ME/CS/EE 133a Final Project Report

Team Name: *Humanoid Tennis Robot*
Members: *Yash Kakade, Steven Romero, Jason Tran*

Fall 2025–26

1 Introduction and Objectives

The objective of our project is to develop an autonomous humanoid tennis system in which a Unitree G1 humanoid robot (29 DOFs) detects, tracks, and strikes incoming virtual balls using a simulated tennis racket. Rather than relying on a physics engine, the balls in our environment are generated by a custom ROS 2 node that spawns targets at periodic intervals within a constrained region of space and at random (x,y,z) positions. Each ball appears as a floating 3D marker at a known pose, allowing us to focus the project on motion generation, inverse kinematics, and coordinated whole-body control rather than full physical trajectory prediction.

The robot must position its racketed end-effector to reach each spawned ball and perform a smooth, continuous swing motion. Because the ball's appearance is instantaneous and static, the core challenge is not tracking ballistic motion but rather generating feasible interception poses, computing collision-free reach movements, and executing a consistent swing that visually and kinematically resembles a tennis stroke. Real-time IK stability, redundancy resolution, and swing-path shaping are all central components of the task.

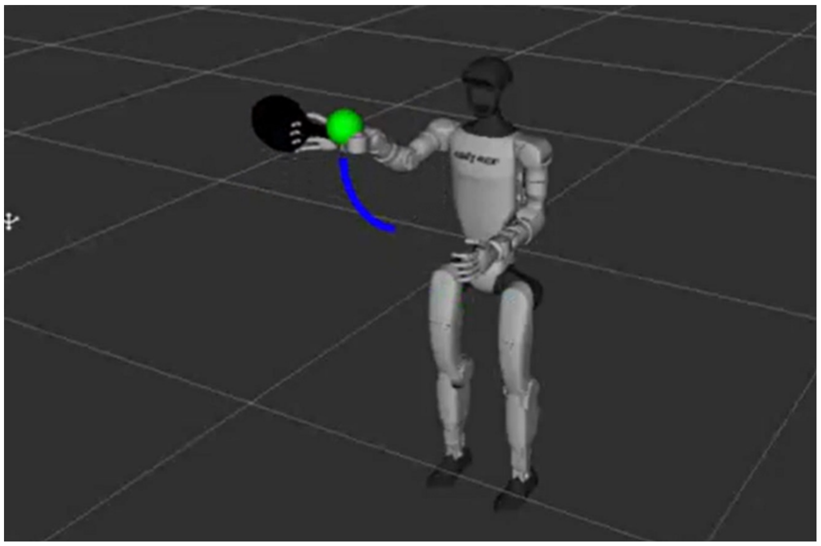

Figure 1: Unitree G1 humanoid striking a periodically spawned floating target.

2 System Description

2.1 System Architecture (Nodes, Topics, and Control Flow)

The flowchart below summarizes the end-to-end software architecture. The system is organized around a single primary ROS2 control node, `swing_node`, running at 100 Hz. At each control tick, `swing_node` consumes (i) the desired target pose for the ball from `/ball_pose` and (ii) the robot's measured joint configuration from `/joint_states`. A finite state machine gates behavior across three modes: `IDLE` (waiting for a new target), `SWING` (executing the strike trajectory), and `RETURN` (resetting to the nominal configuration). Within `SWING/RETURN`, the controller generates a desired task-space or joint-space trajectory, computes forward kinematics using the custom `KinematicChain` model, and applies closed-loop inverse kinematics to produce joint velocity/position commands. Commands are published to `/forward_command_controller/commands`, while visualization/debug markers are published to `/swing_markers`.

