3D Flexible Human Skeleton Model for Ballroom Dance Simulation

Advanced Physics-Based Approach with Foot Detail & Upper Body Hold Constraints

---

Introduction

Creating authentic ballroom dance movements requires understanding the human body as an integrated system of flexible joints and rigid segments. This guide presents a comprehensive approach to modeling a 3D human skeleton suitable for Ten Dance movements using physics-based simulation.

The key innovation is combining **flexible joint mechanics** with **ballroom-specific constraints** to capture the essential characteristics of competitive ballroom dancing: smooth rise and fall, sharp weight transfers, and the critical "frame" that defines the aesthetic quality of partner dancing.

---

1. Model Architecture Overview

1.1 Full 3D Body Segment Definition

The model extends to full 3D rotational freedom (3 degrees of freedom per joint) plus translational freedom for the root (pelvis/torso base).

Body Segment	Length (m)	Mass (kg)	DOF (Degrees of Freedom)	Notes
HEAD	0.20	4.0	3 Rotation	Neck constraints limit rotation
TORSO (rigid core)	0.50	20.0	3 Rotation	Reference frame for upper/lower body
PELVIS	0.30	8.0	6 (3 Trans + 3 Rot)	Root link, affected by

				gravity
Upper Arm L/R	0.35	2.0 each	3 Rotation	Shoulder socket (3 DOF ball joint)
Forearm L/R	0.28	1.5 each	3 Rotation	Elbow (3 DOF for wrist flexibility)
Hand L/R	0.18	0.5 each	3 Rotation	Finger/palm orientation
Upper Leg L/R	0.45	8.0 each	3 Rotation	Hip ball joint
Lower Leg L/R	0.45	5.0 each	3 Rotation	Knee (constrained to ~1 DOF)
Foot L/R (rigid)	0.28	1.5 each	3 Rotation	Ankle + foot base
Toe Ball L/R	0.06	0.3 each	3 Rotation	**Metatarsal joint (NEW - CRITICAL)**
Heel L/R	0.05	0.2 each	3 Rotation	**Heel contact point (NEW)**

Total DOF: 36 (unrestricted) → 20-24 (with realistic constraints)

1.2 Key Advantages of 3D Modeling

3D vs. 1D Comparison:

- **Full 3D Rotations**: Enables lateral sway, spinal twist, and complex partner dynamics that 1D cannot capture
- **Lateral Movement**: Critical for Quickstep, Foxtrot, and partner connection
- **Spinal Flexibility**: Different dances require different spine behaviors (Waltz = smooth, Tango = sharp twists)
- **Realistic Foot Dynamics**: Multi-segment foot structure enables authentic rise/fall mechanics
- **Professional Quality**: Matches competitive ballroom standards at Blackpool level

---

2. Advanced Foot Model (The Foundation of Ballroom Movement)

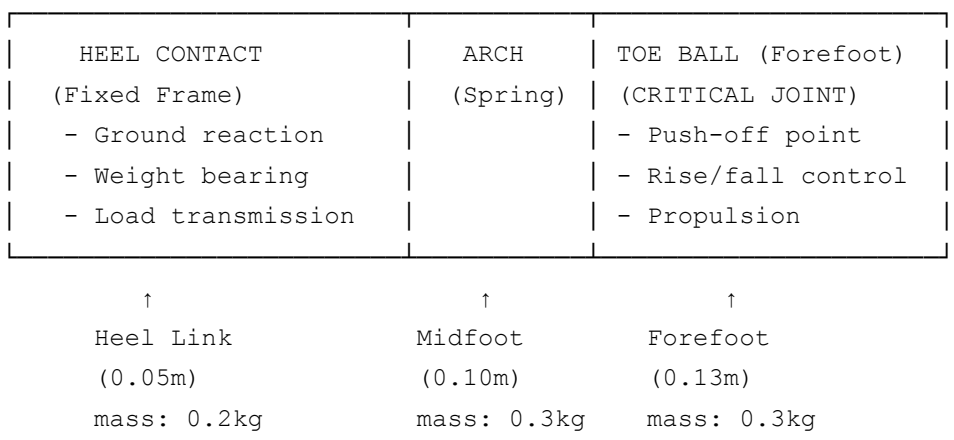
2.1 Why Foot Modeling Matters

The foot is not a rigid block—it's a sophisticated articulated system that provides:

