Real-Time Inverse Kinematics for Generating Multi-Constrained Movements of Virtual Human Characters

Hendric Voss
Social Cognitive Systems Group
Bielefeld University
Bielefeld, Germany
hvoss@techfak.uni-bielefeld.de

Stefan Kopp
Social Cognitive Systems Group
Bielefeld University
Bielefeld, Germany
skopp@techfak.uni-bielefeld.de

Abstract

Generating accurate and realistic virtual human movements in real-time is of high importance for a variety of applications in computer graphics, interactive virtual environments, robotics, and biomechanics. This paper introduces a novel real-time inverse kinematics (IK) solver specifically designed for realistic human-like movement generation. Leveraging the automatic differentiation and just-in-time compilation of TensorFlow, the proposed solver efficiently handles complex articulated human skeletons with high degrees of freedom. By treating forward and inverse kinematics as differentiable operations, our method effectively addresses common challenges such as error accumulation and complicated joint limits in multi-constrained problems, which are critical for realistic human motion modeling. We demonstrate the solver's effectiveness on the SMPLX human skeleton model, evaluating its performance against widely used iterative-based IK algorithms, like Cyclic Coordinate Descent (CCD), FABRIK, and the nonlinear optimization algorithm IPOPT. Our experiments cover both simple end-effector tasks and sophisticated, multi-constrained problems with realistic joint limits. Results indicate that our IK solver achieves real-time performance, exhibiting rapid convergence, minimal computational overhead per iteration, and improved success rates compared to existing methods. The project code is available at https://github.com/hvoss-techfak/TF-JAX-IK

CCS Concepts

• **Human-centered computing** → **Interactive systems and tools**; **Interaction devices**; • **Computing methodologies** → **Machine learning**; **Real-time simulation**.

Keywords

machine learning; deep learning; co-speech gesture generation; gesture synthesis; multimodal data; transformer; behavior cloning; reinforcement learning

1 Introduction

Modern robotics, computer graphics, and biomechanics often face the challenge of moving complex articulated structures, ranging from robotic arms to digital human characters, to achieve desired poses or movements [17, 38]. Accurate and efficient motion generation requires solving the inverse kinematics (IK) problem: determining a set of joint configurations that achieve a desired position or orientation of an end-effector [33]. In many applications, from animating lifelike characters in virtual environments to controlling multi-joint robotic manipulators, IK provides the essential computational framework for translating high-level motion goals into precise joint movements [33].

Solving the IK problem is particularly acute in the generation of real-time, realistic movements for virtual human characters, where systems must simultaneously address speed, accuracy, and the physical and anatomical plausibility of human-like motion [16]. This is inherently challenging due to the complexity of the human skeleton, with its intricate assembly of joints and highly non-linear and interdependent degrees of freedom [27]. In the field of humanoid robotics and virtual agents, this challenge is compounded, as systems must not only mimic the natural motion of the human body but also operate in real-time to enable seamless interaction and engagement [13, 31, 42].

Traditional IK approaches, such as Jacobian-based iterative methods [14], often struggle with local minima or require significant simplifications that compromise the naturalness of motion [39]. Furthermore, in a high degree of freedom system, even small numerical errors can propagate along the kinematic chain, leading to large discrepancies at the end effectors [3, 12, 14]. This is especially critical when modeling human-like motion under multi-constraint conditions, where arbitrary and dynamic constraints, such as joint limits, collision avoidance, contact interactions, and expressive behavior, must be accommodated simultaneously [43], sometimes even in dynamic environments and real-time interactive applications involving human characters. These challenges highlight the need for IK solvers that can efficiently handle complex kinematic chains with multiple constraints while maintaining both accuracy and flexibility.

In this paper, we present a TensorFlow-based IK solver designed for the real-time generation of multi-constrained movements in virtual human characters. Emphasizing simplicity, speed, and accuracy, our approach leverages the automatic differentiation and

just-in-time (JIT) compilation features of TensorFlow [1] to formulate both forward and inverse kinematics as fully differentiable functions. This design not only accelerates the computation for complex kinematic chains but also allows the creation of arbitrarily complex objective functions that can be used to model any number of different end-effector constraints. Our solver is particularly well suited for a high degree of freedom human body models, where minimizing the error between target positions and computed joint configurations is challenging due to complex rotational dynamics and strict boundary conditions along the kinematic tree.

2 Related Work

As one of the earliest approaches to inverse kinematics, analytical methods solve a set of closed-form equations to position an end effector at a specified target. While effective, these methods have historically relied on strong assumptions about a robot's specific structure [39]. More recently, analytical solutions have emerged that are independent of predefined structural constraints. Tian et al. [46] proposed a solution for 7-DOF anthropomorphic manipulators by decomposing the problem into eight groups of arm configuration manifolds.

In contrast, numerical methods use iterative algorithms that converge to a solution based on an initial estimate. A fundamental numerical approach is the Moore-Penrose pseudo-inverse method, introduced by Barata and Hussein [4], which determines joint configurations by minimizing the least-squares error between the current and desired end-effector positions. To improve stability, especially near singular configurations, Buss and Kim [11] introduced the Selectively Damped Least Squares (SDLS) method, which dynamically adjusts the damping factor to mitigate instability.

Another widely used numerical technique is Cyclic Coordinate Descent (CCD) [12], which is particularly useful in applications that require computational efficiency, such as real-time animation and robotics. CCD iteratively adjusts each joint in a kinematic chain, working backward from the end effector to the base to progressively minimize positional error.

Forward And Backward Reaching Inverse Kinematics (FABRIK) [3] refines joint positions by successively adjusting them in forward and backward passes along the kinematic chain. Unlike CCD, which rotates joints to minimize end-effector error, FABRIK directly manipulates joint positions while maintaining link constraints, resulting in smoother and more stable solutions. This makes it ideal for character animation and humanoid robotics applications where natural motion is essential.

Similarly, Interior Point OPTimizer (IPOPT) [47, 48] is a primal-dual interior-point solver for large-scale nonlinear programming. It tackles a sequence of logarithmic-barrier sub-problems and uses a filter line-search to separately enforce decreases in constraint violation or objective value, which guarantees global convergence without resorting to merit functions. Each Newton step is complemented by second-order corrections, inertia-controlled linear solves and a dedicated feasibility-restoration phase, giving the method robustness even from poor initial guesses. This approach makes IPOPT a preferred choice for trajectory optimisation, process engineering and other engineering optimisation tasks where both scalability and numerical reliability are critical.

