Instead, we implement simplified muscles that directly act on the joints as follows. We place a
torque actuator that can produce positive and negative torque around the axis of each of the seven independent joints. At any given time step $n$, for $i \in \{1, \ldots, 7\}$ the applied torque $\tau^i(n)$ of each actuator depends on its current activation $x^i_\sigma(n)$, scaled by the maximum voluntary torque $g^i$ for the respective joint:

$$\tau^i(n) = g^i x^i_\sigma(n).$$

The current activation $x^i_\sigma(n)$ of each torque actuator is obtained through a simplified second-order muscle model, which is explained in detail below.

For practical applications, one challenge with this simplified muscle model is to determine the maximum voluntary torques $g = (g_1, \ldots, g_7)^T \in \mathbb{R}^7$ for each independent joint, as to prevent unrealistic movements. To this end, we propose CFAT, a tool described in Section 4, to obtain better matching torques.

3.1.2 Second-Order Muscle Dynamics. Modeling and simulating human muscles has proven to be challenging. This is not only because of the sheer amount of muscles—in the original OpenSim model [56], the shoulder and arm alone are moved by a total of 31 muscles—but also because of the complex interaction of force generation, tendon lengths, tendon positioning, and so on. Optimizing for each muscle activation simultaneously is a challenging problem, which so far has only become feasible through techniques like hierarchical optimization [42] or through aggregation. We follow the approach by van der Helm et al. [69], who aggregate muscles for each DOF using second-order dynamics. We discretize these muscle dynamics using the forward Euler method [7]. The vector $x^i_\sigma = (x^i_\sigma[-1], x^i_\sigma[-1])^T \in [-1, 1]^2$ contains the activation for all seven DOFs. The vector of activation derivatives is denoted by $x^i_\sigma = (x^i_\sigma^1, \ldots, x^i_\sigma^7)^T \in \mathbb{R}^7$, and is affected by the vector of applied controls denoted by $u = (u^1, \ldots, u^7)^T$. In formulas, the discrete-time dynamics for each DOF $i \in \{1, \ldots, 7\}$ can be described as follows, where $n$ is the current time step and $n+1$ the next one:

$$\begin{bmatrix} x^i_\sigma(n+1) \\ x^i_\sigma(n+1) \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ \frac{-\Delta t}{t_e t_a} & 1 - \Delta t \frac{t_e + t_a}{t_e t_a} \end{bmatrix} \begin{bmatrix} x^i_\sigma(n) \\ x^i_\sigma(n) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\Delta t}{t_e t_a} \end{bmatrix} u^i(n),$$

with initial constraints

$$x^i_\sigma(0) = \sigma_0, \quad x^i_\sigma(0) = \dot{\sigma}_0.$$ 

Here, $\Delta t = 2 \text{ ms}$ is the update interval, $t_e = 30 \text{ ms}$ and $t_a = 40 \text{ ms}$ are the fixed excitation and activation time constants, respectively, which are taken from van der Helm et al. [69], and $\sigma_0$ and $\dot{\sigma}_0$ are initial values for the activation and its derivative.\footnote{An idea about the magnitudes of these initial values is obtained through practical experiments, where these initial values are obtained from data for each trial, see Section 5.4.}
3.1.3 Necessary Adjustments for Other Use Cases. Since we focus on the simulation of (right-handed) mid-air pointing movements of adults, we only model the upper extremity of an adult, i.e., the right arm and shoulder. However, by adjusting the MuJoCo model (which comes down to editing an XML file), e.g., to size or physique, different target user groups can be considered.

We note that, although we use a state-of-the-art biomechanical model, we do not model the complete human body. If more complex movements involving additional extremities are needed, the model needs to be augmented, respectively. For example, another arm can be added by mirroring, as has been done in [29].

Note that, in general, the interaction technique and interaction task have an influence on the choice of the user model. For example, if the input device is a handheld controller, the hand must be able to hold the controller. In particular, the (rigid) hand must be re-arranged to match the position of holding the considered device. More extensive models may contain palm and finger joints to enable fine movements. Similarly, changing the interaction task may require adjustments in the user model. If, for example, the task is to grasp and move some virtual object, the MuJoCo model would require a biomechanically more accurate model of the hand. For further details on how to adjust the MuJoCo model, we refer to the documentation provided by MuJoCo.

Major changes to the user model may also affect the maximum voluntary torques, which is why, in this case, we recommend reapplying the CFAT tool described in Section 4.

3.2 Input/Output Device

In addition to biomechanics, we implement several mid-air interaction techniques. Following the scheme from Figure 1, we divide interaction techniques into their (physical) input devices (e.g., a joystick, touch screen, or motion capture system) and output devices (e.g., a monitor or HMD), and the mapping from the information that the computer receives to the virtual state that it displays.

The model of the input device should be able to realistically capture the same data from the user model as the input device captures from the real user. If, for example, a joystick senses angular movement in two axes, the model of the joystick should be able to obtain the same information. Formally, we understand an input device as a function $f_{\text{dev}}$ that maps the user’s current state $x_{\text{user}}$ (e.g., body posture) and the current device state $x_{\text{dev}}$ (e.g., joystick angle and/or motion capture marker position) to the updated device state $x_{\text{dev}}^+$, i.e.,

$$x_{\text{dev}}^+ = f_{\text{dev}}(x_{\text{dev}}, x_{\text{user}}). \quad (5)$$

In the considered use case of mid-air pointing without any handheld device, the input device corresponds to the motion capture system PhaseSpace,\(^6\) which allows to continuously track the movement of the user. An LED marker is placed at the tip of the right index finger, whose position is used to determine the motion of a virtual cursor. To model this input device, we use a virtual marker on our MuJoCo user model’s index finger to track its position, which we denote as $x_{\text{ee}}$. In this particular use case, we thus have

$$x_{\text{dev}} = x_{\text{ee}}.$$

Since we can obtain this data directly from MuJoCo, we do not need to implement any additional dynamics here, which would be necessary when modeling, e.g., a joystick. Therefore, the device dynamics in our case are given by the very simple mapping

$$f_{\text{dev}}(x_{\text{dev}}, x_{\text{user}}) = x_{\text{ee}}. \quad (6)$$

\(^5\)https://mujoco.readthedocs.io.
\(^6\)https://www.phasespace.com/x2e-motion-capture/.
Output modalities may also differ between interaction techniques. Most commonly, users get a visual feedback via a screen that shows how the virtual environment reacts to their input. With this information, users can evaluate their actions, e.g., through the position of a virtual cursor, and possibly change their strategy to fulfill the given task, e.g., pointing towards a virtual target. In this article, we demonstrate the simulation of mid-air pointing in VR by assuming perfect observation. This particularly implies that users always see the exact cursor position.

3.2.1 Necessary Adjustments for Other Use Cases. The MuJoCo model can easily be extended to other input devices, since many different sensors like gyroscopes as well as force, torque, or touch sensors are directly available in MuJoCo. Visual input (to the computer) can be implemented with cameras that can sense RGB pictures or just depth information. If the input device ought to be directly manipulated by the user, one needs to adjust the user model such that it can actually use the device as intended. For example, to grab and use a handheld controller, the posture of the hand would have to be adjusted to fit a controller that needs to be implemented in the same physics engine. Furthermore, for some input devices (e.g., a joystick or gamepad), fine motor finger movements are necessary, which are currently not possible with our used MuJoCo model.

To implement output devices and perception, one would need to add another layer after the System Dynamics in Figure 1, which maps the “real” interface state to the one perceived by the user. For example, if the output device was a 2D screen that does not allow to directly infer depth information of the regarded scene, the output device model would need to take into account the underlying projection.

3.3 Interface Dynamics

Once the human input is received, the user interface needs to be updated. For example, a change in position of the input device should entail a movement of the controlled virtual object, e.g., the virtual cursor. Additionally, the virtual world itself may have virtual dynamics. For example, throwing a virtual ball at a virtual pin may lead to that pin being knocked over.

To formalize the whole process, we use three functions. First, a transfer function \( f_{\text{if}} \) transfers physical movement to virtual. As such, it depends on the current state of the input device, \( \mathbf{x}_{\text{dev}} \).

Next, the virtual dynamics come into play via the function \( f_{\text{vd}} \), which takes as arguments the current state of the interface \( \mathbf{x}_{\text{if}} \) (e.g., cursor or button position) and the output of the transfer function. Finally, these two components are wrapped by the wrapper function \( f_{\text{if}} \) into the Interface Dynamics of the considered interaction technique, which yields the updated virtual state \( \mathbf{x}_{\text{if}}^+ \), i.e.,

\[
\mathbf{x}_{\text{if}}^+ = f_{\text{if}}(\mathbf{x}_{\text{if}}, \mathbf{x}_{\text{dev}}) = f_{\text{vd}}(\mathbf{x}_{\text{if}}, f_{\text{if}}(\mathbf{x}_{\text{dev}})).
\]  

Since the Interaction Dynamics is part of a nonlinear OCP, it is possible to include arbitrary complex virtual dynamics here (although continuity and smoothness of the functions are desirable).

The flip-side, i.e., if no explicit virtual dynamics are required, still fits in this framework. In this case, \( f_{\text{if}} \) simply is the transfer function:

\[
f_{\text{if}}(\mathbf{x}_{\text{if}}, \mathbf{x}_{\text{dev}}) = f_{\text{if}}(\mathbf{x}_{\text{dev}}).
\]

In our case of mid-air pointing, we do not need explicit virtual dynamics and as such use (8). This is due to the fact that we only simulate single aimed movements to a static target, i.e., the only change in the interface state concerns the position of the virtual cursor, which we denote by \( x_p \). This position is updated based on the transfer function that maps the (physical) end-effector position \( x_{ee} \), which is perceived by the computer through the input device and thus is part of the input device state \( x_{\text{dev}} \), to the position of the virtual cursor \( x_p \), which is part of \( x_{\text{if}} \). This leads to
simple transfer functions of the form:

\[ f_{\text{If}}(x_{ee}) = x_p. \]  

\[(9)\]

But even without virtual dynamics, solely using \( f_{\text{If}} \), we can encompass a variety of interaction techniques.

First, we consider the class of virtual cursors in VR [50]. These simple interaction techniques introduce a displacement between the physical and the virtual hand of the user. The transfer functions for these techniques can be given in an Input-Output-Space formulation. In our case, the virtual cursor is uniquely given by an input space origin \( \omega_I \in \mathbb{R}^3 \) and an output space origin \( \omega_O \in \mathbb{R}^3 \). The cursor position is obtained by transferring the fingertip position in input space coordinates to the output space. The complete transfer function for Virtual Cursor is thus given by

\[ f_{\text{If}}(x_{ee}) = x_{ee} - \omega_I + \omega_O. \]  

\[(10)\]

In particular, placing the end-effector at the input origin, i.e., \( x_{ee} = \omega_I \), results in the cursor being at the output origin. The choice of the input origin \( \omega_I \) influences task performance. For example, a lower input origin allows to achieve the same cursor position with a lower end-effector position, i.e., with a lowered arm, eventually resulting in more comfortable movements. To match horizontal alignment, we define the output space such that it represents a virtual 3D space in front of the user by setting \( \omega_O = (-0.1 \text{ m}, 0.0 \text{ m}, 0.55 \text{ m}) \), i.e., 10 cm right and 55 cm in front of the user.

As a slightly more complex interaction technique, we select the group of Virtual Pad techniques [1], which project the 3D fingertip position to a 2D cursor position. The technique can be described as using a tablet placed on a table to move a cursor on a screen in front, with the differences that the tablet is indefinitely large and that there is no need to touch it. The virtual display, i.e., the output plane on which the cursor moves, is characterized by its origin \( \omega_O \in \mathbb{R}^3 \) and normal vector \( n_O \in \mathbb{R}^3 \). We set this output plane to be in front of and facing the user, i.e., \( \omega_O = (-0.1 \text{ m}, 0.0 \text{ m}, 0.55 \text{ m}) \) and \( n_O = (0, 0, -1) \). An input plane is analogously defined by its origin \( \omega_I \in \mathbb{R}^3 \) and normal vector \( n_I \in \mathbb{R}^3 \). The cursor position is obtained in two steps: First, the fingertip is projected onto the input plane by a function \( \text{Proj}_I \). Then, this point is rotated from input to output plane orientation by a function \( \text{Rot}_{IO} \). Finally, the cursor position is translated such that it lies on the output plane. In total, the Virtual Pad transfer function is therefore given by

\[ f_{\text{If}}(x_{ee}) = \text{Rot}_{IO}(\text{Proj}_I(x_{ee})) + \omega_O. \]  

\[(11)\]

Exact formulas for \( \text{Proj}_I \) and \( \text{Rot}_{IO} \) are given in Appendix A.

3.3.1 Necessary Adjustments for Other Use Cases. In the case of pointing, the presented transfer functions can easily be modified to match different interaction techniques. Modeling different pointing techniques such as ray casting [40] is also possible by adjusting the transfer function accordingly. In this case, one must add additional information of the virtual environment to the transfer function, e.g., the position and size of selectable objects. When creating new transfer functions, it is a good practice to implement them based on parameters, such as the input origin for the transfer functions used in this work. A parameter-based implementation allows for a quick adaptation and optimization of the interaction technique. Since the interface dynamics can incorporate both transfer functions and virtual dynamics, one could also model techniques where the interface has its own internal state, such as driving a virtual vehicle. In general, the virtual dynamics need to formalize the internal and mutual state dependencies of all interface objects that are relevant for the interaction, e.g., moving targets that need to be tracked by the virtual cursor, or interactable objects that may change their color or size depending on the context.
3.4 Modeling Interaction as Nonlinear Optimal Control Problem

Modeling human–computer interaction requires the use of dynamics that do not only change the state of the interface based on human input, but also capture biomechanics. Due to the redundancy of the human biomechanical system, there are infinitely many body movements that can be used to execute a given interaction task.