1. **Weight bearing** (heel contact)
2. **Push-off power** (ball of foot)
3. **Balance control** (toe articulation)
4. **Aesthetic elevation** (rise and fall pattern)

2.2 Multi-Segment Foot Structure

The foot is modeled as three connected segments with individual joints:



2.3 Foot Joint Parameters

Heel Joint (Ankle-Vertical):

- Forward-back stiffness: 150 N·m/rad
- Lateral stiffness: 200 N·m/rad
- Vertical stiffness: 250 N·m/rad (highest for rise/fall control)
- Damping: 15 N·m·s/rad

Metatarsal Ball Joint (TOE BALL):

- Forward-back stiffness: 200 N·m/rad

- Lateral stiffness: 180 N·m/rad
- **Vertical stiffness: 300 N·m/rad** (HIGHEST - critical for support)
- Damping: 12 N·m·s/rad
- Full 3 DOF rotation capability

2.4 Waltz Foot Motion Pattern

Rise and Fall—The Defining Characteristic

Waltz rise and fall is the most iconic movement in ballroom dance. The pattern follows a precise timing:

Step 1-2: Rise Phase (0-0.3 seconds)

- Heel gradually rises from ground
- Weight transfers to ball of foot
- Smooth sinusoidal motion: $\theta = 0.35 \times \sin(\pi t / 0.3)$
- Peak ankle angle: ~20 degrees
- Torque control: High proportional gain (Kp = 50)

Step 3: Fall Phase (0.3-0.8 seconds)

- Controlled descent back to heel contact
- Primarily damping-driven: $\tau = -c \times d\theta/dt$
- Damping coefficient: 30 N·m·s/rad
- Smooth, controlled lowering (no sudden impact)

Quickstep Foot Motion:

- Sharp, staccato weight transfers (NOT smooth)
- Rapid heel-ball alternation
- Period: 0.5 seconds (faster tempo)
- Sharpness parameter: 0.85 (on scale 0-1)
- Control gains: Higher (Kp_sharp = 120, Kd = 8)

---

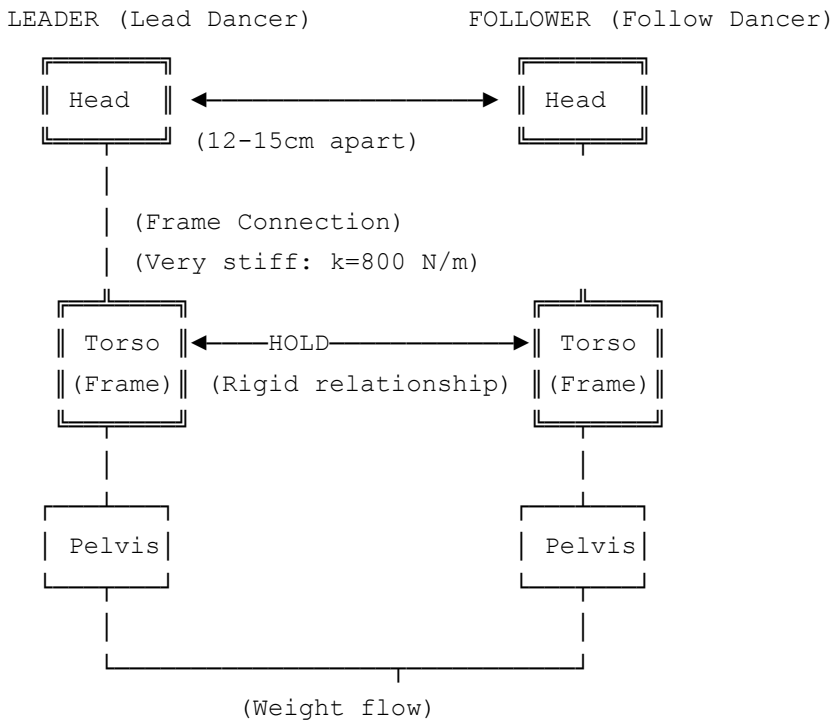
3. Upper Body Hold Constraints (Silhouette & Aesthetics)

3.1 The Ballroom Frame

In ballroom dancing, the **frame** is the foundation of aesthetic quality. It's a semi-rigid partnership structure that:

1. **Maintains distance**: 12-15cm between dancers' bodies
2. **Transmits movement**: Lead communicates through frame
3. **Creates visual harmony**: The "silhouette" that judges evaluate
4. **Enables control**: Frame rigidity allows leader to guide follower