Further advances in real-time motion synthesis were made by Rakita et al. [41], who developed a method that ensures smooth, feasible manipulator motions while avoiding joint space discontinuities by framing inverse kinematics as a weighted sum nonlinear optimization problem. Similarly, Zhang et al. [50] proposed an approach based on constrained linear programming that incorporates joint bounds and velocity and acceleration bounds as inequality constraints to refine kinematic solutions. In addition, nonlinear solvers such as Sequential Quadratic Programming (SQP) [9] and Interior-Point Methods (IPM) [34] have been used to efficiently solve multi-constraint inverse kinematic problems.

2.1 Machine Learning Inverse Kinematic

Another important development in numerical inverse kinematics is the use of machine learning approaches to learn optimization-based techniques. Current implementations of inverse kinematics in the machine learning domain focus mainly on training neural networks to learn the IK problem from training data, and using these trained models during the inference phase to solve the IK problem for new, unseen end-effector targets [7]. As one of the earliest approaches, Köker et al. [21] trained an artificial neural network (ANN) on five thousand trajectories and inferred cubic trajectories directly from the neural network output.

Building on this foundation, more recent advances have used deep learning architectures to improve the generalization and efficiency of inverse kinematics (IK) solutions. Levine et al. [22] demonstrates the potential of deep reinforcement learning (RL) to solve IK problems in robotic manipulation tasks without requiring explicit kinematic modeling. By training policies to maximize task performance, RL-based IK methods can adapt to different environments and constraints.

A notable extension of this paradigm is the MAPPO IK algorithm proposed by Zhao et al. [51], which uses multi-agent proximal policy optimization to simultaneously optimize position and posture control in six-degree-of-freedom manipulators. Unlike previous RL implementations that focus primarily on positional accuracy, this approach introduces a dual reward mechanism that combines Gaussian distance for positional accuracy and cosine distance for orientation alignment.

The challenge of solution multiplicity in redundant manipulators has inspired alternative machine learning frameworks. Bocsi et al. [8] pioneered structured output learning using joint kernel support vector machines to model the probability distribution of end-effector positions and joint angles. By embedding sine/cosine joint angle transformations in kernel space, their method achieved 1.4 mm tracking accuracy on a 7-DOF Barrett WAM while maintaining temporal consistency across ambiguous configurations.

Hybrid analytic-statistical approaches have emerged to bridge the gap between model-based and data-driven methods. Nguyen and Marvel [35] developed a Gaussian process framework that uses closed-form kinematic equations as prior distributions updated by Bayesian inference with optical motion capture data, significantly reducing calibration points compared to pure neural network approaches.

Recent innovations in neural architectures address inverse kinematics through hierarchical probability modeling. The IKNet framework by Bensadoun et al. [6] uses variational autoencoder principles with Gaussian mixture models to sequentially sample joint configurations along the kinematic chain. By conditioning the probability distribution of each joint on previous angles and end-effector targets, the method generates multiple physically plausible solutions with high success rates.

Although powerful, Eapen et al. [15] showed that Gaussian regression and support vector machine algorithms often outperform neural network approaches for IK solutions.

3 Architecture and Method

In this section, we describe the architecture and methodology of our IK solver. Our system consists of two main components: a pure, JIT-compiled forward kinematics (FK) module and an inverse kinematics solver that minimizes an objective function formulated from the IK error and additional biomechanical penalties. We first outline the forward kinematics formulation, then describe the inverse kinematics optimization framework, discuss the integration of TensorFlow for automatic differentiation and gradient descent, and finally explain the various objective functions used to enforce kinematic and task-specific constraints.

3.1 Forward Kinematics

The FK module forms the basis of the solver by computing global transformations for a given human skeleton. A skeleton is loaded and represented as a tree structure with nodes corresponding to bones. A local transformation matrix is defined for each bone, and the FK computation uses these local transformations along with parent-child relationships to compute the global pose.

To update controlled bones, the forward kinematics (FK) model computes the global transformation of each joint in the kinematic tree from the local joint transformations. For a joint i with parent index $p(i)$, its transformation T_i is recursively computed as

$$T_i = \begin{cases} L_i R_i, & \text{if } p(i) < 0 \\ T_{p(i)} L_i R_i, & \text{otherwise} \end{cases} \tag{1}$$

where $L_i \in \mathbb{R}^{4 \times 4}$ is the fixed local transformation (encoding the length and rest pose of the bone), and $R_i \in \mathbb{R}^{4 \times 4}$ is the rotation matrix associated with the joint i.

The rotation matrix R_i is derived from the Euler angles $\theta = [\theta_x, \theta_y, \theta_z]^\top$. Specifically, we compute the rotation matrices about the x-, y-, and z-axes as follows:

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

$$R_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

$$R_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

The combined rotation is then given by:

$$R_i = R_z(\theta_z) \, R_y(\theta_y) \, R_x(\theta_x). \tag{5}$$

The FK is computed over the entire chain, resulting in a set of global transformations $\{T_i\}$ that are used for further IK computations. This differentiable and modular FK pipeline ensures that gradients can be propagated back through the kinematic chain, a key requirement for the subsequent IK optimization.

3.2 Inverse Kinematics

The inverse kinematics (IK) problem seeks the optimal set of joint angles θ that minimizes a given error function $J(\theta)$. An iterative gradient-descent-based optimization routine refines the angle vector over several iterations. At each iteration, the gradient of the objective function is computed with respect to the joint angles, and a learning rate is applied to update the solution while respecting the lower and upper bounds of the joint angles.

In our formulation, the objective function is constructed as:

$$J(\theta) = J_{\text{main}}(\theta) + \sum_k J_k(\theta), \tag{6}$$

where J_{main} is the primary objective (typically enforcing the distance between the end-effector and the target) and the additional terms $\{J_k\}$ incorporate further constraints such as orientation alignment or collision avoidance.

3.3 TensorFlow Integration and Gradient Descent

Our implementation uses the TensorFlow framework to provide efficient automatic differentiation and just-in-time (JIT) compilation [1]. By expressing all kinematic computations as TensorFlow functions, we can compute gradients of the objective function $J(\theta)$ with respect to the joint angles θ. In machine learning tasks, the gradient descent algorithm updates the joint angles by computing the gradient of the objective function and then applying the update rule:

$$\theta_{t+1} = \theta_t - \eta \, \nabla J(\theta_t), \tag{7}$$

where η is the learning rate.