Building on the idea of optimal human movement control [65, 66], we assume that humans aim at behaving optimally with respect to an internalized cost function, subject to the dynamics of the human–computer-interaction system. This allows us to make use of the optimal control framework and rephrase the considered human–computer-interaction as nonlinear OCP, using an appropriate cost function as well as (discrete-time) system dynamics that describe the complete interaction loop. Formally, this can be written as

\[
\min_{u(\cdot)} J_{\infty}(x_0, u(\cdot)) = \min_{u(\cdot)} \sum_{k=0}^{\infty} \ell(x(k), u(k))
\]

such that \( x(k + 1) = f(x(k), u(k)), \ x(0) = x_0, \)

\[ x(k) \in \mathcal{X}, u(k) \in \mathcal{U}, \text{ for all } k \in \mathbb{N}. \]

Here, \( J_{\infty} \) is the cost to minimize, which is defined by the stage cost or running cost \( \ell : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R} \) that we need to design, \( f : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{X} \) is the nonlinear, continuous state transition map that takes the current state and control and yields the subsequent state according to the system dynamics, and \( x(\cdot) \) denotes the overall state trajectory that results from the forward simulation of the system with initial state \( x_0 \) and control sequence \( u(\cdot) \). State and control constraints are incorporated in the spaces \( \mathcal{X} \) (e.g., biomechanically feasible joint angles) and \( \mathcal{U} \) (e.g., maximum permissible aggregated control signal strength for each joint), respectively.

The equation \( x(k + 1) = f(x(k), u(k)) \) can be written in a shorter form, analogous to the previous sections, as \( x^+ = f(x, u) \), but we kept the current time \( k \) explicitly because it occurs in \( \ell \). Subsequently, we show how to apply the abstract formulation of the OCP Equation (12) to our case.

3.4.1 System Dynamics. The complete discrete-time system dynamics are obtained by combining the user model, input device, and interface dynamics that we have described in Sections 3.1, 3.2, and 3.3, respectively. These dynamics map the control signal \( u \) and the current state of the overall system \( x = (x_{\text{user}}, x_{\text{dev}}, x_{\text{if}}) \), consisting of user, device and interface states, to the next system state \( x^+ \). Therefore, we can formalize the system dynamics as

\[
x^+ = f(x, u) = (x_{\text{user}}^+, x_{\text{dev}}^+, x_{\text{if}}^+) \tag{13}
\]

where the formulas for \( x_{\text{user}}^+, x_{\text{dev}}^+, \) and \( x_{\text{if}}^+ \) are given by Equations (1), (5), and (7), respectively.

In particular, in our mid-air pointing use case, the state of the complete system consists of

\[
x_{\text{user}} = \begin{cases} x_{\text{qpos}}, x_{\text{qvel}}, x_{\text{qacc}} : \text{joint angles, angular velocities, and accelerations} \\ x_{\sigma}, x_{\bar{\sigma}} : \text{aggregated muscle activation and their derivatives} \end{cases}
\]

\[
x_{\text{dev}} = \begin{cases} x_{\text{ce}} : \text{physical end-effector position, and} \end{cases}
\]

\[
x_{\text{if}} = \begin{cases} x_{\text{p}} : \text{cursor position}. \end{cases}
\]

3.4.2 Cost Function. The cost function that is assumed to be minimized by a user during interaction needs to reflect the task requirements, goals, and intrinsically motivated objectives that can represent specific user strategies. Using the notation of the OCP Equation (12), the cost function is given by a stage cost function \( \ell(x, u) \), which maps the state of the system \( x \) and the control signal \( u \) to the respective cost.
Simulating Interaction Movements via Model Predictive Control

Considering mid-air pointing as an example, the task is to reach a given target with a virtual cursor. Reaching the target is often modeled as a terminal constraint. For this, however, the duration of the movement must be fixed beforehand. Since movement times vary from trial to trial and are not known in advance, they must either be estimated or calculated from experiments. However, we want to enable the simulation of human-like movements based on a model of the interaction dynamics and the user only, without relying on experimentally observed or estimated movement duration. Instead of using a terminal constraint, we thus follow a well-known approach and mitigate this problem by penalizing the distance between the cursor position \( x_p \in \mathbb{R}^3 \) and the target position \( p^* \in \mathbb{R}^3 \) at each time step \([11, 18, 51, 63]\). This does not only incentivize moving the cursor towards the target, but implicitly penalizes the movement duration as well, since slow movements result in higher accumulated distance costs (note the sum in Equation (12)).

Furthermore, humans are known to prefer moving with low effort \([25, 41, 65]\). We implement this concept by penalizing the aggregated muscle control signal, i.e., the control vector \( u \). As it is usually done in numerical optimization to improve the performance, we take the squared Euclidean norm, denoted by \( \| \cdot \| \), in the following. If required, each element of the control vector can be scaled individually before taking the norm to replicate different effort at different joints. Since this approach would introduce additional parameters, we assume that effort is solely dependent on the normalized muscle activations. In this way, we also reward the use of stronger muscles, since, for example, the same activation in the shoulder instead of the wrist produces a higher torque with the same cost.

In addition, we introduce two different cost terms that have previously been used to model optimal human behavior. The first one corresponds to the well-established commanded torque change \([35, 48, 71, 73]\), which penalizes the derivative of the commanded torques, that is, the torques that directly result from the applied motor commands. In our case, this corresponds to the derivative\(^7\) of the applied torque \( \tau \), which we denote by \( \dot{\tau} \) in the following.\(^8\) The second, less frequently used cost term corresponds to the joint acceleration, which leads to smooth movements towards the target \([71]\). We denote the vector of (angular) joint accelerations by \( x_{qacc} \), which is part of \( x_{user} \), see Equation (14).

With these components, we propose three different stage costs:

- **DC**: Distance and Control Costs.
  The distance between cursor and target as well as the aggregated muscle control are penalized at each time step:
  \[
  \ell(x(k), u(k)) = \| x_p(k) - p^* \| + r_1 \| u(k) \|^2 \tag{15}
  \]

- **CTC**: Commanded Torque Change Cost.
  This cost function adds to Equation (15) a third cost term, penalizing the commanded torque change:
  \[
  \ell(x(k), u(k)) = \| x_p(k) - p^* \| + r_1 \| u(k) \|^2 + r_2 \| \dot{\tau}(k) \|^2 \tag{16}
  \]

- **JAC**: Joint Acceleration Costs.
  This cost function adds to Equation (15) a third cost term, penalizing the squared joint accelerations:
  \[
  \ell(x(k), u(k)) = \| x_p(k) - p^* \| + r_1 \| u(k) \|^2 + r_2 \| x_{qacc}(k) \|^2 \tag{17}
  \]

The cost weights \( r_1, r_2 > 0 \) define the tradeoff between the different cost terms.

---

\(^7\)Due to the discrete-time setting, we take central/one-sided differences using \texttt{numpy.gradient}.

\(^8\)We obtain \( \tau \) and \( \dot{\tau} \) from the activations via Equation (2).
3.4.3 Fitting Cost Weights. The choice of those cost weights has a significant impact on the resulting simulation trajectories (an evaluation of the effect of cost weights can be found in Section 6.3). To find the most appropriate weights for our cost functions, we need to evaluate different weight pairs \((r_1, r_2)\). Since we aim at generating joint movements that are as close to human movements as possible, we evaluate a cost weight pair by comparing the resulting simulation sequence of joint angles \(x_{\text{qpos}}\) to that of a sequence of joint angles \(\hat{x}_{\text{qpos}}\) obtained through a user study, considering independent joints only. More precisely, we compute the root mean squared error (RMSE) between simulation and experimental data, i.e.,

\[
\text{RMSE} \left( x_{\text{qpos}}, \hat{x}_{\text{qpos}} \right) = \sqrt{\frac{1}{M} \sum_{k=0}^{M-1} \left\| x_{\text{qpos}}(k) - \hat{x}_{\text{qpos}}(k) \right\|^2},
\]

(18)

where \(M\) is the number of steps of the experimental data trajectory. To “rate” a weight pair, we sum the RMSE values for a given number \(S\) of simulated trajectories, resulting in the following loss function used for parameter optimization:

\[
\mathcal{L}_{\text{param}}(r_1, r_2) = \sum_{s=0}^{S-1} \text{RMSE} \left( x_{\text{qpos}}^{(s)}, \hat{x}_{\text{qpos}}^{(s)} \right),
\]

(19)

where \(x_{\text{qpos}}^{(s)}\) denotes the simulation trajectory obtained from the cost weights \(r_1\) and \(r_2\), and \(\hat{x}_{\text{qpos}}^{(s)}\) denotes the corresponding experimental trajectory, given a trial \(s \in \{0, \ldots, S - 1\}\). Since each evaluation of the RMSE Equation (18) requires solving a single OCP Equation (12) and thus results in large computation times, we decided to use a state-of-the-art derivative-free optimization algorithm that works with a low number of function evaluations and non-convex problems, and is easy to parallelize: the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [27].

For evaluation purposes, we additionally compute the RMSE on state components other than joint angles, e.g., joint velocities or accelerations, as well as cursor positions, velocities, or accelerations. The respective metric is defined analogously to Equation (18), with \(x_{\text{qpos}}\) and \(\hat{x}_{\text{qpos}}\) being replaced by the respective quantity.

3.4.4 Necessary Adjustments for Other Use Cases. The generalized formulation as an OCP is valid for a wide range of interactions between humans and virtual objects. If the target user group or the interaction technique is varied, one has to modify the relevant parts of the system dynamics as described in Sections 3.1, 3.2, and 3.3. If an interaction task different from pointing is considered, the cost function needs to be adjusted. For example, in the case of throwing in VR, a cost penalizing the distance of a virtual ball to a target area could replace the distance cost term described above. The presented method to obtain user specific cost weights can be used for a variety of cost functions, but joint trajectories from user trials are necessary. If such data is not available, one can instead use, for example, cursor trajectories instead of joint trajectories in the loss function Equation (19).

3.5 Simulating Movements with Model Predictive Control

Since the biomechanical simulation alone has highly nonlinear dynamics, we need to solve a nonlinear OCP. This renders it impossible to use solvers for linear OCPs recently introduced to the HCI audience such as LQR [18, Ch. 7] or LQG [18, Ch. 8]. Solving nonlinear OCPs is generally quite challenging, and on longer time horizons, they are often computationally intractable [24]. This problem can be tackled with a receding horizon approach, also known as MPC. Due to its notable properties—easy to implement, handles nonlinear constraints in contrast to the LQR [12], theorems guaranteeing that MPC produces sensible results [14, 23, 55]—MPC has matured into a
standard control method for linear and nonlinear dynamical systems, both from the academic and application [52, 70] point of view.

The main idea of MPC is complexity reduction in time. The solution of the OCP Equation(12) is approximated by iteratively solving sub-problems of Equation (12) on a much shorter time horizon. The first control of the resulting optimal control sequence is then applied to the system. Iterating this process results in a closed-loop system, which is able to react to perturbations that may occur during execution (e.g., due to signal-dependent noise in the motor system [13]), without the need to handle them explicitly within each optimization step [22]. The resulting closed feedback loop in our framework is depicted in Figure 3.

More formally, MPC computes a feedback law \( \mu : X \to U \), which maps arbitrary states \( x \in X \) to optimal controls \( u \in U \), via the following MPC algorithm:

1. Given the initial state \( x(0) \in X \), choose the horizon length parameter \( N \geq 2 \) and set \( n := 0 \).

2. Initialize the state \( x_0 = x(n) \) and solve the following open-loop\(^9\) OCP (we use the scipy.optimize.minimize from the Python scipy module,\(^{10} \) which implements the Broyden-Fletcher-Goldfarb-Shanno (L-BFGS-B) algorithm [34, 75]):

\[
\min_{u(\cdot) \in U^N} J_N(x_0, u(\cdot)) = \min_{u(\cdot) \in U^N} \sum_{k=0}^{N-1} \ell(x(k), u(k))
\]

such that \( x(k+1) = f(x(k), u(k)) \) for all \( k \in \{0, \ldots, N-1\} \),
\[
x(0) = x_0,
\]
\[
x(k) \in X \text{ for all } k \in \{0, \ldots, N\}.
\]

Use the first value of the resulting optimal control sequence denoted by \( u^*(\cdot) \in U^N \) for the feedback law, i.e., set \( \mu(x(n)) := u^*(0) \).

3. Update the state via
\[
x(n + 1) = f(x(n), \mu(x(n))),
\]
set \( n := n + 1 \) and go to step 1.

We specifically differentiate between \( k \) and \( n \) to distinguish open-loop dynamics (\( k \)) from closed-loop ones (\( n \)). Design parameters include, among others, the sampling times of the state and of

\(^9\)We call the OCP Equation (20) open-loop to emphasize that the solution of Equation (20) is not of feedback nature, i.e., cannot react to disturbances. This ability comes from the full MPC algorithm.