3.2 Frame Structure Diagram



3.3 Shoulder Hold Model - Technical Details

Position Lock Constraint:

- Leader's right shoulder ↔ Follower's left shoulder
- Target separation: X=0.15m (forward-back), Y=0.10m (side-side), Z=0.05m (height)

Spring-Damper Constraint:

```
F_shoulder = -800 × (r_relative - r_target) - 40 × v_relative
```

Where:

- 800 N/m = shoulder lock stiffness (very high for rigidity)
- 40 N·s/m = damping coefficient (prevents oscillation)

Frame Rigidity by Dance Type:

Dance	Frame Stiffness	Damping	Characteristic
Waltz	700 N/m	50 N·s/m	Smooth, continuous connection
Tango	650 N/m	40 N·s/m	Sharp weight shifts, staccato
Quickstep	500 N/m	30 N·s/m	Reactive, allows follower input
Foxtrot	680 N/m	45 N·s/m	Feather-light, flowing
Viennese Waltz	720 N/m	50 N·s/m	Fast continuous

3.4 Arm Positioning (PID Control)

The arms maintain fixed beautiful lines through proportional-derivative (PID) control:

Target Arm Angles (Leader):

- Left arm (non-leading hand): 0.8 rad (~46°) - held high
- Right arm (leading hand): 0.4 rad (~23°) - slightly lower
- Elbow height: Fixed relative to torso
- Wrist angle: Small articulation allowed (±0.2 rad)

Control Law:

```
τ_arm = Kp × (θ_desired - θ_actual) + Kd × (0 - ω_actual)
```

Where:

- Kp (proportional gain) = 80 N·m/rad (maintains position)
- Kd (derivative gain) = 5 N·m·s/rad (damps oscillation)

Follower Arm Positioning:

- Mirrors leader's positions with slight delay (0.1-0.2 seconds)
- Allows for natural response to leader's frame signals
- Small oscillations permitted for artistic expression

---

4. Mathematical Foundation

4.1 Lagrangian Mechanics (OpenModelica Foundation)

Each flexible joint follows the Lagrangian equation:

$$d/dt(\partial L/\partial \dot{q}_i) - \partial L/\partial q_i = \tau_i - f\_damping - f\_constraint$$

Where:

- **L = T - V** (Kinetic energy minus Potential energy)
- **q_i** = joint angle (radians)
- **τ_i** = motor/control torque (N·m)
- **f_damping** = cω damping force
- **f_constraint** = partner coupling forces

4.2 Joint Dynamics Equation

For each joint with spring-damper behavior:

$$\tau = I \times \alpha = -k \cdot \theta - c \cdot \omega + \tau\_applied$$

Where:

- **I** = moment of inertia (kg·m²)
- **α** = angular acceleration (rad/s²)
- **k** = spring constant (N·m/rad)
- **c** = damping coefficient (N·m·s/rad)
- **θ** = angle displacement (rad)
- **ω** = angular velocity (rad/s)

4.3 Euler Integration (Numerical Solution)

For simulation, we use discrete time steps:

```
θ(t+Δt) = θ(t) + ω(t)·Δt
ω(t+Δt) = ω(t) + α(t)·Δt

where α(t) = (τ - k·θ - c·ω) / I
```

Typical parameters:

- Time step: Δt = 0.01 seconds (100 Hz simulation)
- Music tempo: 120 BPM = 2 beats/second = period 0.5 seconds per beat