To improve convergence and stability, we use the Adam gradient descent algorithm [20]. Adam refines the gradient descent update rule by maintaining exponential moving averages of the gradients

and their squares. At each iteration t, these moment estimates are computed as

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla J(\boldsymbol{\theta}_t), \tag{8}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)\left(\nabla J(\boldsymbol{\theta}_t)\right)^2, \tag{9}$$

with bias-corrected estimates given by:

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}, \tag{10}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}. \tag{11}$$

The joint angles are then updated according to:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon}, \tag{12}$$

where ϵ is a small constant for numerical stability.

In addition to the standard Adam update, we incorporate a cautious modification to suppress unstable updates. Inspired by recent work on cautious optimizers [23], we compute a binary mask M that retains only those coordinates where the bias-corrected momentum $\hat{m}_t$ and the gradient $\nabla J(\boldsymbol{\theta}_t)$ share the same sign, i.e.,

$$M = \frac{\mathbb{I}\left(\hat{m}_t \circ \nabla J(\boldsymbol{\theta}_t) > 0\right)}{\max\left(\text{mean}\left(\mathbb{I}\left(\hat{m}_t \circ \nabla J(\boldsymbol{\theta}_t) > 0\right)\right), \epsilon\right)}, \tag{13}$$

where $\circ$ denotes the element-wise product and ϵ is a small constant for numerical stability. This mask is then applied to weight the momentum update, zeroing out contributions with inconsistent sign information. A compensatory scaling factor,

$$\alpha(x) = \frac{\dim(x)}{\text{nnz}(x > 0) + 1}, \tag{14}$$

ensures that the overall update magnitude is preserved. Consequently, the cautious update

$$\tilde{m}_t = \alpha(\hat{m}_t)\left(\hat{m}_t \circ M\right) \tag{15}$$

is used in place of $\hat{m}_t$ in the parameter update rule, leading to enhanced convergence and stability during training [23].

3.4 Objective Functions

The design of the objective functions is critical to ensure both accuracy and natural motion. We implement several objective functions in our framework:

Distance Objective. This term penalizes the Euclidean distance between the computed end-effector position and the target:

$$J_{\text{distance}}(\boldsymbol{\theta}) = \|\mathbf{t} - T_{\text{effector}}(\boldsymbol{\theta})\|_2. \tag{16}$$

Look-At Objective. To enforce a desired orientation, such as aligning a bone with a target direction, we define a look-at penalty:

$$J_{\text{look}}(\boldsymbol{\theta}) = \left(\arccos\frac{\langle\mathbf{d}_b(\boldsymbol{\theta}), \mathbf{d}_t\rangle}{\|\mathbf{d}_b(\boldsymbol{\theta})\|\|\mathbf{d}_t\|}\right)^2, \tag{17}$$

where $\mathbf{d}_b(\boldsymbol{\theta})$ is the direction of the bone computed from the FK, and $\mathbf{d}_t$ is the target direction. The formulation involves modifications to the target point, enabling fine-tuning of the look-at behavior.

Known Rotation Objective. For scenarios where a particular joint rotation is desired, we introduce a penalty that measures the deviation from a candidate rotation $\boldsymbol{\theta}^*$:

$$J_{\text{known}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left\|\theta_i - \theta_i^*\right\|_2^2, \tag{18}$$

with N being the number of controlled degrees of freedom. A binary mask can be applied to focus the penalty on specific joints.

Objective Combination. The overall IK optimization can performed by composing any of the above objectives. For example, for our evaluation, we combined all equations:

$$\begin{aligned} J(\boldsymbol{\theta}) = \lambda_{\text{dist}}J_{\text{distance}}(\boldsymbol{\theta}) + \lambda_{\text{look}}J_{\text{look}}(\boldsymbol{\theta}) \\ + \lambda_{\text{known}}J_{\text{known}}(\boldsymbol{\theta}). \end{aligned} \tag{19}$$

where $\lambda_{\text{dist}}, \lambda_{\text{look}}, \lambda_{\text{known}}$ are weights that balance the contribution of each objective.

The optimization process leverages the forward kinematics module to compute the global transformations, and the gradient descent module iteratively updates $\boldsymbol{\theta}$ to minimize $J(\boldsymbol{\theta})$ subject to bound constraints:

$$\boldsymbol{\theta} \in [\boldsymbol{\theta}_{\min}, \boldsymbol{\theta}_{\max}]. \tag{20}$$

3.5 Extendability

This modular design allows for the easy integration of additional objectives and constraints, making the solver highly adaptable for various applications in the field of robotics and virtual agents. The following pseudo-code shows, on an example of the distance objective described in section 3.4, the ease of implementing new objective functions for the IK system. Through the use of JIT compiling and static arguments, arbitrarily complex functions can be created.

Algorithm 1 Distance Objective Function

Require: fksolver, target_point, bone_name
1: **function** DISTANCE_OBJ(fksolver, target_point, bone_name)
2:     fk_bone ← fksolver(bone_name)
3:     **return** ‖fk_bone − target_point‖
4: **end function**

3.6 Stopping Criteria

The optimization loop in our IK solver incorporates three key stopping criteria to ensure both convergence and real-time performance. Let t denote the current iteration, $N_{\max}$ the user-defined maximum number of iterations, and $J(\boldsymbol{\theta}_t)$ the objective function value at iteration t.

Iteration Count Criterion. The first criterion ensures that the solver does not run indefinitely by terminating the optimization when the number of iterations exceeds a predefined maximum:

$$t \geq N_{\max}. \tag{21}$$

Error Threshold Criterion. The second criterion halts the optimization once the objective function (i.e., the error) falls below a specified threshold ϵ:

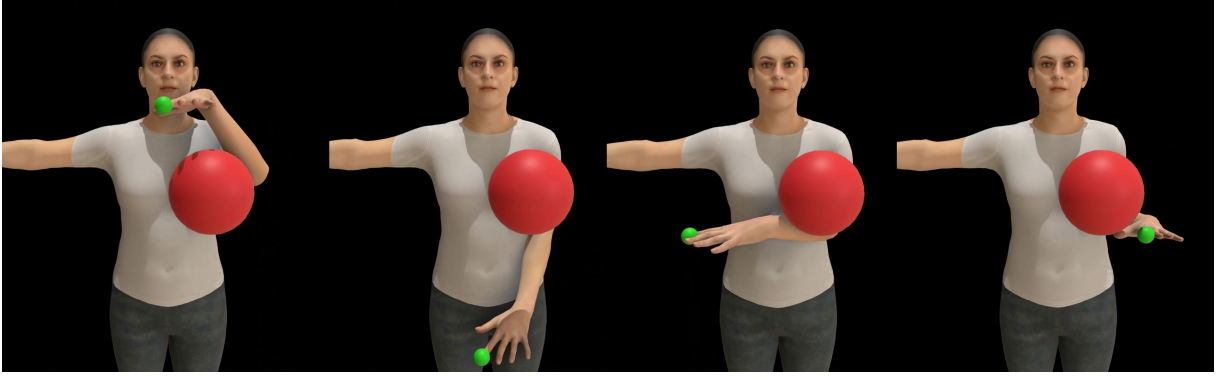$$J(\boldsymbol{\theta}_t) < \epsilon. \tag{22}$$