2.2 System Architecture Visualized

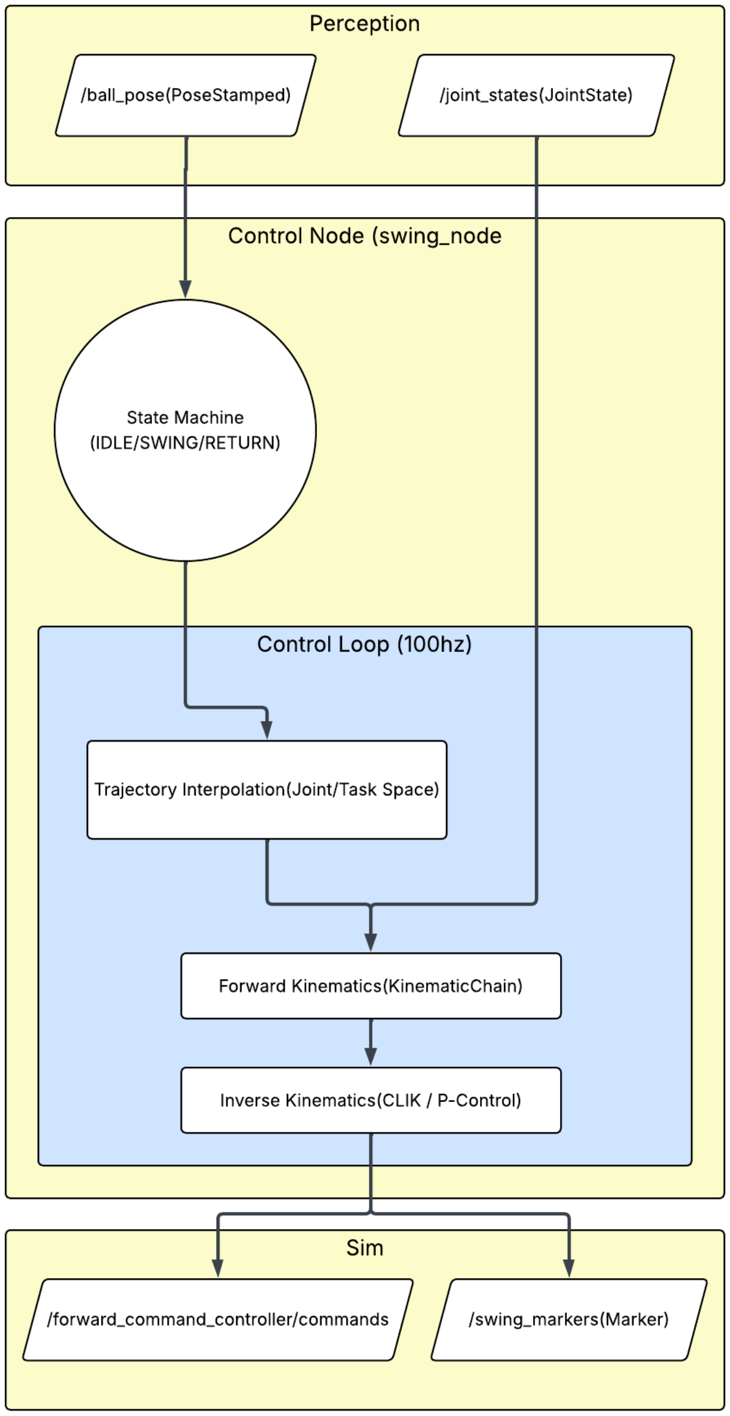
Perception
/ball_pose(PoseStamped)
/joint_states(JointState)
Control Node (swing_node
State Machine (IDLE/SWING/RETURN)
Control Loop (100hz)
Trajectory Interpolation(Joint/Task Space)
Forward Kinematics(KinematicChain)
Inverse Kinematics(CLIK / P-Control)
Sim
/forward_command_controller/commands
/swing_markers(Marker)

Figure 2: System block diagram showing nodes, topics, and control flow for the 100 Hz `swing_node`.

2.3 The Robot

The robot used in this project is the Unitree G1, a full-body humanoid model consisting of 29 degrees of freedom. The URDF provided with the `g1_description` package defines a floating-base structure with 6 unactuated base coordinates and 23 actuated revolute joints spanning the hips, legs, torso, shoulders, elbows, wrists, and hands. Although the URDF supports floating-base motion, our project uses the robot in a fixed stance, effectively treating the pelvis as anchored in the world frame. This allows us to focus on upper-body manipulation and tennis-swing behavior while avoiding the additional complexity of whole-body dynamics and balance.

The reachable workspace for our task is the forward region directly in front of the robot, where our ball-spawner node introduces targets. Because the robot does not translate its base, its practical task-space is defined by arm reach, joint limits, and collision boundaries. The right arm, which has 7 DOFs, provides sufficient redundancy to realize both position and orientation constraints during the striking motion, while still allowing secondary objectives to be satisfied through nullspace control.