---

5. Implementation: OpenModelica Approach

5.1 Basic Skeleton Structure

```
model HumanSkeleton3D
  import Modelica.Mechanics.MultiBody.*;
  import Modelica.Mechanics.MultiBody.Joints as J;
  import Modelica.Mechanics.MultiBody.Parts as P;

  // TORSO
  P.Body torso(m = 20.0,
    I = [0.5, 0, 0; 0, 1.2, 0; 0, 0, 0.6],
    r_CM = {0, 0, 0});

  // PELVIS (Root with 6 DOF)
  P.Body pelvis(m = 8.0,
    I = [0.3, 0, 0; 0, 0.4, 0; 0, 0, 0.2],
    r_CM = {0, 0, 0});

  J.FreeMotion free_joint;

  // SPINE JOINTS
  FlexibleJoint lumbar(k_forward = 150, k_lateral = 120,
    k_twist = 80, c = 15);
  FlexibleJoint thoracic(k_forward = 120, k_lateral = 100,
    k_twist = 100, c = 12);

  // HEAD
  P.Body head(m = 4.0,
```

```
    I = [0.08, 0, 0; 0, 0.08, 0; 0, 0, 0.06],
    r_CM = {0, 0, 0.1});

  FlexibleJoint neck_joint(k = 100, c = 8);

  // ARMS (LEFT)
  FlexibleJoint shoulder_L(k = 180, c = 10);
  P.Body upper_arm_L(m = 2.0, r_CM = {0.17, 0, 0});
  FlexibleJoint elbow_L(k = 150, c = 8);
  P.Body forearm_L(m = 1.5, r_CM = {0.14, 0, 0});
  FlexibleJoint wrist_L(k = 100, c = 6);
  P.Body hand_L(m = 0.5, r_CM = {0.09, 0, 0});

  // LEGS (LEFT)
  FlexibleJoint hip_L(k = 300, c = 25);
  P.Body upper_leg_L(m = 8.0, r_CM = {0.22, 0, 0});
  FlexibleJoint knee_L(k = 250, c = 20);
  P.Body lower_leg_L(m = 5.0, r_CM = {0.22, 0, 0});

  // ADVANCED FOOT (LEFT)
  FlexibleJoint ankle_L(k_vertical = 250, k_forward = 150,
    k_lateral = 200, c = 15);
  P.Body heel_L(m = 0.2, r_CM = {0, -0.025, 0});
  FlexibleJoint foot_rise_L(k = 150, c = 12);
  P.Body midfoot_L(m = 0.3, r_CM = {0.05, 0, 0});
  FlexibleJoint metatarsal_L(k = 200, c = 12);
  P.Body forefoot_L(m = 0.3, r_CM = {0.065, 0, 0});

  // ... (Right arm and leg, mirrored)

equation
  // CONNECTIONS
  connect(world.frame_b, free_joint.frame_a);
  connect(free_joint.frame_b, pelvis.frame_a);
  connect(pelvis.frame_b, lumbar.frame_a);
  connect(lumbar.frame_b, torso.frame_a);
  connect(torso.frame_b, thoracic.frame_a);
  connect(thoracic.frame_b, neck_joint.frame_a);
  connect(neck_joint.frame_b, head.frame_a);

  // LEFT ARM
  connect(torso.frame_a, shoulder_L.frame_a);
  connect(shoulder_L.frame_b, upper_arm_L.frame_a);
  connect(upper_arm_L.frame_b, elbow_L.frame_a);
  connect(elbow_L.frame_b, forearm_L.frame_a);
  connect(forearm_L.frame_b, wrist_L.frame_a);
  connect(wrist_L.frame_b, hand_L.frame_a);
```

```
  // LEFT LEG
  connect(pelvis.frame_a, hip_L.frame_a);
  connect(hip_L.frame_b, upper_leg_L.frame_a);
  connect(upper_leg_L.frame_b, knee_L.frame_a);
  connect(knee_L.frame_b, lower_leg_L.frame_a);
  connect(lower_leg_L.frame_b, ankle_L.frame_a);
  connect(ankle_L.frame_b, heel_L.frame_a);
  connect(heel_L.frame_b, foot_rise_L.frame_a);
  connect(foot_rise_L.frame_b, midfoot_L.frame_a);
  connect(midfoot_L.frame_b, metatarsal_L.frame_a);
  connect(metatarsal_L.frame_b, forefoot_L.frame_a);

end HumanSkeleton3D;
```

5.2 Flexible Joint Component (Custom)

```
model FlexibleJoint
  parameter Real k = 150;           // Spring constant (N·m/rad)
  parameter Real c = 10;            // Damping (N·m·s/rad)
  parameter Real theta_max = 1.57;  // Max angle (~90°)
  parameter Real theta_min = -1.57;

  Real theta;    // Joint angle
  Real omega;    // Angular velocity
  Real tau;      // Applied torque

  Modelica.Mechanics.MultiBody.Interfaces.Frame_a frame_a;
  Modelica.Mechanics.MultiBody.Interfaces.Frame_b frame_b;
  Modelica.Blocks.Interfaces.RealInput u_tau;

equation
  der(theta) = omega;

  // Spring-damper-motor equation
  tau = -k * theta - c * omega + u_tau;

  // Rotational dynamics
  I_joint * der(omega) = tau - tau_friction;

  // Angular limits (soft constraints)
  if theta > theta_max then
    tau = tau - k_limit * (theta - theta_max);
  elseif theta < theta_min then
    tau = tau - k_limit * (theta - theta_min);
  end if;
```

```
end FlexibleJoint;
```