Figure 1: Visualization of multiple combined objectives. The green ball is the target position, while the red ball has to be avoided. We combine the *Distance Objective,* a hand palm down *Look-At Objective,* an optional hand forward *Look-At Objective* and the motion planning *Smoothness Objective.* Full demo videos are available at https://reviewanonio.github.io/TF-IK.github.io/

Dynamic Iteration Adjustment. In addition to the fixed iteration and error criteria, the solver dynamically adjusts the maximum allowed iterations based on the average computation time per iteration. Let $T_{\max}$ represent the maximum allowed total computation time and $\bar{t}$ the average time per iteration:

$$\bar{t} = \frac{1}{t} \sum_{i=1}^{t} t_i, \qquad (23)$$

where t_i is the computation time for the i-th iteration. The maximum allowed iterations, N_{allowed}, are then determined as:

$$N_{\text{allowed}} = \min\left\{ N_{\max}, \left\lfloor \frac{T_{\max}}{\bar{t}} \right\rfloor \right\}. \qquad (24)$$

Collectively, these stopping criteria ensure that the optimization process terminates either when a satisfactory solution is reached (i.e. when the error is sufficiently low), the iteration limit is met, or the time constraints are exceeded, thereby balancing computational efficiency and solution accuracy.

3.7 Motion Planning

Our IK solver can be naturally extended to perform motion planning, finding smooth transitions between an initial pose and a target configuration through intermediate points. To achieve this, we introduce a trajectory consisting of the original position, a predefined number of intermediate subpoints, and the final target position. The goal is to optimize the entire trajectory simultaneously, resulting in natural and smooth motions.

To ensure smoothness throughout the planned trajectory, we generalize the smoothness objectives using finite differences. The generalized smoothness objective penalizes rapid changes in the trajectory of joint angles, formulated as:

$$J_{\text{smoothness}}^{(n)}(\boldsymbol{\theta}) = \sum_{t=1}^{T-n} \left\| \sum_{k=0}^{n} (-1)^k \binom{n}{k} \boldsymbol{\theta}_{t+n-k} \right\|_2^2, \qquad (25)$$

where n represents the order of the finite difference (velocity $n = 1$, acceleration $n = 2$, jerk $n = 3$), and T is the total number of points in the trajectory.

Implementation	Device	Avg. Time per Iteration (ms)
JAX	GPU	1.2 ± 4.0
JAX	CPU	0.5 ± 1.6
TensorFlow	GPU	3.9 ± 13.7
TensorFlow	CPU	**0.3 ± 0.8**

Table 1: Comparison of iteration times between the CPU and GPU configurations of both the TensorFlow and JAX implementation. In all fields we report the mean value with the respective standard deviation.

The optimization framework combines this generalized smoothness objective with previously defined task-specific objectives (e.g., distance, look-at, known rotations, collision avoidance) to construct the complete motion-planning objective:

$$\begin{aligned} J_{\text{trajectory}}(\boldsymbol{\theta}) = \; & \lambda_{\text{dist}} J_{\text{distance}}(\boldsymbol{\theta}) + \lambda_{\text{look}} J_{\text{look}}(\boldsymbol{\theta}) \\ & + \lambda_{\text{known}} J_{\text{known}}(\boldsymbol{\theta}) \\ & + \sum_{n=1}^{3} \lambda_n J_{\text{smoothness}}^{(n)}(\boldsymbol{\theta}), \end{aligned} \qquad (26)$$

where the weights $\lambda_{\text{dist}}, \lambda_{\text{look}}, \lambda_{\text{known}}, \lambda_n$ balance the influence of each term in the overall optimization.

The solver initializes a trajectory vector $\boldsymbol{\theta}$ containing the joint angles for all points (initial, intermediate, and target). The Adam optimizer iteratively refines this trajectory by minimizing $J_{\text{trajectory}}(\boldsymbol{\theta})$ while adhering to joint-angle constraints:

$$\boldsymbol{\theta}_t \in [\boldsymbol{\theta}_{\min}, \boldsymbol{\theta}_{\max}], \quad \text{for all trajectory points } t. \qquad (27)$$

This approach yields a smooth and natural motion sequence, with a variable amount of intermediate points, from the initial configuration to the final target, while adhering to all given objective functions. 1 shows a combination of multiple objectives together with the smoothness objective.

4 Experimental Results

To evaluate the performance of our proposed TensorFlow-based inverse kinematics solver, we performed objective comparisons

Algorithm	Custom Objective	Solving Time (ms)	Iterations	Time per Iteration (ms)	Success Rate (%)
CCD	No	25.44 ± 62.61	53.57 ± 139.19	0.69 ± 0.27	91.19 ± 28.34
FABRIK	No	85.28 ± 26.92	453.38 ± 144.18	0.23 ± 0.16	96.12 ± 29.47
IPOPT	No	164.73 ± 132.25	**13.24 ± 3.30**	12.41 ± 8.97	99.80 ± 4.47
JAX	No	12.62 ± 5.79	37.27 ± 17.76	0.34 ± 0.01	**100.00 ± 0.00**
TensorFlow	No	**8.06 ± 3.38**	36.99 ± 17.52	**0.22 ± 0.02**	**100.00 ± 0.00**
CCD	Yes	570.49 ± 54.95	496.11 ± 43.27	1.15 ± 0.08	0.80 ± 8.91
FABRIK	Yes	439.90 ± 40.93	496.08 ± 43.69	0.89 ± 0.05	0.80 ± 8.91
IPOPT	Yes	2759.28 ± 8045.75	**44.52 ± 88.44**	45.31 ± 17.13	96.50 ± 18.39
JAX	Yes	86.18 ± 58.17	67.09 ± 45.98	1.29 ± 0.04	**99.60 ± 6.32**
TensorFlow	Yes	**43.55 ± 30.03**	67.40 ± 46.00	**0.65 ± 0.10**	**99.60 ± 6.32**

Table 2: Performance metrics for all four testes implementations both with the simple end effector target and the complex objective $J(\theta)$. In all fields we report the mean value with the respective standard deviation. For solving time, iterations, and time per iteration, lower is better. For success rate, higher is better.

against several established methods: Cyclic Coordinate Descent (CCD) [12], FABRIK [3], Interior Point OPTimizer (IPOPT) [47, 48], and an implementation of our library in JAX with just-in-time (JIT) compilation. This JAX implementation is exactly the same as the TensorFlow version but replaces all TensorFlow calls with JAX calls. There are no other implementation differences and the resulting value outputs (apart from slight rounding differences) are exactly the same, given the same input seed. We also implemented our library in PyTorch using TorchScript. However, despite our best efforts, the code performed excessive recompilations during each iteration resulting in execution times of 10-20 seconds per target. For clarity, we therefore excluded the PyTorch implementation from further analysis.