\(^{10}\)https://docs.scipy.org/doc/
the control. These parameters are hidden in the definition of \( f \), which corresponds to the system dynamics (including the MuJoCo simulation in our case), and determine the resolution with which the physics are simulated, and how frequently users are assumed to be able to change their control, respectively. In order to achieve high physical accuracy, we set the sampling time of the state to 2 ms. To reflect the fact that humans are not able to adjust their behavior continuously, but only intermittently [21], we set the sampling time of the control at 40 ms (i.e., the piecewise constant control signal can be adjusted every 40 ms).

An additional design parameter introduced by the MPC algorithm is the horizon length \( N \). Deciding on the horizon means facing a tradeoff. A longer horizon increases computation time, whereas a shorter horizon may lead to poor results. For example, if \( N \) is chosen too small, the cursor cannot be moved towards the target far enough to effectively reduce the total costs in the truncated horizon, i.e., the optimal control sequence \( u^* \) that minimizes the finite-horizon cost functional \( J_N \) does not result in the expected behavior. A more detailed analysis of the effect of \( N \) on the resulting closed-loop trajectories is presented in Section 6.4. Unless stated otherwise, we set \( N = 8 \) (i.e., 320 ms), as this value showed a good balance between performance and quality of simulation.

The L-BFGS-B algorithm was chosen as a solver for Equation (20) due to its computation and memory efficiency and ability to include control constraints easily. The parameters of the L-BFGS-B algorithm, which is used to solve the finite-horizon OCPs at each MPC step, are chosen as follows: objective function tolerance \( ftol = 10^{-6} \), gradient tolerance \( gtol = 10^{-5} \), step size for the numerical approximation of the Jacobian \( eps = 10^{-8} \), the maximum number of objective function evaluations \( \text{maxfun} = 10,000 \), and maximum number of iterations \( \text{maxiter} = 1,000 \).

Previous findings suggest that human motor control signals are affected by different noise sources, e.g., sensory and motor noise [13, 28, 57, 60, 63, 68]. In order to create realistic human movements that also exhibit intraindividual variance similar to real users, perturbations can be included in the state-transition-map \( f \). Note that, as the MPC is a closed-loop controller, we do not necessarily need to include the noise during optimization, i.e., the optimizer assumes that the system is deterministic. Instead, we include noise to the applied control \( \mu(x(n)) \) in step 1 of the MPC algorithm, i.e., before applying the second-order muscle model and proceeding with the next step. Applying the noise in the closed loop only considerably simplifies the OCPs and allows them to be solved efficiently. As suggested by van Beers et al. [68], we add signal-dependent and constant motor noise, i.e., two Gaussians with zero mean and a standard deviation of 0.103 \( \cdot \mu(x(n)) \) and 0.185, respectively, to the control \( \mu(x(n)) \).

4 CCFAT: A METHOD TO COMPUTE MAXIMUM VOLUNTARY TORQUES FOR JOINT-ACTUATED MODELS

Omitting real muscles in biomechanical models and replacing them with simplified muscles acting directly at the joints greatly simplifies computations, but it also creates another challenge. It is unclear how strong these simplified muscles need to be. Since the relative strength of each actuator has a large impact on how it needs to be actuated [32, 74], an appropriate choice of the maximum voluntary torques is crucial to generate biomechanically plausible movements. We, therefore, need to define the torque ranges of all actuators, i.e., the maximal positive and negative torques that can be applied at each DOF.

The natural approach to identify the torques humans apply during interaction would be to use existing Inverse Dynamics tools, as implemented in OpenSim. However, such tools obtain the complete inter-segmental torques acting on both independent and dependent joints, including passive forces, e.g., due to spring-dampers. In addition, the dependent joints cannot be actively actuated, but their torques emerge implicitly from the torques applied to the independent joints, i.e., the results from Inverse Dynamics cannot be used to determine the maximum voluntary
torques at the independent joints.\textsuperscript{11} Instead of relying on Inverse Dynamics, we thus apply a method similar to \textbf{Computed Muscle Control (CMC)} \cite{62}, which yields the sequence of muscle excitations that accounts for experimentally observed movements, given a fully muscle-actuated biomechanical model.

Starting with an initial posture from experimental data, the goal of our method to \textbf{compute feasible applied torques (CFAT)} is to find the sequence of applied torques that best explains the sequence of joint postures observed during an experiment. Due to the curse of dimensionality, we solve a sequence of optimization problems, one for each time step, as opposed to an optimization problem covering the entire motion, minimizing the following loss function:

\begin{equation}
L_{\text{CFAT}}(\tau) = \alpha e_{\text{qpos}}(\tau) + \beta e_{\text{qvel}}(\tau) + \gamma e_{\text{qacc}}(\tau).
\end{equation}

Here, the error terms $e_{\text{qpos}}(\tau)$, $e_{\text{qvel}}(\tau)$, and $e_{\text{qacc}}(\tau)$ denote the Euclidean distance between the one-step MuJoCo forward simulation with applied torques $\tau$ and the corresponding user data at this time step, in terms of joint angles, velocities, and accelerations, respectively (only incorporating the independent joints). According to our experience, penalizing an appropriate combination of joint angles, velocities, and accelerations turned out to be necessary to guarantee stability—choosing the weights $\alpha = 1,000$, $\beta = 50$, and $\gamma = 0.01$ showed good results in our case. After each optimization, one forward step is taken in the MuJoCo environment using the computed optimal torque. The resulting joint angles and velocities in the next time step are then used as initial values for the subsequent optimization, which returns the next optimal torques, and so on. Using this CFAT tool, we thus obtain a sequence of \textit{applied torques} that result in the original user trajectory when sequentially applied at the DOFs of the biomechanical model. Additionally, for each trial, CFAT yields the initial activations $\sigma_0$ and their derivatives $\dot{\sigma}_0$ used in our muscle model described in Section 3.1.2.

We clean the obtained torques from outliers by removing those that deviate more than three standard deviations from the respective mean. The vectors of maximum positive and negative torques, $\tau^+$ and $\tau^-$, are then determined as the component-wise maximum and minimum of the computed torques $\tau$ of all considered movements.

For technical reasons, the maximum and minimum torques are normalized such that the larger of both equals one for each DOF, and the resulting values are used as boundaries for the control $u$.\textsuperscript{12} The positive \textit{scaling ratio} vector $g = \max(|\tau^+|,|\tau^-|)$, with maximum taken component-wise, is then used as a gain vector, mapping the normalized activations $x_\sigma$ to the applied torques $\tau$ as in Equation (2).

It should be noted that the CFAT tool requires reference user data to measure how “human-like” a simulated joint trajectory is. In this work, we used the data from our user study, which was explicitly recorded for the considered interaction task. However, the obtained torque ranges should be appropriate for related interaction techniques and tasks as well, as was recently shown for the case of mid-air keyboard typing \cite{29}. If major changes to the user model are made, such as modifying the physiology, running CFAT on new reference data is recommended.

\section{USE CASE: ISO POINTING IN VR}

As a use case, we demonstrate the applicability of our simulation framework to mid-air pointing in VR. In Section 5.1, we describe the considered task and techniques. In Sections 5.2 and 5.3, \textsuperscript{11}A comparison of applied torques obtained from Inverse Dynamics and CFAT can be found in the Appendix B.2. \textsuperscript{12}Note that, because we use the muscle model described in Section 3.1.2, the applied controls $u$ might differ from the activation $x_\sigma$. However, given that $\Delta t < \sqrt{\frac{I_m}{\tau_{\text{max}}}}$ holds for the second-order muscle dynamics Equation (3), the activation $x_\sigma$ cannot exceed the applied controls $u$ in absolute terms. It is thus reasonable to impose the normalized torque boundaries on $u$ instead of $x_\sigma$.\textsuperscript{12}
we proceed with a description of the user study that we conducted to collect data for the user model generation and the evaluation of our simulation. Finally, in Section 5.4, we explain how our approach can be used to replicate individual trials of the user study.

5.1 Target User Group, Interaction Techniques, and Interaction Task

Our target user group includes healthy adults of average size and body shape. Therefore, we do not need to make special adjustments to the biomechanical user model. Nonetheless, since we did not have user models beforehand and aim at comparing our simulation trajectories to those obtained from our user study, we derived user models that matched the biomechanical properties of the participants in the user study, as described in Section 5.3. Technically, our target user group thus corresponds to those six participants (see Section 5.2.1).

We are interested in how well our model can synthesize human movement given different interaction techniques. As input device, we use a motion capturing system, which tracks the position of an LED marker that is placed on the tip of the right index finger, modeled in MuJoCo as described in Section 3.2. Using the notation introduced in Section 3.3, we investigate transfer functions without any additional virtual dynamics.

For each of the two interaction technique classes Virtual Cursor (10) and Virtual Pad (11), we define a basic variant in which the input space is at the same position as the output space, i.e., \( \omega_I = \omega_O \). For the virtual cursor, this means that the cursor always matches the position of the fingertip (i.e., the transfer function is the identity function), and for the virtual pad, the cursor is the orthogonal projection of the fingertip onto the input/output plane. Therefore, we refer to these techniques as Virtual Cursor Identity/ID and Virtual Pad Identity/ID, respectively. For both classes, we also consider an “ergonomic” condition, where the input space is at a lower, more comfortable height,\(^{13}\) denoted as Virtual Cursor Ergonomic and Virtual Pad Ergonomic in the following. The input and output normal vectors \( n_I \) and \( n_O \), respectively, are selected in such a way that the planes face the user and coincide for both interaction techniques. Hence, for the Virtual Pad ID technique, the fingertip is orthogonally projected onto a “virtual display” that is 55 cm in front of the user and on which the targets are also displayed. The Virtual Pad Ergonomic technique adds an additional 30 cm shift upward after the projection. Details on the input and output spaces of all considered techniques are given in Table 1.

Our interaction task is mid-air pointing in VR. Following the ISO 9241-9 standard, 13 targets with a diameter of 5 cm were placed on a circle of 30 cm diameter, resulting in an index of difficulty of 2.8 bits (cf. Figure 4). The center of the circle is placed 55 cm in front and 10 cm to the right of the right shoulder. We chose this placement, since most interactions with the right hand take place on the right side of the body. In each trial, the task is to move the virtual cursor as quickly

\(^{13}\)In a small preliminary study, we tried different input options for both techniques, and the variants we consider here proved suitable to reach all targets comfortably.
Simulating Interaction Movements via Model Predictive Control

Fig. 4. The 13 targets of the ISO 9241-9 standard pointing task, which are displayed separately in ascending order. The task is to move the cursor towards the active target and keep it inside until the next target is shown.

Fig. 5. (a) Image showing the user study. Motion is captured by tracking the movements of the LEDs. (b) The VR scene, which is perceived via a HMD, shows the target spheres (active: yellow; inactive: gray). (c) The MuJoCo simulation of the Virtual Cursor Ergonomic technique. The xyz-axes are colored in red, green, and blue, respectively. Important objects are displayed as follows. Green sphere: The virtual marker placed at the (physical) end-effector, i.e., the tip of the right index finger. Purple sphere: The virtual cursor (after applying the considered transfer function). Yellow sphere: active target. Gray spheres: inactive targets. The shadows show the orthogonal projections of the model for each dimension.

and accurately as possible toward the active target, which is represented as a yellow sphere with a diameter of 5 cm (cf. Figure 5(b)), and then hold within the target. As soon as the cursor reaches the target (with a velocity lower than 0.5 m/s to avoid early termination in case of overshoot), the next target according to the ISO 9241-9 standard is displayed after 500 ms.

5.2 User Study
We ran a user study for several reasons. First, the obtained experimental data can be used to create user-specific variants of the default biomechanical model introduced in Section 3.1. For example,
in Section 4, we introduce CFAT as a tool to identify the maximum voluntary torques at each DOF, given experimentally observed user trajectories. Second, having user data allows to evaluate the quality and realism of simulated movements against observed human motion. In particular, it can be used as reference data to compare simulations for different cost functions and weights, which allows to identify the cost function parameters that best replicate observed behavior (see Section 3.4.3).

We, therefore, asked participants to perform the task described above, using the presented interaction techniques.

5.2.1 Participants. We recruited six participants (Mean Age = 28.8, SD = 6.6, 4 Male, all right-handed) from our local university campus for the study. Half of the participants had previous experience of interaction in VR, and no participants suffered from perceptual or neuromotor impairments. In the following, we refer to the different users as U1, ..., U6. All four interaction techniques are varied within subjects.

5.2.2 Apparatus and Procedure. We used a Phasespace X2E\textsuperscript{14} motion capture system with a full-body suit to track the participants’ movements at 240 Hz. The movements of the upper extremity and torso were continuously tracked by 14 optical markers placed at anatomical landmarks. Participants were immersed in Virtual Reality using a HTC Vive Pro VR headset.\textsuperscript{15} The setup is shown in Figure 5(a). The VR scene and experimental setup were implemented in Unity3D\textsuperscript{16} using the SteamVR plugin\textsuperscript{17} (cf. Figure 5(b)). We aligned the coordinate systems of Phasespace and Unity as follows. We placed a Phasespace marker at the origin of a HTC Vive Pro VR controller. We then performed wanding of the interaction space using this controller, creating a set of 3D point pairs in both coordinate systems. We calculated a rigid transform between both coordinate systems using translation between the centroids to compute the translation component of the transformation, and the singular value decomposition to compute the rotation between the Phasespace and SteamVR coordinates [59].