---

6. Implementation: Python-Based Approach (Recommended)

6.1 Why Python?

Advantages for AI Integration:

- Easy integration with LLM systems
- Better debugging and visualization
- Faster prototyping and parameter tuning
- Compatible with TensorFlow, PyTorch for machine learning
- Easier to generate training data

6.2 Complete Python Implementation

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

class FlexibleJoint3D:
    """3D flexible joint with spring-damper in all 3 axes"""

    def __init__(self, k_x=150, k_y=150, k_z=150, c=10, I=0.5):
        self.k = np.array([k_x, k_y, k_z])  # Spring constants
        self.c = c                           # Damping
        self.I = I                           # Inertia
        self.euler = np.array([0.0, 0.0, 0.0])  # Euler angles
        self.omega = np.array([0.0, 0.0, 0.0])  # Angular velocities

    def dynamics(self, tau_applied, dt=0.01):
        """Calculate rotational dynamics"""
        tau_spring_damper = -self.k * self.euler - self.c * self.omega + tau_appl
        alpha = tau_spring_damper / self.I
        return alpha

    def update(self, tau_applied, dt=0.01):
        """Update joint state over time step"""
```

```
        alpha = self.dynamics(tau_applied, dt)
        self.omega = self.omega + alpha * dt
        self.euler = self.euler + self.omega * dt


class FootSegment:
    """Advanced foot model: heel + midfoot + forefoot"""

    def __init__(self):
        self.heel = FlexibleJoint3D(k_x=150, k_y=200, k_z=150, c=12, I=0.2)
        self.midfoot = FlexibleJoint3D(k_x=160, k_y=180, k_z=160, c=12, I=0.3)
        self.forefoot = FlexibleJoint3D(k_x=200, k_y=150, k_z=300, c=12, I=0.3)

        self.m_heel = 0.2
        self.m_midfoot = 0.3
        self.m_forefoot = 0.3

        # Foot segment positions
        self.pos_heel = np.array([0, -0.025, 0])
        self.pos_midfoot = np.array([0.05, 0, 0])
        self.pos_forefoot = np.array([0.13, 0, 0])

    def get_toe_ball_position(self, ankle_position, ankle_euler):
        """Calculate world position of toe ball (forefoot)"""
        # Rotation matrices
        Rx = self._rotation_matrix_x(ankle_euler[0])
        Ry = self._rotation_matrix_y(ankle_euler[1])
        Rz = self._rotation_matrix_z(ankle_euler[2])
        R = Rz @ Ry @ Rx

        # Transform forefoot to world frame
        toe_ball_world = ankle_position + R @ self.pos_forefoot
        return toe_ball_world

    @staticmethod
    def _rotation_matrix_x(angle):
        c, s = np.cos(angle), np.sin(angle)
        return np.array([[1, 0, 0], [0, c, -s], [0, s, c]])

    @staticmethod
    def _rotation_matrix_y(angle):
        c, s = np.cos(angle), np.sin(angle)
        return np.array([[c, 0, s], [0, 1, 0], [-s, 0, c]])

    @staticmethod
    def _rotation_matrix_z(angle):
        c, s = np.cos(angle), np.sin(angle)
```

```
        return np.array([[c, -s, 0], [s, c, 0], [0, 0, 1]])

    def get_ground_contact(self, heel_z, forefoot_z, ground_z=0):
        """Determine foot contact phase"""
        heel_contact = heel_z <= ground_z
        ball_contact = forefoot_z <= ground_z

        return {
            'heel_contact': heel_contact,
            'ball_contact': ball_contact,
            'phase': 'heel' if heel_contact and not ball_contact
                   else ('ball' if ball_contact else 'flight')
        }


class HumanSkeleton3D:
    """Full 3D skeleton with advanced foot and upper body control"""