Our experiments were conducted using the SMPLX skeleton [37], a widely adopted standard for human body armatures in applications that require modeling complex joint behavior. In particular, we focused on the upper extremities, testing the bones *<hand>_collar*, *<hand>_shoulder*, *<hand>_elbow* and *<hand>_wrist* for both the left and right hand. This subset was chosen because of the challenging rotational dynamics at the wrist and the strict boundary conditions at the elbow.

Each method was evaluated using two types of IK targets. The first was a simple target end-effector position where the goal was to reach a certain target position with the index finger of the respective hand, while the second was a combination of the previously described objective functions $J(\theta)$ designed to encapsulate realistic human bone boundary constraints. In all experiments, The weight of λ_{dist} and λ_{known} was set to 1.0, while the weight of λ_{look} was set to 0.1.

For consistency, each implementation was allowed a maximum of 500 iterations per run. Each run was concluded and reported as a success if the solving algorithm reached a total loss threshold of 0.005 or lower. If the implementation did not reach the threshold inside the 500 iterations, the run was considered a failure.

The experimental dataset was generated by first sampling one million target configurations around the SMPLX skeleton. We then filtered these to retain only those targets that could be successfully solved by any of the tested implementations, resulting in 24,531 valid configurations. To ensure a representative and evenly distributed set, we performed k-means clustering on the solved targets

and selected 1010 centroids. Furthermore, to mitigate any initialization effects, measurements were taken only after the first ten targets had been processed, leaving us with 1000 sampled target locations. After each target, the SMPLX skeleton was reset to its original resting position, to not skew the iteration count between targets.

Performance metrics were computed by averaging the results over five independent runs per algorithm to account for any processing variations.

As both the JAX and TensorFlow implementation can be run with or without the GPU, we first measured the difference in iteration time for all targets with both the simple end effector and the objective function $J(\theta)$. Table 1 shows the average result over all 1000 targets. Here, both CPU configurations are faster than their GPU counterpart. Therefore, we use the CPU configuration of the JAX and TensorFlow solver for all of our performance metrics.

Table 2 summarizes the performance metrics for the tested algorithms under both objective formulations. Under the simpler IK objective (without the custom objective $J(\theta)$), the TensorFlow-based solver achieved a mean solution time of 8.06 ms with an average of 36.99 iterations and a success rate of 100%. The JAX implementation was competitive in iteration count (37.27 ± 17.76) and success rate (100%), although its solving time was slightly higher at 12.62 ms. In comparison, IPOPT had a notably lower iteration count of 13.24 ± 3.30, indicating rapid convergence in terms of iterations, but with a higher mean solving time of 164.73 ± 132.25 ms. Conversely, both CCD and FABRIK exhibited substantially lower performance under the simple end-effector objective; while CCD achieved a moderate solving time of 25.44 ms with a success rate of 91.19%, FABRIK performed noticeably worse, with a mean solving time of 85.28 ms.

When the custom objective function $J(\theta)$ was incorporated to simulate realistic human joint constraints, the performance gap between the methods became even more pronounced. Both CCD and FABRIK experienced substantial increases in computational cost, with mean solving times of 570.49 ms and 439.90 ms, respectively, and iteration counts approaching the maximum limit of 500 iterations per target; their success rates plummeted to under 1%, rendering them ineffective for applications requiring stringent adherence to biomechanical constraints. IPOPT's performance further

deteriorated under the more challenging objective, with its solving time surging to 2759.28 ± 8045.75 ms and its iteration count averaging 44.52 ± 88.44, coupled with a slightly lower success rate of 96.50%. Both the JAX and TensorFlow implementations maintained robust performance; the JAX-based solver incurred a moderate solving time of 86.18 ms with an average of 67.09 iterations and a success rate of 99.60%, while the TensorFlow-based solver outperformed the others with a mean solving time of 43.55 ms, an iteration count of 67.40, and a success rate of 99.60%.

4.1 Motion Planning Results

Trajectory Points	Solving Time (ms)	Success Rate (%)
0	8.06 ± 3.38	100.00 ± 0.00
5	17.12 ± 11.01	99.30 ± 8.34
10	21.16 ± 16.92	97.40 ± 15.92

Table 3: Distance trajectory effector performance metrics for different numbers of points in a trajectory. In all fields we report the mean value with the respective standard deviation. For solving time lower is better. For success rate higher is better.

Trajectory Points	Solving Time (ms)	Success Rate (%)
0	43.55 ± 30.03	99.60 ± 6.32
5	88.88 ± 77.82	94.99 ± 21.80
10	162.83 ± 125.96	83.67 ± 44.04

Table 4: $J_{\text{trajectory}}(\theta)$ objective performance metrics for different numbers of points in a trajectory. In all fields we report the mean value with the respective standard deviation. For solving time lower is better. For success rate higher is better.

We evaluated our motion planning extension on the TensorFlow implementation by comparing two configurations: a simple distance-based objective applied to the trajectory end effector $J_{\text{distance}}(\boldsymbol{\theta}) + \sum_{n=1}^{3} J_{\text{smoothness}}^{(n)}(\boldsymbol{\theta})$, and the full trajectory objective $J_{\text{trajectory}}(\boldsymbol{\theta})$, which incorporates additional task-specific constraints and smoothness terms. We used the same weights for all hyperparameters as in the original experiment, except for the new parameter λ_n, which we set to 0.01. For both cases, we varied the number of trajectory points by testing no intermediate trajectory points, 5 trajectory points, and 10 trajectory points to assess the impact on solution time and success rate.