Participants interacted with the VR scene using an end-effector marker placed at the tip of their right index finger. The movements are tracked in the Phasespace coordinate system. The cursor and target positions were only converted to the VR coordinate system right before the visualization. During the experiment, we logged the motion capture data and the experimental meta-data, as well as the timestamps at which the targets were hit.

Participants were informed about the ISO pointing task described in Section 5.1. Since we were interested in arm-only movements, participants were also instructed to only move their arm, while keeping the rest of the body as still as possible. This is important because the torso in our biomechanical model cannot move. Substantial torso movements would therefore distort the comparison between user and simulation. Nonetheless, we observed some slight torso movement in our motion capture data. For future user studies, it might therefore be helpful to use some kind of brace to keep the participant’s torso stable during the task.

Since the main objective of the user study was to collect movement data for different interaction techniques, we only tested a single index of difficulty to limit the impact of fatigue. After recording a T-pose for model scaling, participants put on the HMD and performed several movements for each interaction technique. During a warm-up phase, each interaction technique was trained for at least 30 movements. Afterward, all participants performed the complete ISO task consisting of

\textsuperscript{14}https://www.phasespace.com/x2e-motion-capture/.
\textsuperscript{15}https://www.vive.com/de/product/vive-pro/.
\textsuperscript{16}https://unity.com.
\textsuperscript{17}https://valvesoftware.github.io/steamvr_unity_plugin/.
13 subsequently shown targets 5 times per interaction technique, resulting in 65 movements per interaction technique and user, or 1,560 movements in total. In order to reduce fatigue, participants were asked to take a break of one minute between interaction techniques.

5.2.3 Processing Data and Inverse Kinematics. The raw motion capture data is preprocessed according to the common conventions for biomechanical analyses [3]. The marker data is first cleaned from artifacts caused by marker occlusions and reflections based on the condition values delivered by the motion capture system, and then by filtering out outliers (i.e., segments with a difference of more than four standard deviations from the mean). The resulting gaps in the data are linearly interpolated, while keeping track of the gaps. Afterward, the data is smoothed using a Kalman filter [72], and divided into individual aimed movements using the target switch times from the experiment.

We then run the OpenSim Inverse Kinematics (IK) tool for each movement of any considered participant and interaction technique individually. This tool computes the joint angles for each frame of motion capture data through solving an optimization problem. To this end, it applies the kinematic constraints and freely modifies the independent joint coordinates of the model to minimize the IK loss function, which is the weighted sum of squared distances between all virtual and the corresponding experimental markers. We use a larger weight for the end-effector marker than for the other markers, as it is critical to the considered pointing task to track the end-effector as accurately as possible.

In the experimental data, the time spans between target switch and movement onset differ substantially between trials. Since we are not interested in modeling reaction times, we decided to remove these frames from user data. To this end, we determine movement onset as the time at which the acceleration of the cursor reaches 1 m/s² for the first time. We also removed trials that started too early (i.e., the cursor left the previous target before the new target appeared), and movements of exceptional length (i.e., the movement duration deviated more than three standard deviations from the average duration for the considered participant and interaction technique) from the dataset. In total, 158 out of 1,560 recorded trials were removed, which is equivalent to 10.1%.  

Note that we have different time scales in simulation (2 ms) and data (1/240 s ≈ 4.17 ms). To be able to compare user and simulation trajectories on a moment-by-moment basis, we therefore align the two time series by applying linear interpolation on the user data.

5.3 Customized Models

In the following, we explain how the generic user model described in Section 3.1 is adjusted, both in terms of its biomechanical properties and in terms of the cost weights, which determine the tradeoff between the constituents of the cost functions introduced in Section 3.4.2.

First, we scale the models to match the kinematic and inertial properties of each participant of our user study using the OpenSim scaling tool. This tool computes ratios between pairs of markers recorded for a static posture in the experiment and the corresponding virtual markers attached to the model, only using the markers attached at the anatomical landmarks. These ratios are then used to scale the respective body segments. We ensure good quality of model scaling and marker adjustment by visually inspecting the resulting models with respect to experimental data. The scaling is then transferred from the OpenSim model to the MuJoCo model.

In addition, we adjust the joint limits to include all joint angles corresponding to the movement data of the respective participant. This is necessary because joint ranges are enforced in

---

114 of these trials are due to participants 2 and 5 occasionally starting their movements before the target switch.
Table 2. Joint Torque Ranges Obtained through CFAT

| Joint | U1 | U2 | U3 | U4 | U5 | U6 |
|-------|----|----|----|----|----|----|
| r−    | 12.74 | 16.12 | 22.08 | 26.12 | 14.92 | 19.20 | 14.33 | 18.38 | 10.99 | 15.16 | 21.64 | 26.73 |
| r+    | 20.43 | 20.43 | 18.36 | 18.36 | 20.92 | 15.49 | 14.69 | 17.08 | 10.05 | 17.82 | 2.41 | 6.42 |
| SE    | −3.35 | 0.70 | −4.28 | 1.00 | −3.88 | 0.71 | −4.03 | 0.98 | −3.54 | 1.37 | −5.11 | 2.41 |
| SR    | 0.25 | 5.08 | −0.17 | 5.36 | 0.21 | 5.88 | 0.48 | 5.54 | 0.42 | 4.81 | −0.92 | 6.42 |
| PS    | −1.82 | 1.71 | −1.36 | 1.15 | −3.06 | 2.73 | −0.81 | 0.58 | −0.41 | 3.68 | −1.42 | 1.14 |
| WD    | −2.11 | 2.00 | −1.60 | 1.35 | −1.98 | 1.72 | −0.95 | 0.57 | −1.87 | 1.64 | −1.36 | 1.07 |
| WF    | −1.86 | 0.78 | −1.52 | 0.72 | −1.76 | 0.71 | −1.24 | 0.41 | −1.77 | 1.02 | −1.36 | 0.43 |

The unsigned bold values are used as respective scaling ratios \( g \). (EA: Shoulder elevation angle; SE: Shoulder elevation; SR: Shoulder rotation; EF: Elbow flexion; PS: Pronation/Supination; WD: Wrist deviation; WF: Wrist flexion).

MuJoCo only via “soft” constraints, that is, high opponent forces are applied to postures outside the permissible region, which would reduce the reliability of the CFAT tool described in Section 4. However, it is important to note that the joint angles obtained from IK (see Section 5.2.3) are inherently dependent on the joint boundaries from the original OpenSim model (which can be found in Appendix B.1). This makes large deviations very unlikely.\(^{19}\)

After scaling, we obtain the maximum voluntary torques for each user by running CFAT for all available movements. An overview of the computed maximum and minimum torques are given in Table 2. We use the initial activations \( \sigma_0 \) and their derivatives \( \dot{\sigma}_0 \) also obtained by CFAT as valid initial values for the muscle dynamics used in our simulations as described in Section 3.1.2.

To obtain reasonable cost weights, we perform parameter fitting as described in Section 3.4.3 for each user and interaction technique. That is, we identify cost weights, i.e., user strategies, that best explain observed user behavior, both in terms of general behavior and intraindividual variance. This is in contrast to previous approaches, where parameters were fitted to replicate a single user trajectory [18, 47]. To ensure computational efficiency, we create simulation trajectories for five different movement directions from the ISO task, and compute the RMSE in terms of joint angles (cf. Equation (18)) between each simulation trajectory and the respective reference user trajectory. The loss function used for the cost weight fitting is thus given by Equation (19) with \( S = 5 \). As described in Section 3.4.3, we use CMA-ES as a derivative-free solver. We omit motor noise during the parameter fitting, since the resulting stochastic outcome for a given set of parameters would considerably complicate the parameter search.

In cases where the optimization did not converge, we ran CMA-ES for 24 hours for each setup and took the parameter set with the lowest RMSE. The resulting cost weights for each user and interaction technique are listed in Table B.2 in the Appendix.

### 5.4 Simulation

Our method cannot only be used to replicate existing movements, but also to predict movements in arbitrary conditions (i.e., for different interaction techniques, tasks, and user models). To evaluate the performance of our approach, however, we need to simulate movements with the same “prerequisites” as the users in the study we are comparing to. This includes the kinematic

\(^{19}\)Indeed, all user-specific joint limits were within a range of ±5 degrees around the default model values.
and inertial properties of the body as well as its initial joint configuration, which should coincide between simulation and user study.

Each aimed movement that was carried out in the user study is simulated separately. That is, for a given reference user trajectory (also referred to as Baseline U1/.../U6), we generate a corresponding simulation trajectory using the corresponding user model as described in Section 5.3 and the same interaction technique that was used in the study, i.e., we used the user-specifically scaled MuJoCo model and relevant cost weights. The simulation is shown in Figure 5(c), where our model performs a task with the Virtual Cursor Ergonomic interaction technique.

To ensure a fair comparison, we then set the torso position and orientation to that of the participant at movement onset. As mentioned above, the torso is fixed during simulation. Next, we set the initial state (including joint angles and velocities, aggregated muscle activations and their derivatives, and the virtual state of the interface, i.e., the cursor position), to the initial values of the reference user trajectory. We then synthesize the aimed movement using our MPC method. We want to emphasize that the optimization problem does not explicitly depend on the duration of the corresponding user movement. Instead, the receding time horizon approach allows to simulate arbitrarily long movements. Since comparing the resulting trajectories to that of the user study requires them to have equal length, we need to adjust the simulation trajectory to match the movement time of the participant in the particular trial. Therefore, the simulation stops when the movement time of the respective trial is reached.

This simulation is performed for all trials that passed the preprocessing, resulting in a total of 1,402 simulation trajectories.

6 RESULTS

In the following, we compare the ISO task trajectories resulting from our simulation to those observed during the user study described in Section 5.2. We recall from Section 5.3 that five trials from the dataset were used to fit the cost weights of the considered user and condition. In all evaluations, we exclude these five trials.

In Section 6.1, we first compare the three proposed cost functions regarding their ability to replicate and predict human movement trajectories. Using the Joint Acceleration Costs (JAC), which turn out to be most suitable for simulating human pointing movements, we show in Section 6.2 that our simulation predicts user trajectories with an accuracy that is comparable to or even better than between-user comparisons, while making use of biomechanically plausible joint postures. In Section 6.3, we show that the predicted trajectories continuously depend on the choice of the cost weights $r_1$ and $r_2$, aiding the parameter optimization and paving the road to simulating new user strategies, “tailored” to some desired movement characteristics such as speed. Finally, in Section 6.4, we discuss the effect of the MPC horizon $N$ and provide some general thumb rule on how to choose this hyperparameter.

For qualitative evaluation, we mainly focus on the following six quantities: cursor position and velocity time series, which are orthogonally projected onto the direct path between the initial and target position, as well as joint angles and velocities for both shoulder rotation and elbow flexion, as these are two of the most impactful joints for the considered mid-air movements. The angle and velocity plots of the five remaining joints are shown in Appendix B.

6.1 Comparison of Cost Functions: Joint Acceleration Costs Best Predict Human Motion

As described in Section 3.4.2, we use the following stage costs to simulate human movement in the ISO pointing task:
— **DC**: Distance and Control Costs (15),
— **CTC**: Distance, Control, and Commanded Torque Change Cost (16),
— **JAC**: Distance, Control, and Joint Acceleration Costs (17).

For each cost function, participant, and interaction technique, the respective cost weights $r_1$ (weight for control costs) and $r_2$ (weight for commanded torque change or JAC) are optimized to match joint angles between simulation and user data, as described in Section 5.3. The resulting parameter values are shown in Table B.2 in the Appendix. We evaluate the accuracy of our simulations in terms of predicted cursor and joint trajectories, both qualitatively and quantitatively.

There are clear qualitative differences between the three cost functions, as shown in Figure 6 for an example user study trial (black dashed lines; U4, Virtual Pad Identity, first movement from targets 7 to 8).

DC (blue solid lines) exhibits the highest velocities both in joint and cursor space, resulting in movements that are slightly faster than humans. The peak velocity tends to be too large, and for some trials, corrective submovements are required toward the end of the movement.

With CTC (green dashed lines), there is a considerable undershoot of the aimed target, with the cursor often not reaching the target at all within simulation time. As can be seen in the bottom left and right plots of Figure 6, penalization of commanded torque change seems to impose too restrictive constraints on the underlying joint dynamics, resulting in velocity time series of both elbow flexion and cursor that differ considerably from the typical bell-shaped velocity profiles observed in the user data.

In contrast, the simulation trajectories obtained from JAC (orange dash-dotted lines) match the human trajectories better: there are only slight differences between simulation and study in the projected cursor position and velocity profiles; it outperforms the other variants in terms of elbow flexion, and outperforms DC in shoulder rotation. Similar results can be obtained for the elevation angle, while pronation/supination as well as wrist deviation and flexion are predicted well by any of the considered cost functions (see Figure B.2 in the Appendix).