    def __init__(self):
        # Major joints
        self.joints = {
            'hip_L': FlexibleJoint3D(k_x=300, k_y=300, k_z=250, c=25, I=0.8),
            'hip_R': FlexibleJoint3D(k_x=300, k_y=300, k_z=250, c=25, I=0.8),
            'knee_L': FlexibleJoint3D(k_x=250, k_y=100, k_z=100, c=20, I=0.5),
            'knee_R': FlexibleJoint3D(k_x=250, k_y=100, k_z=100, c=20, I=0.5),
            'ankle_L': FlexibleJoint3D(k_x=150, k_y=200, k_z=250, c=15, I=0.4),
            'ankle_R': FlexibleJoint3D(k_x=150, k_y=200, k_z=250, c=15, I=0.4),
            'lumbar': FlexibleJoint3D(k_x=150, k_y=120, k_z=100, c=15, I=0.6),
            'thoracic': FlexibleJoint3D(k_x=120, k_y=100, k_z=100, c=12, I=0.4),
            'neck': FlexibleJoint3D(k_x=100, k_y=100, k_z=80, c=8, I=0.2),
            'shoulder_L': FlexibleJoint3D(k_x=180, k_y=180, k_z=180, c=10, I=0.3)
            'shoulder_R': FlexibleJoint3D(k_x=180, k_y=180, k_z=180, c=10, I=0.3)
            'elbow_L': FlexibleJoint3D(k_x=150, k_y=150, k_z=150, c=8, I=0.25),
            'elbow_R': FlexibleJoint3D(k_x=150, k_y=150, k_z=150, c=8, I=0.25),
        }

        # Foot models
        self.foot_L = FootSegment()
        self.foot_R = FootSegment()

        # Body masses
        self.masses = {
            'head': 4.0, 'torso': 20.0, 'pelvis': 8.0,
            'upper_arm_L': 2.0, 'forearm_L': 1.5, 'hand_L': 0.5,
            'upper_arm_R': 2.0, 'forearm_R': 1.5, 'hand_R': 0.5,
            'upper_leg_L': 8.0, 'lower_leg_L': 5.0, 'foot_L': 1.5,
            'upper_leg_R': 8.0, 'lower_leg_R': 5.0, 'foot_R': 1.5,
```

```
        }

        # Upper body hold parameters
        self.hold_params = {
            'k_shoulder': 800,
            'c_shoulder': 40,
            'k_frame': 700,
            'c_frame': 50,
            'shoulder_sep': np.array([0.15, 0.10, 0.05]),
        }

        # Root position
        self.pelvis_pos = np.array([0.0, 0.0, 0.9])
        self.pelvis_euler = np.array([0.0, 0.0, 0.0])

    def waltz_controller(self, t, beat_period=0.8):
        """Generate Waltz movement pattern"""
        phase = t % beat_period
        rise_duration = 0.3

        # RISE PHASE (0-0.3s)
        if phase < rise_duration:
            progress = phase / rise_duration
            rise_amount = 0.35 * np.sin(np.pi * progress)

            tau_ankle_L = np.array([0, 0, 50 * (rise_amount - self.joints['ankle_
            tau_ankle_R = np.array([0, 0, 50 * (rise_amount - self.joints['ankle_

        # FALL PHASE (0.3-0.8s)
        else:
            progress = (phase - rise_duration) / (beat_period - rise_duration)
            fall_amount = 0.35 * (1 - progress)

            tau_ankle_L = np.array([0, 0, -30 * self.joints['ankle_L'].omega[2]])
            tau_ankle_R = np.array([0, 0, -30 * self.joints['ankle_R'].omega[2]])

        # HIP/TORSO ROTATION
        hip_rotation_ref = 0.4 * np.sin(2*np.pi*t/beat_period)
        tau_hip = 100 * (hip_rotation_ref - self.joints['lumbar'].euler[2])
        tau_torso = 50 * (0.15 * np.sin(2*np.pi*t/beat_period + np.pi/4) - self.j

        # ARM POSITIONING (HOLD CONSTRAINT)
        tau_arm_L = 80 * (np.array([0.1, 0, 0.8]) - self.joints['shoulder_L'].eul
        tau_arm_R = 80 * (np.array([0.1, 0, 0.4]) - self.joints['shoulder_R'].eul

        return {
            'ankle_L': tau_ankle_L, 'ankle_R': tau_ankle_R,
```

```
            'lumbar': tau_hip, 'thoracic': tau_torso,
            'shoulder_L': tau_arm_L, 'shoulder_R': tau_arm_R,
        }

    def quickstep_controller(self, t, beat_period=0.5):
        """Generate Quickstep pattern with sharp weight transfers"""
        phase = t % beat_period