Table 3 shows the performance of the distance-based objective as the number of trajectory points increases. With no intermediate points (i.e., a single-target configuration), the solver achieved an average solving time of 8.06 ms, a mean iteration count of 36.99 ± 17.52, and a success rate of 100%. When the trajectory was extended to 5 trajectory points, the solving time increased to 17.12 ms, the iteration count increased to 83.01 ± 57.81, while the success rate decreased slightly to 99.30%. With 10 points, the average solution time reached 21.16 ms, the iteration count increased to 102.65 ± 87.65, and the success rate lowered to 97.40%. Similarly, solving times under the full trajectory objective $J_{\text{trajectory}}(\theta)$ increased linearly

with the number of trajectory points (see table 4). For the baseline case with no intermediate trajectory points, the solver recorded an average solving time of 43.55 ms, an average iteration count of 67.40 ± 46.00, and a success rate of 99.60%. With 5 intermediate points, the solving time nearly doubled to 88.88 ms, the iteration count increased to 128.19 ± 113.24, and the success rate dropped to 94.99%. When the number of trajectory points was increased to 10, the mean solving time increased to 162.83 ms, the average iteration count increased to 229.62 ± 178.41, and the success rate further decreased to 83.67%. This strong decrease in success rate can be partially explained by the strong increase in iteration count, as 71 of our samples exceeded the maximum iteration count of 500. Doubling the iteration count could therefore lead to improved success rates for higher numbers of intermediate trajectory points. Regardless, these results indicate that the solver remains highly robust and works well within real-time constraints even as the problem dimensionality increases, especially if only a small amount of objective functions is used.

5 Discussion

Our experimental results demonstrate significant advantages of our TensorFlow-based inverse kinematics (IK) solver, particularly in terms of adaptability, computational speed, and reliability for complex systems with high degrees of freedom. Cyclic Coordinate Descent (CCD) methods, while efficient in simpler contexts, often encounter problems such as error accumulation and local minima when scaled up to more complex scenarios, especially under joint angle constraints [10, 18, 49]. These limitations are particularly pronounced in human motion modeling, which is characterized by intricate joint interdependencies and stringent biomechanical constraints. Similarly, FABRIK often exhibits inefficiencies and lower success rates when realistic boundary conditions and multiple concurrent constraints are enforced [2, 18, 29, 44].

IPOPT exhibited notable strengths in iteration efficiency, rapidly converging within fewer steps compared to all other implementations. This efficiency highlights IPOPT's effectiveness in navigating the optimization landscape quickly due to its interior-point method design, making it suitable for problems where the iteration cost is very high. However, IPOPT's computational overhead per iteration, presumably due to the high cost of computing Newtown's direction [25], proved to be a substantial drawback limiting its practicality in real-time applications or scenarios requiring high-frequency updates. Additionally, IPOPT demonstrated sensitivity to the complexity introduced by biomechanical constraints encoded in the custom objective function, leading to increased variability and occasional convergence issues, which impacted its overall reliability compared to methods specifically optimized for computational efficiency like our TensorFlow solver.

In computer animation and gaming, character rigs typically have many degrees of freedom and require highly responsive IK solutions to achieve believable and realistic motion [19]. Our proposed solver effectively meets these interactive performance requirements without sacrificing realism. The modular design allows seamless integration of constraints such as contact points, gaze direction, or stylistic elements without requiring retraining or modification of the underlying IK algorithms. Particularly in virtual reality and

telepresence applications, where sparse tracking data points are typically available, our solver could reliably infer complete full-body poses while respecting essential human kinematic constraints such as realistic elbow extension and knee flexion limits [36].

In humanoid robotics and virtual human simulation, it could generate complex movements on demand by optimizing joint motions to achieve goals such as balance, reaching, and obstacle avoidance. This on-demand optimization accelerates development and prototyping, as one of the major benefits of our differentiable solver is its extensibility via modular objective functions. In contrast to analytic or Jacobian pseudoinverse methods that struggle to handle multiple objectives simultaneously [28], our approach allows arbitrary constraints to be added as terms in the loss function. For instance, one can impose joint limit penalties, collision avoidance terms, or even soft biomechanical targets simply by including the corresponding differentiable cost functions in the optimization. The solver would then naturally balance all these objectives via the gradient descent process. This capability is akin to the penalty-based formulations in traditional IK solvers [5], but implemented in a powerful unified end-to-end differentiable manner within TensorFlow. Consequently, our IK solver easily accommodates diverse tasks. For instance, achieving precise reaching motions while preserving human-like postures can be realized by combining an end-effector position loss with a reference pose similarity term. Adjustments to motion objectives require no additional retraining or specialized algorithmic modifications to the objective function itself. Additionally, because our method fundamentally relies on the provided kinematic model, it generalizes seamlessly across different character rigs and robotic manipulators. A single solver implementation can thus manage multiple distinct avatars or robotic systems without requiring model-specific training, eliminating the overhead typically associated with neural network-based IK solutions [24]. Although we evaluated our IK solver on the different arm joints, this approach is fully extendable to arbitrary amounts of joints. In the future, we would like to use this approach to solve movements for the entire SMPLX skeleton and

6 Limitations

Despite its advantages, the differentiable inverse kinematics (IK) solver has trade-offs, notably higher computational cost per frame compared to analytical methods or pre-trained neural networks. Unlike direct feedforward network solutions, our iterative optimization approach requires more computational time, especially for highly redundant systems or scenarios requiring precise accuracy. To mitigate this, we use TensorFlow's XLA JIT compilation to optimize the computational graph, including Jacobian computations through automatic differentiation, and warm start each frame with the previous solution to minimize iterations. Although our method converges in milliseconds, it remains somewhat slower per frame than specialized analytical IK methods or trained networks, which can provide solutions in microseconds. Also, this approach inherently runs the risk of converging to local minima due to the nonconvex optimization landscape. However, by incorporating physically realistic constraints such as joint bounds and continuity from previous frames, we can significantly reduce the likelihood of unnatural poses or sudden deviations. In challenging

or abruptly changing goal scenarios, the use of random restarts or secondary goals can further improve robustness, although such measures were rarely required in our testing. Furthermore, our model currently relies on an accurate kinematic model for solving the inverse kinematic problem, which is not always available. This limitation could be resolved using recent work in adaptable kinematic models, but in our work, this is not yet realized [40, 45].

Another limitation is the naturalness of the produced motion, especially for virtual agents. Although we already have objection functions that minimize the changes in velocity, acceleration, and jerk, this can lead to sudden changes or movements that do not feel natural to an outside observer. In the future, we could implement different learned objective functions that expand upon this problem by, for example, introducing GAN discriminators for natural human motion Men et al. [30], learned motion classificators from large datasets [26], or even motion retrieval systems that align the solving with human input data, similar to Mughal et al. [32].