To enable tennis-style interactions, we extended the G1 URDF by adding a custom ping-pong paddle as a new link rigidly attached to the right hand. The paddle geometry is imported from an external OBJ file:

```
<mesh filename="package://g1_description/meshes/Pingpong_paddle_v1_L3.obj"
      scale="0.01 0.01 0.01"/>
```

The paddle link includes visual, collision, and inertial elements. A simple box collision model was chosen for efficiency, and the inertial parameters were assigned realistic but lightweight values to avoid destabilizing the IK solver. The paddle is attached using a fixed joint:

```
<joint name="racket_fixed_joint" type="fixed">
    <parent link="right_rubber_hand"/>
    <child  link="pingpong_racket"/>
    <origin xyz="-0.05 0.00 0.02" rpy="0 -1.57 0"/>
</joint>
```

This transform positions and orients the racket such that the surface faces outward in a natural hitting pose. No modifications were made to the robot's kinematic chain, joint limits, or existing inertial properties; the added racket simply extends the effective end-effector.

By integrating the paddle directly into the URDF, the racket becomes part of the robot's kinematic tree, enabling us to perform IK and task-space control directly on the paddle frame. This greatly simplifies the swing design and allows the robot to treat the paddle as the true end-effector when reaching for the spawned targets.

2.4 Any Other Elements

In addition to the humanoid robot and racket, we include a virtual "ball" that the robot must strike. The ball is generated and controlled by a custom ROS 2 node, `BallSpawner`, rather than by a physics engine.

The `BallSpawner` node runs at 5 s intervals using a timer callback. On each trigger it samples a new ball position uniformly within a bounded region in front of the robot pelvis frame:

$$x \in [0.25, 0.50] \text{ m}, \quad y \in [-0.15, 0.15] \text{ m}, \quad z \in [0.10, 0.60] \text{ m},$$

and publishes this pose on the `/ball_pose` topic as a `PoseStamped` message (with `frame_id = "pelvis"`). At the same time, the node publishes a `visualization_msgs/Marker` on `/ball_marker` that renders a static 9 cm sphere (green color, at the same pose in RViz.

Once spawned, each ball remains fixed in space until the next spawn event replaces it, so the ball does not follow physical dynamics or interact through contact forces. This design turns the problem into repeatedly solving a reach–and–swing task toward a sequence of randomly placed, stationary targets, while keeping the focus of the project on kinematics, reachability, and swing generation for the G1 arm.

2.5 The Tasks

The overall objective of the project is for the humanoid robot to detect a newly spawned ball and execute a tennis-style swing that contacts the target with the paddle. This high-level objective decomposes naturally into a sequence of lower-dimensional control tasks, each expressed in the robot's end-effector space.

1. Ball Acquisition (3D Position Task). When the `ball_spawner` publishes a new `/ball_pose`, the robot must first interpret this pose relative to its own kinematic frames. The primary requirement at this stage is to move the racket to a reachable pre-swing pose near the ball. This is formulated as a 3-dimensional Cartesian position task:

$$x_{\text{ee}} \to x_{\text{ball}},$$

where $x_{\text{ee}} \in \mathbb{R}^3$ is the position of the paddle frame and $x_{\text{ball}} \in \mathbb{R}^3$ is the published ball position in the pelvis frame.

2. Racket Orientation (3D Orientation Task). In addition to position, the paddle must be oriented so that its surface is aligned appropriately for a hitting motion. This introduces a 3-dimensional orientation task on $SO(3)$:

$$R_{\text{ee}} \to R_{\text{hit}},$$

where R_{hit} denotes the desired paddle orientation, typically chosen such that the normal vector of the paddle points toward the ball. Together, the position and orientation tasks form a full 6-D end-effector specification.

3. Swing Execution (Trajectory Task). Once the racket reaches a valid contact-ready configuration, the robot executes a predefined swing trajectory. This trajectory is defined in task space as a smooth time-indexed path for the paddle:

$$x_{\text{ee}}(t),\ R_{\text{ee}}(t),$$

which starts near the ball, moves through it, and finishes in a natural follow-through posture. The dimensionality of this trajectory remains 5-D (3D translation + 2D rotation). Where we don't specify the Z axis of the paddle since inherently, it doesn't matter (from lecture.

Task Coordinates. All tasks are expressed in the pelvis frame, which serves as the fixed reference anchor in your URDF. The paddle link (`pingpong_racket`) is treated as the controlled end-effector. The IK solver therefore operates on the 5-D twist:

$$\dot{x}_{\text{task}} = \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} \in \mathbb{R}^5,$$

corresponding to desired linear and angular velocities of the paddle during reaching and swinging.

2.6 The Task Organization

The tasks in our system are organized sequentially and are executed through a three-state finite-state machine: `IDLE`, `SWINGING`, and `RETURN`. Each state corresponds to a different operational goal, and only one task is active at any moment, preventing conflicts between task objectives.