        # SHARP WEIGHT TRANSFER
        weight_on_ball = (1 + np.sign(np.sin(4*np.pi*t/beat_period))) / 2
        ball_flex_ref = 0.25 * np.sign(np.sin(4*np.pi*t/beat_period))

        # Sharp control gains
        Kp_sharp = 120
        Kd_sharp = 8

        tau_ankle_L = Kp_sharp * (ball_flex_ref - self.joints['ankle_L'].euler[2]
                      - Kd_sharp * self.joints['ankle_L'].omega[2]
        tau_ankle_R = Kp_sharp * (ball_flex_ref - self.joints['ankle_R'].euler[2]
                      - Kd_sharp * self.joints['ankle_R'].omega[2]

        # HIP PROPULSION
        hip_rotation_ref = 0.5 * np.sign(np.sin(4*np.pi*t/beat_period))
        tau_hip = 150 * (hip_rotation_ref - self.joints['lumbar'].euler[2])

        # FRAME MAINTAINED
        tau_arm_L = 100 * (np.array([0.1, 0, 0.8]) - self.joints['shoulder_L'].eu
        tau_arm_R = 100 * (np.array([0.1, 0, 0.4]) - self.joints['shoulder_R'].eu

        return {
            'ankle_L': tau_ankle_L, 'ankle_R': tau_ankle_R,
            'lumbar': tau_hip, 'thoracic': np.array([0, 0, 0]),
            'shoulder_L': tau_arm_L, 'shoulder_R': tau_arm_R,
        }

# SIMULATION EXAMPLE
if __name__ == "__main__":
    skeleton = HumanSkeleton3D()

    # Simulate 5 seconds of Waltz
    t_sim = np.linspace(0, 5, 500)
    waltz_results = {'ankle_L': [], 'toe_ball_L': [], 'torso': []}

    for t in t_sim:
        tau_dict = skeleton.waltz_controller(t, beat_period=0.8)
        skeleton.joints['ankle_L'].update(tau_dict['ankle_L'], dt=0.01)
```

```
        ankle_pos = skeleton.pelvis_pos + np.array([0, 0, -0.9])
        toe_ball_pos = skeleton.foot_L.get_toe_ball_position(ankle_pos,
                                                             skeleton.joints['ank

        waltz_results['ankle_L'].append(skeleton.joints['ankle_L'].euler[2])
        waltz_results['toe_ball_L'].append(toe_ball_pos[2])
        waltz_results['torso'].append(skeleton.joints['thoracic'].euler[2])

    # Plot results
    fig, axes = plt.subplots(3, 1, figsize=(12, 10))

    axes[0].plot(t_sim, waltz_results['ankle_L'], label='Ankle Angle (Z-rotation)
                 linewidth=2, color='#2E86AB')
    axes[0].set_ylabel('Angle (radians)')
    axes[0].set_title('Waltz: Ankle Rise and Fall', fontsize=14, fontweight='bold
    axes[0].grid(True, alpha=0.3)
    axes[0].legend()

    axes[1].plot(t_sim, waltz_results['toe_ball_L'], label='Toe Ball Height',
                 linewidth=2, color='#A23B72')
    axes[1].set_ylabel('Height (meters)')
    axes[1].set_title('Waltz: Toe Ball Rise and Fall', fontsize=14, fontweight='b
    axes[1].grid(True, alpha=0.3)
    axes[1].legend()

    axes[2].plot(t_sim, waltz_results['torso'], label='Torso Rotation (Z-axis)',
                 linewidth=2, color='#F18F01')
    axes[2].set_xlabel('Time (seconds)')
    axes[2].set_ylabel('Angle (radians)')
    axes[2].set_title('Waltz: Torso Rotation', fontsize=14, fontweight='bold')
    axes[2].grid(True, alpha=0.3)
    axes[2].legend()

    plt.tight_layout()
    plt.savefig('waltz_3d_simulation.png', dpi=150)
    plt.show()

    print("✓ 3D Waltz simulation complete!")
    print(f"Peak ankle angle: {max(waltz_results['ankle_L']):.3f} rad")
    print(f"Toe ball height range: {min(waltz_results['toe_ball_L']):.3f} to {max
```

---

7. Key Parameters Reference

Foot Parameters