7 Conclusion

In this paper, we have presented a TensorFlow-based inverse kinematics solver that uses automatic differentiation and just-in-time compilation to solve complex, multi-constrained IK problems efficiently and accurately. By formulating both forward and inverse kinematics as differentiable functions, our approach addresses the challenges inherent in high-degree-of-freedom systems, such as error accumulation along kinematic chains and complicated joint dynamics. Our experimental evaluation has shown that the TensorFlow-based solver consistently outperforms established inverse kinematic methods. For both simple and custom IK targets, the proposed solver achieved rapid convergence with minimal iteration times and near-perfect success rates. In contrast, conventional methods such as CCD and FABRIK struggled under realistic boundary conditions, exhibiting significantly higher iteration counts and lower success rates. The performance of our JAX-based implementation, while competitive in simpler scenarios, could not match the efficiency of the TensorFlow approach under more demanding conditions.

These results underscore the practical advantages of modern differential approaches for real-time applications in robotics, computer graphics, and biomechanics. The flexibility to define arbitrary objective functions further broadens the solver's applicability in modeling complex joint rotations and enforcing realistic constraints. Future work may explore faster optimization and extension to additional kinematic models, solidifying the role of differentiable programming in advancing inverse kinematics solutions for complex articulated systems.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, San-jay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Andreas Aristidou, Yiorgos Chrysanthou, and Joan Lasenby. 2016. Extending FABRIK with model constraints. *Computer Animation and Virtual Worlds* 27 (2016), 35 – 57. https://api.semanticscholar.org/CorpusID:7367339

[3] Andreas Aristidou and Joan Lasenby. 2011. FABRIK: A fast, iterative solver for the Inverse Kinematics problem. *Graphical Models* 73, 5 (2011), 243–260.

[4] João Carlos Alves Barata and Mahir Saleh Hussein. [n. d.]. The Moore–Penrose Pseudoinverse: A Tutorial Review of the Theory. 42, 1 ([n. d.]), 146–165. doi:10.1007/s13538-011-0052-z

[5] Patrick Beeson and Barrett Ames. 2015. TRAC-IK: An open-source library for im-proved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 928–935.

[6] Raphael Bensadoun, Shir Gur, Nitsan Blau, Tom Shenkar, and Lior Wolf. [n. d.]. Neural Inverse Kinematics. doi:10.48550/arXiv.2205.10837 arXiv:2205.10837 [cs]

[7] Raphael Bensadoun, Shir Gur, Nitsan Blau, and Lior Wolf. 2022. Neural inverse kinematic. In *International Conference on Machine Learning*. PMLR, 1787–1797.

[8] Botond Bocsi, Duy Nguyen-Tuong, Lehel Csató, Bernhard Schölkopf, and Jan Peters. [n. d.]. *Learning inverse kinematics with structured prediction.* doi:10.1109/IROS.2011.6094666 Journal Abbreviation: IEEE International Conference on Intelligent Robots and Systems Pages: 703 Publication Title: IEEE International Conference on Intelligent Robots and Systems.

[9] Paul T Boggs and Jon W Tolle. 1995. Sequential quadratic programming. *Acta numerica* 4 (1995), 1–51.

[10] Ronan Boulic and Richard Kulpa. 2007. Inverse Kinematics and Kinetics for Virtual Humanoids. In *Eurographics*. https://api.semanticscholar.org/CorpusID:37999282

[11] Samuel R. Buss and Jin-Su Kim. [n. d.]. Selectively Damped Least Squares for Inverse Kinematics. 10, 3 ([n. d.]), 37–49. doi:10.1080/2151237X.2005.10129202

[12] Adrian A. Canutescu and Roland L. Dunbrack. [n. d.]. Cyclic coordinate descent: A robotics algorithm for protein loop closure. 12, 5 ([n. d.]), 963–972. doi:10.1110/ps.0242703

[13] Markos Diomataris, Nikos Athanasiou, Omid Taheri, Xi Wang, Otmar Hilliges, and Michael J. Black. [n. d.]. WANDR: Intention-guided Human Motion Genera-tion. doi:10.48550/arXiv.2404.15383 arXiv:2404.15383 [cs]

[14] Ignacy Dulęba and Michał Opałka. [n. d.]. A comparison of Jacobian-based methods of inverse kinematics for serial robot manipulators. 23, 2 ([n. d.]), 373–382. doi:10.2478/amcs-2013-0028

[15] Jittu George Eapen, Chinn Mohanan, and Harinarayanan Nampoothiri M G. 2023. Comparative Study of Machine Learning Techniques for Inverse Kinematics in a 5-DOF Manipulator. *2023 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)* (2023), 1–4. https://api.semanticscholar.org/CorpusID:266558457

[16] Jiaxi Jiang, Paul Streli, Xuejing Luo, Christoph Gebhardt, and Christian Holz. 2024. MANIKIN: biomechanically accurate neural inverse kinematics for human motion estimation. In *European Conference on Computer Vision*. Springer, 128–146.

[17] Marilyn Keller, Keenon Werling, Soyong Shin, Scott Delp, Sergi Pujades, C Karen Liu, and Michael J Black. 2023. From skin to skeleton: Towards biomechanically accurate 3d digital humans. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–12.

[18] Ben Kenwright. 2012. Inverse Kinematics - Cyclic Coordinate Descent (CCD). *J. Graph. Tools* 16 (2012), 177–217. https://api.semanticscholar.org/CorpusID:26498902

[19] Ben Kenwright. 2022. Real-time character inverse kinematics using the gauss-seidel iterative approximation method. *arXiv preprint arXiv:2211.00330* (2022).

[20] Diederik P. Kingma and Jimmy Ba. [n. d.]. Adam: A Method for Stochastic Optimization. doi:10.48550/arXiv.1412.6980 arXiv:1412.6980 [cs]

[21] Raşit Köker, Cemil Oz, Tarik Cakar, and Hüseyin Ekiz. [n. d.]. A study of neural network based inverse kinematics solution for a three-joint robot. 49 ([n. d.]), 227–234. doi:10.1016/j.robot.2004.09.010

[22] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. [n. d.]. End-to-End Training of Deep Visuomotor Policies. doi:10.48550/arXiv.1504.00702 arXiv:1504.00702 [cs]

[23] Kaizhao Liang, Lizhang Chen, Bo Liu, and Qiang Liu. 2024. Cautious optimizers: Improving training with one line of code. *arXiv preprint arXiv:2411.16085* (2024).

[24] Oliver Limoyo, Filip Marić, Matthew Giamou, Petra Alexson, Ivan Petrović, and Jonathan Kelly. 2024. Generative Graphical Inverse Kinematics. *Ieee transactions on robotics* (2024).