For quantitative comparison, boxplots containing the RMSEs of all ISO pointing movements for each considered cost function are shown in Figure 7, considering both cursor (top row) and joint...
space (bottom row). A breakdown of the cursor position and joint angle boxplots by individual users can be found in Figure B.3 in the Appendix. Kolmogorov–Smirnov tests showed that for each of the three cost functions, none of the considered RMSE distributions fits the assumption of normality (all values $p < 0.0001$). Thus, we carried out the non-parametric Wilcoxon Signed Rank tests with Bonferroni corrections.

For the following statements, details on the results of the statistical tests are provided in Table 3. The simulation trajectories generated with JAC Equation (17) replicate the respective user study trajectories significantly better than those generated with DC. The JAC trajectories also significantly outperform the CTC trajectories in terms of RMSE. Comparing CTC to DC, some RMSE quantities yield significant differences in favor of CTC, while for others, the cost function has no or only a small significant effect.

We thus conclude that, although both CTC Equation (16) and JAC Equation (17) open the door to a better fit through an additional weight parameter $r_2$, by far the best results in terms of replicating observed human trajectories is obtained by JAC Equation (17), which we focus on in the following.

### 6.2 Simulation vs. Users: MPC is Able to Simulate User Movement in Mid-Air Pointing

We compare the movements generated by our simulation with JAC to those from the user study in terms of both projected cursor trajectories and joint postures. In particular, we show that
a. Joint angles – Shoulder rotation

b. Joint angles – Shoulder elevation

Fig. 8. a. The joint angle ranges predicted by our simulation for different movements in the ISO task (orange solid lines) match those observed in our user study (black dashed lines) fairly well. The mean of all movements of a single participant/user model (U2) is shown together with the entire value ranges. b. In the ISO task, the Virtual Cursor Identity technique (black dashed lines in left plot) requires considerably higher shoulder elevation angles than the Virtual Cursor Ergonomic technique (black dashed lines in right plot). This characteristic difference is captured well by our simulation (orange solid lines).

(1) **Our simulated movements exhibit biomechanically plausible joint movements,**
(2) the produced cursor and joint trajectories predict human movements within between-user variability, and
(3) the method can predict motion of individual users.

(1) **Our simulated movements exhibit biomechanically plausible joint movements.** Figure 8 shows the shoulder rotation, shoulder elevation, and elbow flexion angles for one example user along with the corresponding simulation data. The user’s mean angles over time (black dashed lines) are captured well by our simulation (orange solid lines) for each joint. In addition, the range of joint angles applied during any of the considered movements (black area) exhibits the same structure as in our simulation (orange area). It should be noted that these ranges only make up a relatively small portion of the admissible model joint ranges (see Table B.1). The plots for the remaining four joints are shown in Figure B.4 in the Appendix. In addition, Figure B.5 in the Appendix depicts an example simulation and user trajectories for three different movement directions showing that direction-dependent differences in the joint kinematics are also captured by our model.

There are also characteristic differences in the joint ranges when simulating different interaction techniques. Due to its shifted input space, the Virtual Cursor Ergonomic technique allows
the participant to perform movements to arbitrary directions using considerably lower shoulder elevation angles than needed for the Virtual Cursor Identity technique, as can be inferred from the bottom plots in Figure 8 (black dashed lines, black areas). These technique-dependent movement characteristics are captured by our simulation, which predicts comparable joint ranges for both techniques (orange solid lines, orange areas).

In summary, our proposed MPC simulation is capable of generating motions that are plausible from a biomechanical perspective.

(2) The produced cursor and joint trajectories predict human movements within between-user variability. We argue that the movements JAC generates are within between-user variability. To this end, we first predict the movements of U1 with JAC, for all trials and interaction techniques, and compare the similarity in terms of RMSE with how well the trajectories from the remaining participants U2–U6 match those of U1. The results are displayed in Figure 9. Comparing the orange “JAC” whisker to the blue U2–U6 whiskers, Figure 9 shows that the movement trajectories generated by JAC are well within the RMSE ranges of other users, i.e., within the between-user variability. What stands out are the low RMSE values of JAC in the joint angles. It should be noted that this comparison is slightly biased because our simulation is necessarily initialized with the same joint angles as U1, while the other users might have started in slightly different postures. We extend this procedure, i.e., to infer the movements of one participant from JAC and from the respective remaining participants, to all participants, and combine the participants’ RMSE values in the blue “User vs. User” whiskers in Figure 10. The plots in Figure 10 show that the results for the special case of inferring U1’s movement can be extended to all participants.

We, therefore, conclude that our simulation predicts the movements of a given user not worse than other users on average, i.e., our simulation trajectories are within the between-user variability our dataset exhibits.

(3) We can predict movement of individual users. Besides these quantitative comparisons, we analyze how well we can predict the movement characteristics of individual users. In Figure 11, both the simulation (orange solid lines) and the corresponding reference user trajectories (black
Fig. 10. RMSE comparisons, where all user study trials of a single participant (e.g., U1) are compared to either movements predicted by our simulation (Simulation vs. User, orange whiskers) or by the remaining five participants (e.g., U2–U6) (User vs. User; blue whiskers). In contrast to Figure 9, the trials of all six participants are once used as baseline.

Fig. 11. Given an interaction technique (here: Virtual Pad Ergonomic) and a movement direction (here: movements from targets 1 to 2), the characteristic cursor and joint trajectories of an individual user (here: U4, black dashed lines; trajectories of the remaining users are shown as blue dotted lines for comparison) can be predicted by our simulation (orange solid lines).

dashed lines; for details, see Section 5.4) are shown for an example trial from the user study (U4, Virtual Pad Ergonomic, third movement from targets 1 to 2). We also show the respective trajectories of the remaining users for this trial (blue dotted lines). Figure 11 (left column) shows that we can match the characteristic projected position and velocity time series of a specific user. Moreover, the target is reached within a single ballistic movement, and the velocity time series exhibit
Fig. 12. Trajectories of the simulation for different cost weights \( r_1 \). It is clearly visible that an increase of \( r_1 \) results in slower, delayed movements toward the targets. Joint patterns are also (slightly) affected by the choice of \( r_1 \).

the bell-shaped velocity profile typically observed in aimed movements [45]. Figure 11 (middle and right columns) illustrates how the simulation is able to distinguish the baseline user from the remaining participants in terms of shoulder rotation and elbow flexion angles and velocities. Similar results can be observed for the remaining joints and interaction techniques, as shown in Appendix B.4.

In summary, these findings show that our simulation generates biomechanically plausible joint postures, it is capable of predicting human trajectories that are within the between-user variability of the respective interaction technique, and it allows to replicate characteristic movement patterns of individual users.

6.3 Effects of the Cost Weights

To be able to replicate characteristic movement patterns of individual users, we need to determine the cost weights \( r_1 \) and \( r_2 \) in JAC (cf. Equation (17)). To understand how much the simulation results depend on these weights, we show in this section how the trajectories change when we alter the weights, and argue that these insights can be used to generate new user strategies.

The most important insight is that the movement trajectories exhibit a continuous dependence on \( r_1 \) and \( r_2 \), i.e., if \( r_1 \) and/or \( r_2 \) change only slightly, the movement trajectories (of cursor and joints) also change only slightly. With this, we provide indicators how the cost weights should be changed in order to generate, e.g., slower movements. To this end, we start by analyzing the effects of \( r_1 \) and \( r_2 \) on the movement trajectories.

To avoid distorting the effects, we omit the motor noise from all simulations in this section.

In Figure 12, projected cursor and joint trajectories are shown for 10 different values of the control cost weight \( r_1 \), with constant joint acceleration cost weight \( r_2 = 0.00012 \) (U4, Virtual Cursor Ergonomic, first movement between targets 4 and 5). There is a clear decrease in peak velocity, as \( r_1 \) increases, resulting in considerably slower movements with target reached later. Moreover, the projected cursor velocity profile becomes more right-skewed, which can be explained by the
increased incentive to apply lower torques per time step (note that the penalization of squared control signals incentives the use of multiple small control signals instead of one large control signal). This suppressive effect of increased control costs can also be observed in the shoulder rotation plots, where larger \( r_1 \) values result in considerably less negative joint velocities at the beginning of the movement, which need to be compensated later in order to reach the target. The elbow flexion is also affected by changes in the control costs weight, however, with comparably small impact (note the considerably smaller joint angle range). Note that for very large \( r_1 \) values, the only relevant objective is to reduce the control cost, i.e., it is optimal to apply no controls during the entire movement, which causes the arm to fall.

In Figure 13, the effects of the JAC weight \( r_2 \) are shown for the same participant, interaction technique, and trial, using a fixed intermediate control costs weight \( r_1 = 0.016 \). The effect on projected cursor trajectories is qualitatively comparable to that of \( r_1 \), albeit considerably more pronounced. Without JAC, i.e., \( r_2 = 0 \) (black lines), the projected velocity time series exhibits a very high peak velocity that is compensated by a corrective submovement starting after \(~ 450\) milliseconds, resulting in a very fast movement towards the target. For shoulder rotation and elbow flexion, the changes in both joint angles and velocities strongly decrease as \( r_2 \) increases. In particular, the characteristic joint patterns that can be observed for \( r_2 = 0 \) (black lines) become more and more flattened as joint accelerations are more penalized.

The effects of the two cost weights on the remaining joints are depicted in Figures B.11 and B.12 in the Appendix, respectively.

To examine how well a certain cost weight pair fits one specific user, the surface plot in Figure 14 shows a simulation vs. user comparison (U4, Virtual Cursor Ergonomic) for different combinations of cost weights. Here, the z-axis shows the mean RMSE used to measure the similarity between simulation and user data trajectories (see Section 3.4.3) with respect to cursor position across all trials of U4 using the Virtual Cursor Ergonomic technique. As can be seen, the surface is clearly convex. While convexity of the RMSE with respect to the cost weights can neither be guaranteed nor expected in general, it ensures that there exists a unique global optimum, towards which most

Fig. 13. Trajectories of the simulation for different cost weights \( r_2 \). The clear cursor and joint profiles that can be observed for \( r_2 = 0 \) (black lines) become more and more smooth as \( r_2 \) increases, resulting in a similar effect as observed for \( r_1 \).
OCP solvers (including the one we use) are guided. In the considered case, the global optimum is located at \( r_1 = 0 \) and \( r_2 \approx 1.1 e - 4 \) (black colored valley).\(^{20}\)

In summary, our analysis of cost weight effects shows that both the cursor and joint trajectories predicted by the closed MPC loop continuously depend on the cost weight parameters. This has two major advantages:

First, the parameter fitting process in finding cost weights that best reflect specific user behavior exhibits a certain robustness. This facilitates computing optimal weights and ensures that cost weights close to the optimum will already provide good results. Second, it allows to generate new user strategies by tweaking the cost weight parameters.

### 6.4 Effects of the MPC Horizon

To better understand the impact of the MPC horizon \( N \), which may need manual adjustments depending on the task under consideration, in the following, we analyze how the simulated movements change with increasing horizon.

We also omit the motor noise from all simulations done in this section to avoid distorting the effect of \( N \).

As depicted in the top left plot of Figure 15 for the same trial as used in Section 6.3, the MPC horizon \( N \), which determines how many future steps are taken into account to select the control at a certain time step, has a considerable effect on the resulting closed-loop trajectories. Choosing \( N \) too small results in movements that are either too slow to reach the target at all (\( N = 2 \), black line), cross the target but do not return within a reasonable time (\( N = 3, 4 \), blue lines), or exhibit some considerable overshoot (\( N = 5 \), grass green line). Starting from \( N = 8 \), the differences in cursor and joint trajectories are quite small and hardly visible anymore. This suggests that a prediction horizon of \( 8 \cdot 40 \text{ ms} = 320 \text{ ms} \) is sufficient to adequately solve our OCP via MPC.

This is also confirmed by the quantitative comparison of MPC horizons shown in Figure 16, where we again computed the mean RMSE on cursor positions, considering all trials of U4 using

---

\(^{20}\)Note, that this does not exactly coincide with the cost weight pair found by our fitting process. This is due to the facts, that during the fitting, we use RMSE based on joint angles, and that out of computational reasons, we only considered five movements to evaluate a cost weight pair.
Fig. 15. Projected cursor and joint trajectories of one trial for varying MPC horizon $N$, using the JAC, without control noise.

Fig. 16. Effect of the MPC horizon $N$ on the mean RMSE between JAC and user data in terms of cursor positions, considering all trials of U4 with the Virtual Cursor Ergonomic technique. The performance clearly deteriorates when the MPC horizon $N$ is set too low (i.e., below $N = 8$).

the Virtual Cursor Ergonomic technique, for different MPC horizons. The performance of our simulation clearly deteriorates when the MPC horizon $N$ is set too low, i.e., $N \leq 6$. Interestingly, user trajectories are best explained by an intermediate MPC horizon of $N = 13$, while a too large horizon ($N \geq 16$) results in slightly worse mean RMSE for this interaction technique. Since the computational time exponentially increases with $N$, we decided to use the lowest MPC horizon replicating human behavior sufficiently well for our simulations, that is, $N = 8$.

7 DISCUSSION AND LIMITATIONS

Our framework allows to simulate interaction movements with optimal control (cf. Figure 1). It is applicable to a wide range of interaction techniques and can be adjusted to simulate different individual users. In order to help the HCI community to explore the simulation of interaction with MPC, we provide our Python code as open source at https://github.com/mkl4r/sim-mpc. In this section, we discuss how our framework can be applied, what issues should be paid particular attention to, and interesting ideas for future work.
In order to help using the framework, Section 3 discusses possible adjustments of the individual parts of the framework, and gives some ideas on how to tailor these to specific interaction techniques, users, and conditions. Moreover, we provide an example-based step-by-step guide on how to apply our framework to novel tasks in Section 8.