```
TOE BALL STIFFNESS:  300 N·m/rad (high for crisp push-off)
HEEL RISE MAX:       0.35 rad (~20° rise, Waltz standard)
BALL FLEX MAX:       0.25 rad (~14° metatarsal flexion)
HEEL CONTACT:        Primary initial contact
BALL CONTACT:        Weight bearing during rise/push-off
```

Frame Parameters

```
FRAME STIFFNESS (Waltz):     700 N/m    (very stiff)
FRAME STIFFNESS (Tango):     650 N/m    (sharp transitions)
FRAME STIFFNESS (Quickstep): 500 N/m    (more reactive)

SHOULDER LOCK:       800 N/m (extremely rigid)
SHOULDER DAMPING:    40 N·s/m (prevents oscillation)
SHOULDER SEPARATION: 15cm forward, 10cm side, 5cm height
```

Arm Hold Parameters

```
ARM POSITION STIFFNESS:  80 N·m/rad (maintain line)
WRIST STIFFNESS:        100 N·m/rad (allow articulation)
LEFT ARM ANGLE:         0.8 rad (~46°) - held high
RIGHT ARM ANGLE:        0.4 rad (~23°) - slightly lower
```

Upper Body Parameters

```
TORSO TWIST LIMIT:       0.4 rad (~23° max rotation)
SPINE STIFFNESS:         150 N·m/rad (controlled bending)
NECK STIFFNESS:          100 N·m/rad (head stabilization)
LUMBAR SPRING:           150 N·m/rad
THORACIC SPRING:         120 N·m/rad
```

---

8. Validation Metrics

For assessing realism against professional dancers:

```
def calculate_metrics(simulated_angles, reference_angles):
    # Root Mean Square Error (RMSE)
    rmse = np.sqrt(np.mean((simulated_angles - reference_angles)**2))

    # Correlation coefficient
```

```
    correlation = np.corrcoef(simulated_angles, reference_angles)[0, 1]

    # Peak timing error
    peak_error = abs(find_peaks(simulated_angles)[0] -
                     find_peaks(reference_angles)[0])

    return {'rmse': rmse, 'correlation': correlation, 'peak_error': peak_error}
```

Target Metrics:

- RMSE < 0.05 rad (2.9 degrees)
- Correlation > 0.85
- Peak timing error < 50 ms (milliseconds)

---

9. Implementation Timeline

Phase	Duration	Deliverables	Priority
1. Foundation	Weeks 1-2	Basic 3D skeleton, 8 major joints	Critical
2. Foot Detail	Weeks 3-4	Heel-midfoot-forefoot structure	Critical
3. Frame Control	Weeks 5-6	Shoulder constraints, arm positioning	High
4. Dance Patterns	Weeks 7-8	All 10 dances implemented	High
5. Validation	Weeks 9-10	Mocap comparison, tuning	High

---

10. Next Steps

1. **Choose Implementation Path**: OpenModelica (pure physics) or Python (LLM-friendly)
2. **Start Small**: Begin with 4-joint model (ankles + hips)
3. **Obtain Reference Data**: Gather motion capture from professional Blackpool-level dancers
4. **Iterative Tuning**: Adjust spring/damping constants against real movement
5. **Expand Complexity**: Add remaining joints and constraint layers

6. **Integration**: Connect to LLM text-to-motion generation system

---

Resources

- **OpenModelica Documentation**: https://openmodelica.org/
- **Modelica Standard Library**: Multibody mechanics
- **Motion Capture Formats**: BVH, C3D, FBX
- **Professional Standards**: Blackpool Dance Festival technical specifications
- **Control Theory**: PID tuning, LQR optimization

Conclusion

This 3D flexible human skeleton model provides the foundation for generating authentic ballroom dance movements using physics-based simulation. The key innovations are:

1. **Advanced foot modeling** with separate heel, midfoot, and forefoot segments
2. **Frame rigidity constraints** that enforce ballroom aesthetics
3. **Dance-specific control patterns** that capture unique characteristics of each dance
4. **Flexible joint mechanics** that allow realistic joint behaviors

By combining Lagrangian mechanics with ballroom-specific constraints, we can generate movements that are both physically realistic and aesthetically pleasing—meeting the rigorous standards of competitive ballroom dancing.

Last Updated: February 5, 2026
Version: 3.0 - Full 3D with Advanced Foot & Hold Constraints
Status: Ready for Implementation