[25] Tianyi Lin, Shiqian Ma, Yinyu Ye, and Shuzhong Zhang. 2021. An ADMM-based interior-point method for large-scale linear programming. *Optimization Methods and Software* 36, 2-3 (2021), 389–424.

[26] Haiyang Liu, Zihao Zhu, Giorgio Becherini, Yichen Peng, Mingyang Su, You Zhou, Xuefei Zhe, Naoya Iwamoto, Bo Zheng, and Michael J Black. 2024. EMAGE: Towards unified holistic co-speech gesture generation via expressive masked audio gesture modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1144–1154.

[27] Lijia Liu, Joseph L. Cooper, and Dana H. Ballard. [n. d.]. Computational Modeling: Human Dynamic Model. 15 ([n. d.]), 723428. doi:10.3389/fnbot.2021.723428

[28] Aryslan Malik, Yevgeniy Lischuk, Troy Henderson, and Richard Prazenica. 2022. A Deep Reinforcement-Learning Approach for Inverse Kinematics Solution of a High Degree of Freedom Robotic Manipulator. *Robotics* 11, 2 (2022). doi:10.3390/robotics11020044

[29] Andres Martin, Antonio Barrientos, and Jaime Del Cerro. 2018. The natural-CCD algorithm, a novel method to solve the inverse kinematics of hyper-redundant and soft robots. *Soft robotics* 5, 3 (2018), 242–257.

[30] Qianhui Men, Howard Leung, Edmond S. L. Ho, and Hubert P. H. Shum. 2021. GAN-based Reactive Motion Synthesis with Class-aware Discriminators for Human-human Interaction. *Comput. Graph.* 102 (2021), 634–645. doi:10.1016/j.cag.2021.09.014

[31] Francisco-Javier Montecillo-Puente, Manish N Sreenivasa, and Jean-Paul Lau-mond. [n. d.]. ON REAL-TIME WHOLE-BODY HUMAN TO HUMANOID MO-TION TRANSFER. ([n. d.]).

[32] M. Hamza Mughal, Rishabh Dabral, Merel C. J. Scholman, Vera Demberg, and Christian Theobalt. 2025. Retrieving Semantics from the Deep: an RAG Solution for Gesture Synthesis. In *Computer Vision and Pattern Recognition (CVPR)*.

[33] Richard M Murray, Zexiang Li, and S Shankar Sastry. 2017. *A mathematical introduction to robotic manipulation.* CRC press.

[34] Arkadi Nemirovski and Michael Todd. [n. d.]. Interior-point methods for opti-mization. 17 ([n. d.]), 191–234. doi:10.1017/S0962492906370018

[35] Vinh Nguyen and Jeremy A. Marvel. [n. d.]. Modeling of Industrial Robot Kine-matics Using a Hybrid Analytical and Statistical Approach. 14, 5 ([n. d.]), 051009. doi:10.1115/1.4053734

[36] Mathias Parger, Joerg H Mueller, Dieter Schmalstieg, and Markus Steinberger. 2018. Human upper-body inverse kinematics for increased embodiment in consumer-grade virtual reality. In *Proceedings of the 24th ACM symposium on virtual reality software and technology*. 1–10.

[37] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. 2019. Expressive Body Capture: 3D Hands, Face, and Body from a Single Image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 10975–10985.

[38] Alexandre N Pechev. 2008. Inverse kinematics without matrix inversion. In *2008 IEEE International Conference on Robotics and Automation*. IEEE, 2005–2012.

[39] Donald L Peiper. 1968. THE KINEMATICS OF MANIPULATORS UNDER COM-PUTER CONTROL. (1968).

[40] Jose Luis Ponton, Víctor Ceballos, Lesly Acosta, Alejandro Ríos, Eva Monclús, and Nuria Pelechano. 2023. Fitted avatars: automatic skeleton adjustment for self-avatars in virtual reality. *Virtual Reality* 27, 3 (2023), 2541–2560.

[41] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. 2018. RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion.. In *Robotics: Science and Systems*, Vol. 14. Pittsburgh, PA, 26–30.

[42] Sophie Sakka, Louise Penna Poubel, and Denis Cehajic. [n. d.]. Tasks prioritization for whole-body realtime imitation of human motion by humanoid robots. In *Digital Intelligence (DI2014)* (Nantes, France, 2014-09). 5 p. https://hal.science/hal-01054887

[43] Kewei Sui, Anindita Ghosh, Inwoo Hwang, Jian Wang, and Chuan Guo. 2025. A Survey on Human Interaction Motion Generation. *arXiv preprint arXiv:2503.12763* (2025).

[44] Songqiao Tao, Huajin Tao, and Yumeng Yang. 2021. Extending FABRIK with obstacle avoidance for solving the inverse kinematics problem. *Journal of Robotics* 2021, 1 (2021), 5568702.

[45] Ravi Tejwani, Cheng qing Ma, Paolo Bonato, and H. Harry Asada. 2023. An Avatar Robot Overlaid with the 3D Human Model of a Remote Operator. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2023), 7061–7068. https://api.semanticscholar.org/CorpusID:257365693

[46] Xinyang Tian, Qinhuan Xu, and Qiang Zhan. [n. d.]. An analytical inverse kine-matics solution with joint limits avoidance of 7-DOF anthropomorphic manipula-tors without offset. 358, 2 ([n. d.]), 1252–1272. doi:10.1016/j.jfranklin.2020.11.020

[47] Andreas Wachter. 2002. *An interior point algorithm for large-scale nonlinear optimization with applications in process engineering.* Carnegie Mellon University.

[48] Andreas Wächter and Lorenz T Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106 (2006), 25–57.

[49] Zichun Xu, Yuntao Li, Xiaohang Yang, Zhiyuan Zhao, Jingdong Zhao, and Hong Liu. 2023. A combined inverse kinematics algorithm using FABRIK with opti-mization. *Journal of Intelligent & Robotic Systems* 108, 4 (2023), 62.

[50] Linlin Zhang, Huibin Du, Zhiying Qin, Yuejing Zhao, and Guang Yang. 2024. Real-time optimized inverse kinematics of redundant robots under inequality constraints. *Scientific Reports* 14, 1 (2024), 29754.
[51] Chengyi Zhao, Yimin Wei, Junfeng Xiao, Yong Sun, Dongxing Zhang, Qiuquan Guo, and Jun Yang. [n. d.]. Inverse kinematics solution and control method of 6-degree-of-freedom manipulator based on deep reinforcement learning. 14, 1 ([n. d.]), 12467. doi:10.1038/s41598-024-62948-6 Publisher: Nature Publishing Group.