One point to consider is that in our simulations, the torso is fixed in space. This is obviously only a reasonable assumption when simulating interaction techniques that do not require substantial torso movement. Enabling natural torso movement is less trivial than one might think. One solution is to provide multiple “virtual” joints for the added six degrees of freedom of the torso. However, according to our experience, such virtual joints can easily lead to unnatural movements. Alternatively, the lower body can be integrated in the biomechanical simulation, leading to the added task for the controller of balancing the body.

Our presented framework assumes complete observation, i.e., complete knowledge of the system state (e.g., joint angle and cursor position). In reality, humans observe their environment, e.g., through proprioception or visual perception, and have to deal with limited information, e.g., they have to estimate the target position from an image on a screen. To control such a system with imperfect observation, it would be an interesting future task to add an observer.

Another aspect is that we only investigated movements to a single, relatively big, target size, which do not require extensive corrective submovements. If one is interested in submovements and movement times for different target sizes, MPC is certainly an interesting way to model these. Our model exhibits some submovements due to the interplay of a limited MPC horizon and movement noise. A promising direction to increase the realism of submovements, and model the change of movement times in response to different target sizes would be the introduction of an observation delay, an observer to account for both motor and sensory noise, and including perception of the target size. All of these are compatible with and can be integrated in the MPC framework.

While introduced in general terms, we apply our SimMPC framework to the use case of mid-air pointing throughout the article. In the future, it would be interesting to evaluate movements that our framework predicts for different interaction techniques and tasks. Similarly, our analysis of between-user variability was based on data from a relatively homogeneous user group (all right-handed adults without impairments and with similar backgrounds). However, since our approach explicitly provides the possibility to capture user-specific characteristics (e.g., using scaled MuJoCo bodies or maximum voluntary torques obtained by CFAT), comparable fits between given user data and our simulation can be expected for rather diverse target groups.

7.1 Why MPC?

We have seen in Section 6.4 that, once a cost function has been chosen (in our case, JAC), the choice of the MPC horizon $N$ is the key governing factor. In practice, some trial and error is needed, but also possible here, to control the tradeoff between computation time and quality of the movement trajectory. To this end, we have provided practical guidance in Section 6.4. Once we have such an $N$, then MPC “works” for this application.

In comparison to some other models of interaction, MPC is particularly interesting, because it allows for an analytical understanding of the dynamics of the interaction loop. MPC has been used successfully for other applications; it is no coincidence that it is performing well in this particular setting. In this article, we have relied on user data and performed an extensive analysis on the performance of MPC. However, if one wants to very quickly check whether the effort of using MPC will be worthwhile in their application, a good indicator is the so-called turnpike property. It states that the computed optimal trajectories remain close to the so-called optimal steady-state, most of the time. As the name suggests, it is the optimal state to remain in with respect to stage costs $\ell$. In our JAC case, it is determined by the state costs (e.g., keeping the cursor near the target)
and control or effort costs (e.g., using a joint posture in the process that is not strenuous or tiring). Usually, this optimal steady-state is not known analytically, so the first step is to approximate it numerically. This can be done by solving the OCP Equation (20) over the time frame of interest. Then, instead of only applying the first value of the resulting optimal control sequence as in the MPC algorithm (Section 3.5), we compute the so-called open-loop optimal trajectory by simply applying the full optimal control sequence. For this purpose, the parameter \( N \) should be chosen as large as possible—in our following example, we have chosen \( N = 16 \). Then we compare this result to open-loop optimal trajectories for smaller \( N \), by plotting their difference to the reference trajectory (large \( N \)). This is visualized in Figure 17.

Figure 17 shows the typical turnpike behavior: First, for all \( N \) the corresponding open-loop trajectories at some point start deviating more and more from the reference trajectory. Second, this happens later and less pronounced the higher \( N \) is. The turnpike property is an essential part in theorems that guarantee that MPC produces “sensible” results, i.e., MPC “works” [15]. This assurance provides trust and reliability, is one of the great strengths of MPC, and is the reason why it is used, e.g., in industry applications [52].

Similar guarantees exist for other controllers, such as LQR or LQG used, for example, in Reference [18]. However, unlike MPC, these cannot handle nonlinear system dynamics, which regularly occur in HCI due to nonlinear. Possibly even more important for HCI, they cannot directly handle state and control constraints, but have to resort to soft constraints via penalty terms in the cost function. These constraints include joint angle and/or torque limits imposed by human biomechanics, body position constraints when interfered by or interacting with physical objects and devices (e.g., desk, gamepad, or monitor), and boundaries on the location of the virtual objects (e.g., due to fixed bounding boxes of application windows or limited screen size).

If, on the other hand, guarantees take a back seat, RL most certainly enters the picture. RL is very powerful, can handle nonlinearities and even state and control constraints, and thus is used in many applications. In particular, RL can be used to control a biomechanical model as well [16]. However, computationally it can be very demanding to compute a single optimal movement. This is because RL learns a policy to then sample movements. This policy may have to be completely

\[ N \]
\[ \frac{2.5}{0.2} \]
\[ \frac{2.0}{0.1} \]
\[ \frac{1.5}{0.05} \]
\[ \frac{1.0}{0.0} \]
\[ 0 \to 50 \to 100 \to 150 \to 200 \to 250 \]

\[ \frac{0.08}{0.06} \]
\[ \frac{0.04}{0.02} \]
\[ \frac{0.00}{0.0} \]

\[ 0 \to 50 \to 100 \to 150 \to 200 \to 250 \]

\[ \frac{0.25}{0.2} \]
\[ \frac{0.15}{0.1} \]
\[ \frac{0.1}{0.05} \]
\[ \frac{0.05}{0.0} \]

Fig. 17. Euclidean norm between the cursor position (left) and the aggregated joint angles (right) of open-loop trajectories with different MPC horizons \( N \) and a reference open-loop trajectory for \( N = 16 \). The Turnpike property is reflected in the fact that the trajectories stay longer in the vicinity of the reference as the horizon \( N \) increases. All simulations use the same trial of U4 and the Virtual Cursor Identity technique (defined in Section 5.1).

\[ \text{We note that getting the policy can be advantageous depending on the use case.} \]
retrained if the interaction technique is changed, e.g., a different input origin for the Virtual Pad is used. In contrast, MPC acts as a complexity reducer in time: instead of directly solving an optimization problem for the full movement time, in MPC, we consider subproblems on smaller time horizons, which are much easier to solve due to exponentially increasing complexity the longer the movement lasts.

From our point of view, it is therefore promising to combine the respective strengths of MPC and RL (or, more generally, Machine Learning techniques) in future applications. Following the current trend, MPC is more and more augmented by data-driven techniques (especially from the control theory community). To put it in the words of Benjamin Recht (UC Berkeley, “Reflections on the Learning-to-Control Renaissance” Keynote at the 2020 IFAC World Congress): I still remain baffled by how Model Predictive Control is consistently underappreciated. [...] there are really very few people, especially in the Machine Learning community, who are trying to analyze why MPC is so successful, especially when it’s coupled with Machine Learning.

8 APPLYING THE SIMMPC FRAMEWORK STEP-BY-STEP

This section gives a step-by-step guide on how to use our framework for other interaction tasks or techniques. In the following, we show how to simulate the task of tracking a moving target using ray casting with a handheld VR controller. In our example, the task is to track a sphere that is constantly moving in a circle in front of the user. The approach consists of the following steps:

1. Define user (group) and model,
2. implement the input/output device in MuJoCo,
3. define and implement the interface dynamics,
4. choose the objective function (and optimize cost weights),
5. run simulations, and,
6. obtain trajectory data and summary statistics.

1) We start by defining the user group and model, see Section 3.1. Formally, the model corresponds to \( f_{user} \) in Equation (1). If the goal is to generally evaluate an interaction task/technique, using one (or several) of the six provided user models of healthy humans is sufficient. If one aims to build an interaction technique for a more specific user (group), we recommend to first collect some movement data in a preliminary study, and use this data to adjust the model scaling and maximum voluntary torques as described in Section 5.3. This data does not have to stem from the task under consideration, but should contain similar movements.

2) Next, we need to implement the input/output devices in MuJoCo, see Section 3.2. Formally, this step corresponds to defining \( f_{dev} \) and \( x_{dev} \) in Equation (5). In our case, we add a body to the MuJoCo model (more specifically, to the hand of the model) that corresponds to a handheld controller, with appropriate mass and inertial properties. Next, we attach a marker to the controller, which is used to infer its position and orientation (i.e., we define the physical end-effector \( x_{ee} \)). The marker should be placed such that this information match those provided by the real controller. Adding a 3D mesh and adjusting the rigid finger positions such that the hand is actually holding the device is optional but can help to place this marker correctly, improves visualization, and is recommended when collisions are expected, e.g., if the interaction space is physically limited. We define the state of the input device \( x_{dev} \) as the position of the (physical) end-effector \( x_{ee} \) and its orientation using a direction vector \( n_{ee} \). The function \( f_{dev} \) defines how we obtain those values from the MuJoCo simulation, i.e., using the position of the marker as \( x_{ee} \) and calculate \( n_{ee} \) using the marker’s orientation. Note that any physical action that is to have an effect on the virtual environment must be modeled in the state of the input device (e.g., button press events could be modeled using a boolean entry in the state).
(3) In the third step, we implement the interface dynamics, see Section 3.3. One part of the interface state \( x_{if} \) is the current position of the sphere \( x_s \). The movement of the sphere is defined by the interface dynamics \( f_{if} \). Using the position and orientation of the controller given by \( x_{dev} \) from the previous step, we can calculate the distance \( x_d \) between the ray that is cast from the controller and the center of the sphere. Note that we do not explicitly need a transfer function \( f_{if} \) here, since we can use the position and orientation of the controller directly. The complete interface state is then given by \( x_{if} = (x_s, x_d) \). This concludes the system dynamics of the OCP in Equation (13) that we want to solve.

(4) Next, we define the stage costs for each time step that depends on the state of our system and the control. For pointing, we suggest using a combination of distance, control, and JAC (JAC, see Equation (17)). While the distance costs reflect the actual task objective, adding control costs is generally recommended for two reasons. First, these costs create an incentive to execute the movement with as little effort as possible, and second, they regularize the optimization procedure from a mathematical point of view. When designing stage costs for tasks other than pointing, we recommend using continuous and convex functions as they lead to a better optimization performance.

In the considered case of tracking with ray casting, we decide for a combination of the squared norm of both the control and the joint acceleration vectors plus the norm of \( x_d \) since the goal is to keep the ray inside the sphere.\(^{22}\) The stage cost is therefore given by
\[
\ell(x(k), u(k)) = \|x_d(k)\|^2 + r_1\|u(k)\|^2 + r_2\|x_{qac}(k)\|^2,
\]
with cost weights \( r_1, r_2 > 0 \).

Since we use a weighted combination of different cost terms, we need to find suitable cost weights. In the considered case of tracking, we expect scaling both \( r_1 \) and \( r_2 \) relative to the modified distance costs to be sufficient to achieve reasonable results. For more severe changes to the task, or if reference user data is already available (e.g., from a preliminary study), running a parameter optimization as described in Section 3.4.3 is recommended to improve the simulation results.

(5) Before running the simulation, the MPC horizon should be chosen long enough for the considered task (see discussion in Section 6.4). Additionally, a termination criterion should be defined, which determines when to stop the simulation. Here, this can be directly inferred from the instructions, i.e., the task is to stay inside the target sphere for a certain amount of time. Alternatively, the simulation can also be run for a certain time, independent of the completion of the task. Finally, to be able to run a simulation, we need to define the initial state of the system. This consists of an initial posture of the biomechanical model (i.e., joint angles and velocities), initial activations of the muscles and their derivatives and the initial state of the input device and interface. This initial state can either be obtained from a user study or by running preliminary simulations with arbitrary initial states.

(6) The output of the simulation includes trajectories for all state variables such as joint angles, angular velocities, or position of virtual objects. In particular, all quantities obtained by a typical motion-capture based user study are easily available. To obtain summary statistics, we can include noise and run multiple simulations to collect an entire set of trajectories reflecting the predicted trial-to-trial variability for the task under consideration.

9 CONCLUSION

We have presented SimMPC, a framework to simulate movements during interaction with computers, combining biomechanical modeling with MPC. It allows to predict kinematic and dynamic quantities of both physical and virtual objects, including cursor positions, joint angles and velocities, and aggregated muscle recruitments, on a moment-by-moment basis, providing richer

---

\(^{22}\)While we do not explicitly include the size of the sphere in the cost function, it can be used in the later evaluation.
information than summary statistics. Our framework allows easy combination of biomechanical models of the human body with arbitrary interaction techniques and tasks, which was not possible with existing approaches that relied on linear optimal control methods. As a use case, we have applied our approach to a joint-actuated state-of-the-art model of the upper extremity, and considered four mid-air interaction techniques for the task of ISO pointing in VR. We have shown that MPC is able to simulate user movement in mid-air pointing both in terms of cursor and joint trajectories, with an accuracy within observed between-user variability. Comparing three different cost functions, the combination of distance, control, and JAC was shown to best explain the movements observed in our user study.

In a practical sense, our approach allows to predict movement of a given individual user, and can be used to generate new user strategies. Moreover, we provide advice on how to apply our framework to simulate interactions for different target user groups, interaction techniques, or interaction tasks. We have also introduced CFAT, a novel tool to compute the applied torques underlying a given joint angle sequence, which we have used to infer the maximum voluntary torques that were applied in the considered ISO pointing task. We have made our code and data publicly available (https://github.com/ml4r/sim-mpc).

The combination of biomechanical simulations with MPC could open the door for online optimization and customization of user interfaces and interaction techniques, based on the predictions of a “digital twin” simulation running in the background. The ability to evaluate the entire interaction loop between humans and computers in terms of efficiency or ergonomics with the help of realistic simulations could allow for partial replacement of costly and time-consuming user studies in the future. We, therefore, believe that the optimal control perspective on interaction and its simulation via MPC provide an interesting interpretation of and useful tools for Human–Computer Interaction.

APPENDICES

A  VIRTUAL PAD TRANSFER FUNCTION

To obtain the cursor position of the Virtual Pad, we have to project the end-effector position \( x_{ee} \) onto the input plane and transfer it onto the output plane. The projection onto the input plane is given by

\[
\text{Proj}_I(x_{ee}) = x_{ee} - ((x_{ee} - \omega_I) \cdot n_I) n_I,
\]  

where the operator \( \cdot \) denotes the inner product, \( \omega_I \) is the input space origin, and \( n_I \) is the input space normal. Note that, while the projected position could be denoted in 2D coordinates of the input plane, the projection maps onto the position in global 3D coordinates to be able to transfer this point correctly onto the output plane. To rotate this new point correctly, we compute the rotation matrix \( R \) that rotates the input normal vector \( n_I \) such that it equals the output normal vector \( n_O \):

\[
R = (1 - C)aa^\top + \begin{pmatrix} C & -a_3\tilde{C} & a_2\tilde{C} \\
-a_3\tilde{C} & C & -a_1\tilde{C} \\
-a_2\tilde{C} & a_1\tilde{C} & C \end{pmatrix},
\]

where \( a \) is the rotation axis, i.e.,

\[
a = (a_1, a_2, a_3)^\top = \frac{n_I \times n_O}{\|n_I \times n_O\|},
\]

\( aa^\top \) is the outer product of \( a \) with itself, and \( C \) and \( \tilde{C} \) are the cosine and sine of the angle between the normals, i.e.,

\[
C = \frac{n_I \cdot n_O}{\|n_I\| \|n_O\|}, \quad \tilde{C} = \sqrt{1 - C^2}.
\]
Since rotation is defined around the global origin, we subtract $\omega_I$ before rotating, i.e., for some point $y \in \mathbb{R}^3$:

$$\text{Rot}_{IO}(y) = R(y - \omega_I).$$  \hspace{1cm} (A.5)

As the last step, we need to translate back onto the output plane by adding the output origin $\omega_O$. The overall transfer function is thus given by

$$f(t(x_{ee}) = \text{Rot}_{IO}(\text{Proj}_I(x_{ee})) + \omega_O = R(x_{ee} - ((x_{ee} - \omega_I) \cdot n_I) n_I - \omega_I) + \omega_O.$$ \hspace{1cm} (A.6)

### B Supplementary Figures and Tables

#### B.1 MuJoCo Model

Table B.1. Default Joint Angle Ranges of the Upper Extremity Model [16]

| Joint                  | Angle Ranges (deg/rad) | Min. | Max. |
|------------------------|------------------------|------|------|
| Elevation Angle        | $-90 \ - \frac{1}{2} \pi$ | 130  | $\frac{13}{18} \pi$ |
| Shoulder Elevation     | $0 \ - \frac{1}{2} \pi$  | 180  | $\frac{1}{2} \pi$   |
| Shoulder Rotation      | $-90 \ - \frac{1}{2} \pi$ | 20   | $\frac{1}{2} \pi$   |
| Elbow Flexion          | $0 \ - \frac{1}{2} \pi$  | 130  | $\frac{13}{18} \pi$ |
| Pronation/Supination   | $-90 \ - \frac{1}{2} \pi$ | 90   | $\frac{1}{2} \pi$   |
| Wrist Deviation        | $-10 \ - \frac{1}{18} \pi$ | 25   | $\frac{17}{18} \pi$ |
| Wrist Flexion          | $-70 \ - \frac{7}{18} \pi$ | 70   | $\frac{7}{18} \pi$  |

#### B.2 CFAT

(Applied) Joint Torques – OpenSim Inverse Dynamics (ID) vs. Our CFAT Tool

Figure B.1. Joint torques of all participants and interaction techniques, both computed using OpenSim Inverse Dynamics (ID, green) and our proposed CFAT tool (yellow). For each DOF, the colored boxes show the respective interquartile ranges (25% to 75% quantiles) and the whiskers correspond to the minimum and maximum torques after removing some outliers (see Section 4).

ACM Transactions on Computer-Human Interaction, Vol. 30, No. 3, Article 44. Publication date: June 2023.
B.3 Comparison of Cost Functions

Table B.2. The Cost Weights Obtained by the CMA-ES Parameter Optimization for Each Participant, Condition, and Cost Function

| Participant | Technique                  | $r_{DC}^1$ | $r_{CTC}^1$ | $r_{CTC}^2$ | $r_{JAC}^1$ | $r_{JAC}^2$ |
|-------------|----------------------------|------------|-------------|-------------|-------------|-------------|
| U1          | Virtual Cursor ID          | 0.51       | 1.49 · 10^{-2} | 1.15 · 10^{-4} | 1.18 · 10^{-5} | 4.25 · 10^{-5} |
| U1          | Virtual Cursor Ergonomic   | 0.22       | 1.94 · 10^{-5} | 1.18 · 10^{-4} | 9.16 · 10^{-5} | 2.23 · 10^{-4} |
| U1          | Virtual Pad ID             | 1.71 · 10^{-2} | 3.01 · 10^{-2} | 4.79 · 10^{-5} | 5.61 · 10^{-2} | 3.13 · 10^{-5} |
| U1          | Virtual Pad Ergonomic      | 8.13 · 10^{-6} | 1.03 · 10^{-5} | 4.2 · 10^{-6}  | 3.03 · 10^{-9} | 1.56 · 10^{-5} |
| U2          | Virtual Cursor ID          | 0.11       | 5.3 · 10^{-2}  | 7.13 · 10^{-8} | 7.85 · 10^{-2} | 4.66 · 10^{-5} |
| U2          | Virtual Cursor Ergonomic   | 0.33       | 0.19          | 2.85 · 10^{-5} | 2.59 · 10^{-2} | 1.68 · 10^{-4} |
| U2          | Virtual Pad ID             | 0.34       | 1.1 · 10^{-2}  | 8.66 · 10^{-5} | 4.57 · 10^{-2} | 1.16 · 10^{-4} |
| U2          | Virtual Pad Ergonomic      | 1.51 · 10^{-7} | 1.14 · 10^{-7} | 8.95 · 10^{-7} | 6.18 · 10^{-9} | 2.71 · 10^{-5} |
| U3          | Virtual Cursor ID          | 3.86 · 10^{-2} | 2.04 · 10^{-2} | 3.54 · 10^{-7} | 2.13 · 10^{-2} | 1.1 · 10^{-6}  |
| U3          | Virtual Cursor Ergonomic   | 1.72 · 10^{-2} | 6.77 · 10^{-5} | 5.04 · 10^{-6} | 2.08 · 10^{-4} | 3.8 · 10^{-7}  |
| U3          | Virtual Pad ID             | 0.25       | 2.79 · 10^{-4} | 2.46 · 10^{-5} | 1.67 · 10^{-2} | 1.02 · 10^{-5} |
| U3          | Virtual Pad Ergonomic      | 2.72 · 10^{-2} | 1.81 · 10^{-8} | 1.11 · 10^{-6} | 2.95 · 10^{-2} | 1.83 · 10^{-8} |
| U4          | Virtual Cursor ID          | 7.84 · 10^{-2} | 1.89 · 10^{-3} | 3.84 · 10^{-7} | 1.17 · 10^{-2} | 1.05 · 10^{-4} |
| U4          | Virtual Cursor Ergonomic   | 0.3        | 1.26 · 10^{-3} | 5.18 · 10^{-5} | 1.57 · 10^{-4} | 1.32 · 10^{-5} |
| U4          | Virtual Pad ID             | 6.7 · 10^{-2}  | 7.18 · 10^{-4} | 9.19 · 10^{-5} | 9.28 · 10^{-3} | 5.07 · 10^{-5} |
| U4          | Virtual Pad Ergonomic      | 5.87 · 10^{-2} | 2.05 · 10^{-3} | 2.48 · 10^{-5} | 3.08 · 10^{-6} | 5.73 · 10^{-5} |
| U5          | Virtual Cursor ID          | 0.12       | 9.5 · 10^{-4}  | 2.09 · 10^{-4} | 1.31 · 10^{-2} | 3.03 · 10^{-4} |
| U5          | Virtual Cursor Ergonomic   | 0.15       | 0.15          | 6.79 · 10^{-14} | 0.13 | 1.05 · 10^{-5} |
| U5          | Virtual Pad ID             | 6.36 · 10^{-2} | 6.15 · 10^{-2} | 1.55 · 10^{-12} | 4.3 · 10^{-2} | 1.3 · 10^{-4}  |
| U5          | Virtual Pad Ergonomic      | 3.73 · 10^{-2} | 8.03 · 10^{-2} | 5.18 · 10^{-5} | 9.05 · 10^{-2} | 1.52 · 10^{-4} |
| U6          | Virtual Cursor ID          | 0.1        | 1.1 · 10^{-2}  | 1.92 · 10^{-6} | 1.76 · 10^{-2} | 1.3 · 10^{-5}  |
| U6          | Virtual Cursor Ergonomic   | 0.35       | 2.45 · 10^{-4} | 1.94 · 10^{-5} | 1.05 · 10^{-9} | 5.2 · 10^{-6}  |
| U6          | Virtual Pad ID             | 0.39       | 8.09 · 10^{-3} | 1.07 · 10^{-5} | 9.94 · 10^{-3} | 1.36 · 10^{-5} |
| U6          | Virtual Pad Ergonomic      | 0.11       | 3.64 · 10^{-3} | 1.79 · 10^{-6} | 0.3 | 9.56 · 10^{-10} |

Figure B.2. Projected joint trajectories for one trial of U4 for the Virtual Pad Identity technique. The Joint Acceleration Costs (JAC; orange dashdotted lines) qualitatively explain observed user behavior best. For pronation/supination, deviation, and flexion, all considered cost function show a good fit to observed user behavior (note the small angle ranges for these joints). Remaining joints are shown in Figure 6.
Figure B.3. Comparison of the different cost functions in terms of cursor positions and joint angles, separated by user. The boxplots show the RMSE of all trials.

B.4 MPC can Simulate User Movements

Figure B.4. The joint angle ranges predicted by our simulation for different movements in the ISO task (orange solid lines) match those observed in our user study (black dashed lines) fairly well. The mean of all movements of a single participant/user model (U2) is shown together with the entire value ranges. Remaining joints are shown in Figure 8.
Figure B.5. U4 and simulation trajectories for different directions in the Virtual Pad Ergonomic condition. In the user study (dashed lines), the joint angle trajectories differ significantly between movement directions (the target configuration is depicted in Figure 4). These characteristic differences are captured well by our simulation (solid lines).

Figure B.6. Given an interaction technique (here: Virtual Pad Ergonomic) and a movement direction (here: movements from targets 1 to 2), the characteristic cursor and joint trajectories of an individual user (here: U4, black dashed lines; trajectories of the remaining users are shown as blue dotted lines for comparison) can be predicted by our simulation (orange solid lines). Remaining joints are shown in Figure 11.
Figure B.7. Given an interaction technique (here: Virtual Cursor ID) and a movement direction (here: movements from targets 8 to 9), the characteristic cursor and joint trajectories of an individual user (here: U2, black dashed lines; trajectories of the remaining users are shown as blue dotted lines for comparison) can be predicted by our simulation (orange solid lines).
Figure B.8. Given an interaction technique (here: Virtual Cursor Ergonomic) and a movement direction (here: movements from targets 8 to 9), the characteristic cursor and joint trajectories of an individual user (here: U3, black dashed lines; trajectories of the remaining users are shown as blue dotted lines for comparison) can be predicted by our simulation (orange solid lines).
Given an interaction technique (here: Virtual Pad ID) and a movement direction (here: movements from targets 8 to 9), the characteristic cursor and joint trajectories of an individual user (here: U4, black dashed lines; trajectories of the remaining users are shown as blue dotted lines for comparison) can be predicted by our simulation (orange solid lines).
Figure B.10. Given an interaction technique (here: Virtual Pad Ergonomic) and a movement direction (here: movements from targets 8 to 9), the characteristic cursor and joint trajectories of an individual user (here: U6, black dashed lines; trajectories of the remaining users are shown as blue dotted lines for comparison) can be predicted by our simulation (orange solid lines).
B.5 Effects of the Cost Weights

Figure B.11. Trajectories of the JAC simulation for different cost weights $r_1$. Remaining joints and cursor trajectories are shown in Figure 12. Shown are simulated movements for U4 in the Virtual Cursor Ergonomic condition.

Figure B.12. Trajectories of the JAC simulation for different cost weights $r_2$. Remaining joints and cursor trajectories are shown in Figure 13. Shown are simulated movements for U4 in the Virtual Cursor Ergonomic condition.

B.6 Effects of the MPC Horizon

Figure B.13. Joint trajectories of one trial for varying MPC horizon $N$, using the JAC, without control noise. Remaining joints and cursor trajectories are shown in Figure 15. Shown are simulated movements for U4 in the Virtual Cursor Ergonomic condition.
REFERENCES

[1] Carlos Andujar and Ferran Argelaguet. 2007. Virtual pads: Decoupling motor space and visual space for flexible manipulation of 2d windows within ves. In Proceedings of the 2007 IEEE Symposium on 3D User Interfaces. IEEE.

[2] Myroslav Bachynskyi and Jörg Müller. 2020. Dynamics of Aimed Mid-Air movements. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI’20). Association for Computing Machinery, New York, NY, 1–12. DOI: https://doi.org/10.1145/3313831.3376194

[3] Myroslav Bachynskyi, Antti Oulasvirta, Gregorio Palmas, and Tino Weinkauf. 2014. Is motion capture-based biomechanical simulation valid for HIC studies? Study and implications. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 3215–3224.

[4] Myroslav Bachynskyi, Gregorio Palmas, Antti Oulasvirta, Jürgen Steimle, and Tino Weinkauf. 2015. Performance and ergonomics of touch surfaces: A comparative study using biomechanical simulation. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI’15). ACM, New York, NY, 1817–1826. DOI: http://doi.acm.org/10.1145/2702123.2702607

[5] Myroslav Bachynskyi, Gregorio Palmas, Antti Oulasvirta, and Tino Weinkauf. 2015. Informing the design of novel input methods with muscle coactivation clustering. ACM Transactions on Computer–Human Interaction (TOCHI) 21, 6. Article 30 (Jan. 2015), 25 pages.

[6] Reinoud J. Bootsmä, Laure Fernandez, and Denis Mottet. 2004. Behind Fitts’ law: Kinematic patterns in goal-directed movements. International Journal of Human–Computer Studies 61, 6 (2004), 811–821.

[7] John Charles Butcher. 2016. Numerical Methods for Ordinary Differential Equations. John Wiley & Sons.

[8] Stuart K. Card, William K. English, and Betty J. Burr. 1978. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. Ergonomics 21, 8 (1978), 601–613.

[9] Noshaba Cheema, Laura A. Frey-Law, Kourosh Naderi, Jaakko Lehtinen, Philipp Slusallek, and Perttu Hämäläinen. 2020. Predicting Mid-Air interaction surfaces and fatigue using deep reinforcement learning. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI’20). Association for Computing Machinery, New York, NY, 1–13. DOI: https://doi.org/10.1145/3313831.3376701

[10] Scott L. Delp, Frank C. Anderson, Allison S. Arnold, Peter Loan, Ayman Habib, Chand T. John, Eran Guendelman, and Darryl G. Thelen. 2007. OpenSim: Open-source software to create and analyze dynamic simulations of movement. IEEE Transactions on Biomedical Engineering 54, 11 (2007), 1940–1950.

[11] Jörn Diedrichsen, Reza Shadmehr, and Richard B. Ivry. 2010. The coordination of movement: Optimal feedback control and beyond. Trends in Cognitive Sciences 14, 1 (2010), 31–39.

[12] Peter Dorato and Alexander Levis. 1971. Optimal linear regulators: The discrete-time case. IEEE Transactions on Automatic Control 16, 6 (1971), 613–620.

[13] A. Aldo Faisal, Luc P. J. Selen, and Daniel M. Wolpert. 2008. Noise in the nervous system. Nature Reviews Neuroscience 9, 4 (2008), 292–303.

[14] Faulwasser Timm, Grüne Lars, and Müller Matthias A. 2018. Economic Nonlinear Model Predictive Control. https://ieeexplore.ieee.org/document/8277242.

[15] Timm Faulwasser, Milan Korda, Colin N. Jones, and Dominique Bonvin. 2014. Turnpike and dissipativity properties in dynamic real-time optimization and economic MPC. In Proceedings of the 33rd IEEE Conference on Decision and Control. IEEE, 2734–2739.

[16] Florian Fischer, Miroslav Bachinski, Markus Klar, Arthur Fleig, and Jörg Müller. 2021. Reinforcement learning control of a biomechanical model of the upper extremity. Scientific Reports 11, 1 (2021), 1–15.

[17] Florian Fischer, Arthur Fleig, Markus Klar, Lars Grüne, and Jörg Müller. 2020. An optimal control model of mouse pointing using the LQR. https://arxiv.org/abs/2002.11596.

[18] Florian Fischer, Arthur Fleig, Markus Klar, and Jörg Müller. 2022. Optimal feedback control for modeling human–computer interaction. ACM Transactions on Computer–Human Interaction 29, 6, Article 51 (November 2022), 70 pages. DOI: https://doi.org/10.1145/3524122

[19] Paul M. Fitts. 1954. The information capacity of the human motor system in controlling the amplitude of movement. Journal of Experimental Psychology 47, 6 (1954), 381.

[20] Tamar Flash and Neville Hogan. 1985. The coordination of arm movements: An experimentally confirmed mathematical model. Journal of Neuroscience 5, 7 (1985), 1688–1703.

[21] Peter Gawthrop, Ian Loram, Martin Lakie, and Henrik Gollée. 2011. Intermittent control: A computational theory of human control. Biological Cybernetics 104, 1-2 (2011), 31–51.

[22] Lars Grüne and Vryan Gil Palma. 2014. On the benefit of re-optimization in optimal control under perturbations. In Proceedings of the 21st International Symposium on Mathematical Theory of Networks and Systems MTNS 2014. University of Groningen, Groningen, 439–446. Retrieved from https://eref.uni-bayreuth.de/11269/.

[23] Lars Grüne and Jürgen Pannek. 2017. Nonlinear Model Predictive Control. Springer International Publishing, Cham, 45–69. DOI: https://doi.org/10.1007/978-3-319-46024-6_3
Simulating Interaction Movements via Model Predictive Control

[47] Jörg Müller, Antti Oulasvirta, and Roderick Murray-Smith. 2017. Control theoretic models of pointing. ACM Transactions on Computer–Human Interaction (TOCHI) 24, 4 (2017), 1–36.

[48] Eri Nakano, Hiroshi Imamizu, Rieko Osu, Yoji Uno, Hiroaki Gomi, Toshinori Yoshioka, and Mitsuo Kawato. 1999. Quantitative examinations of internal representations for arm trajectory planning: Minimum commanded torque change model. Journal of Neurophysiology 81, 5 (1999), 2140–2155.

[49] Xue Bin Feng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Transactions On Graphics (TOG) 37, 4, Article 143 (July 2018), 14 pages. DOI: https://doi.org/10.1145/3197517.3201311

[50] Ivan Poupyrev, Mark Billinghurst, Suzanne Weyhorst, and Tadao Ichikawa. 1996. The go-go interaction technique: Non-linear mapping for direct manipulation in VR. In Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology. 79–80.

[51] Ning Qian, Yu Jiang, Zhong-Ping Jiang, and Pietro Mazzoni. 2013. Movement duration, Fitts’s law, and an infinite-horizon optimal feedback control model for biological motor systems. Neural Computation 25, 3 (2013), 697–724.

[52] S. Joe Qin and Thomas A. Badgwell. 2003. A survey of industrial model predictive control technology. Control Engineering Practice 11, 7 (2003), 733–764.

[53] Philip Quinn and Shumin Zhai. 2018. Modeling gesture-typing movements. Human–Computer Interaction 33, 3 (2018), 234–280. DOI: https://doi.org/10.1080/10737137.2016.1215922

[54] John Rasmussen, Michael Damsgaard, Egidijus Surma, Soren T. Christensen, Mark de Zee, and Vit Vondrak. 2003. Anybody—a software system for ergonomic optimization. In Proceedings of the Fifth World Congress on Structural and Multidisciplinary Optimization, Vol. 4. Citeseer, 6.

[55] James B. Rawlings, David Q. Mayne, and Moritz M. Diehl. 2017. Model Predictive Control: Theory, Computation, and Design (2nd ed.). Nob Hill Publishing.

[56] Katherine R. Saul, Xiao Hu, Craig M. Goehler, Meghan E. Vidt, Melissa Daly, Anca Velisar, and Wendy M. Murray. 2014. Benchmarking of dynamic simulation predictions in two software platforms using an upper limb musculoskeletal model. Computer Methods in Biomechanics and Biomedical Engineering 5842, May 2016 (2014), 1–14. DOI: https://doi.org/10.1080/10255842.2014.916698

[57] Richard A. Schmidt, Howard Zelaznik, Brian Hawkins, James S. Frank, and John T. Quinn Jr. 1979. Motor-output variability: A theory for the accuracy of rapid motor acts. Psychological Review 86, 5 (1979), 415.

[58] Ajay Seth, Jennifer L. Hicks, Thomas K. Uchida, Ayman Habib, Christopher L. Dembia, James J. Dunne, Carmichael F. Ong, Matthew S. DeMers, Apoorva Rajagopal, Matthew Millard, et al. 2018. OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. PLoS Computational Biology 14, 7 (2018), e1006223.

[59] Olga Sorkine-Hornung and Michael Rabinovich. 2017. Least-squares rigid motion using svd. Computing 1, 1 (2017), 1–5.

[60] G. G. Sutton and K. Sykes. 1967. The variation of hand tremor with force in healthy subjects. The Journal of Physiology 191, 3 (1967), 699–711.

[61] Misaki Takeda, Takanori Sato, Hisashi Saito, Hiroshi Iwasaki, Isao Nambu, and Yasuhiro Wada. 2019. Explanation of fitts’ law in reaching movement based on human arm dynamics. Scientific Reports 9, 1 (2019), 1–12. DOI: https://doi.org/10.1038/s41598-019-56016-7

[62] Darryl G. Thelen and Frank C. Anderson. 2006. Using computed muscle control to generate forward dynamic simulations of human walking from experimental data. Journal of Biomechanics 39, 6 (2006), 1107–1115. DOI: https://doi.org/10.1016/j.jbiomech.2005.02.010

[63] Emanuel Todorov. 2005. Stochastic optimal control and estimation methods adapted to the noise characteristics of the sensorimotor system. Neural Computation 17, 5 (2005), 1084–1108.

[64] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 5026–5033.

[65] Emanuel Todorov and Michael I. Jordan. 2002. Optimal feedback control as a theory of motor coordination. Nature Neuroscience 5, 11 (2002), 1226–1235. DOI: https://doi.org/10.1038/nn963

[66] Brian R. Umberger and Ross H. Miller. 2017. Optimal control modeling of human movement. Handbook of Human Motion, Müller Bertram, Wolf Sebastian L., Brueggemann Gert-Peter, Deng Zhigang, McIntosh Andrew, Miller Freeman, and Selbie William Scott (Eds.). Springer International Publishing, Cham, 1–22. https://doi.org/10.1007/978-3-319-30808-1_177-1

[67] Yoji Uno, Mitsuo Kawato, and Rika Suzuki. 1989. Formation and control of optimal trajectory in human multijoint arm movement. Biological Cybernetics 61, 2 (1989), 89–101.

[68] Robert J. van Beers, Patrick Haggard, and Daniel M. Wolpert. 2004. The role of execution noise in movement variability. Journal of Neurophysiology 91, 2 (2004), 1050–1063. DOI: https://doi.org/10.1152/jn.00652.2003
[69] Frans C. T. Van der Helm and Leonard A. Rozendaal. 2000. Musculoskeletal systems with intrinsic and proprioceptive feedback. In Proceedings of the Biomechanics and Neural Control of Posture and Movement. Springer, 164–174.

[70] Sergio Vazquez, Jose Rodriguez, Marco Rivera, Leopoldo G. Franquelo, and Margarita Norambuena. 2017. Model predictive control for power converters and drives. Advances and trends. IEEE Transactions on Industrial Electronics 64, 2 (2017), 935–947. DOI: https://doi.org/10.1109/TIE.2016.2625238

[71] Yasuhiro Wada, Yuichi Kaneko, Eri Nakano, Rieko Osu, and Mitsuo Kawato. 2001. Quantitative examinations for multi joint arm trajectory planning–using a robust calculation algorithm of the minimum commanded torque change trajectory. Neural Networks 14, 4-5 (2001), 381–393.

[72] Eric A. Wan and Rudolph Van Der Merwe. 2000. The unscented Kalman filter for nonlinear estimation. In Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373). 153–158.

[73] Jiaole Wang and Masazumi Katayama. 2011. Optimal model for selecting human arm posture during reaching movement. In Proceedings of the Advances in Cognitive Neurodynamics (II). Springer, 453–458.

[74] Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning symmetric and low-energy locomotion. ACM Transactions on Graphics (TOG) 37, 4 (2018), 1–12.

[75] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Transactions on Mathematical Software (TOMS) 23, 4 (December 1997), 550–560. DOI: https://doi.org/10.1145/279232.279236

Received 19 April 2022; revised 11 November 2022; accepted 21 November 2022