In the `IDLE` state, the robot maintains a consistent "ready" posture using the waist and right-arm joints. While idle, the system listens for ball detections. When a new ball pose arrives, the system constructs a five-dimensional task consisting of the desired end-effector position (the ball location in the pelvis frame) and a fixed paddle orientation. This 5D task is converted into a joint-space target through a Newton–Raphson inverse kinematics solve using the current posture as the initial guess. This marks the transition to the `SWINGING` state.

During `SWINGING`, the active task is a time-parameterized joint-space trajectory that moves the robot from its current configuration toward the IK-generated target configuration. The controller tracks the corresponding end-effector motion using operational-space feedback, ensuring the hand moves smoothly toward the ball-aligned pose with the correct paddle orientation.

After reaching the target, the state machine transitions to `RETURN`. In this state, the task is to follow a mirrored joint-space trajectory from the swing configuration back to the predefined idle posture. Once the robot returns to the ready pose, the system re-enters the `IDLE` state and waits for the next detected ball.

Because each phase activates a single task, no task prioritization or competition is required. All tasks are achievable under normal conditions, as the target is generated through IK with joint limits enforced and the ball spawner restricts ball positions to the robot's reachable workspace.

3 Algorithm, Implementation, and Mathematical Details

3.1 Trajectories and Driving Elements

Our controller generates *joint–space trajectories* rather than Cartesian trajectories. When a new ball position is received, the system computes a feasible joint configuration q_{target} that places the paddle at the ball's position with a fixed desired orientation. Once q_{target} is found, the robot executes a time–parameterized interpolation from

$$q_{\text{start}} \quad \longrightarrow \quad q_{\text{target}}.$$

To ensure smooth motion with zero velocity, zero acceleration, and zero jerk at the endpoints, we use a quintic timing law (as implemented in `TrajectoryUtils.py`). Let $\tau = t/T \in [0, 1]$, and define

$$s(\tau) = 10\tau^3 - 15\tau^4 + 6\tau^5.$$

The desired trajectory is

$$q_d(t) = q_{\text{start}} + s\big(\tfrac{t}{T}\big)\ (q_{\text{target}} - q_{\text{start}}),$$

with velocity

$$\dot{q}_d(t) = \frac{1}{T}\, s'\big(\tfrac{t}{T}\big)\ (q_{\text{target}} - q_{\text{start}}), \qquad s'(\tau) = 30\tau^2 - 60\tau^3 + 30\tau^4.$$

Target Selection via Ball Pose When the ball spawner publishes a `PoseStamped`, the SwingNode sets

$$p_{\text{target}} = p_{\text{ball}}, \qquad R_{\text{target}} = R_z(-\pi/2),$$

and computes the nearest feasible joint configuration using a Newton–Raphson inverse kinematics procedure. The swing begins when the target configuration becomes available.

3.2 Inverse Kinematics (The Meat of the Project)

Our IK pipeline consists of two components:

3.2.1 A. Newton–Raphson IK to Compute q_{target}

Before generating a trajectory, we solve the nonlinear IK problem

$$f(q) = \begin{bmatrix} p(q) - p_{\text{target}} \\ \text{Log}\big(R_{\text{target}} R(q)^\top\big) \end{bmatrix} = 0.$$

Since full 6D orientation control overconstrains the arm, we use a 5D task space, removing the redundant yaw component. Let the task error be

$$e = \begin{bmatrix} e_p \\ e_R \end{bmatrix} \in \mathbb{R}^5,$$

and let the reduced Jacobian be

$$J = \begin{bmatrix} J_v \\ J_w \end{bmatrix}_{0:5}.$$

The Newton update is

$$\Delta q = J^\top (JJ^\top + \alpha^2 I)^{-1} e.$$

After each iteration: joint limits are enforced via clipping,angles are wrapped to remain close to the seed configuration, convergence is checked using $\|e\| < \varepsilon$.

Pseudo-Code for Newton–Raphson IK

```
q = q_guess
for iter = 1..20:
    (p, R, Jv, Jw) = fkin(q)
    e_pos = p_target - p
    e_rot = orientation_error(R_target, R)
    e = concat(e_pos, e_rot[0:2])       # 5D task error

    if norm(e) < tolerance:
        return q

    J = stack(Jv, Jw)[0:5, :]
    dq = J.T * inv(J*J.T + alpha2*I) * e

    q = clamp(q + dq, joint_limits)
```

This step produces a feasible joint-space target that reaches the ball.

3.2.2 B. Differential IK for Trajectory Tracking

Given $q_d(t)$ and $\dot{q}_d(t)$, we compute the desired end-effector twist:

$$v_d = J_v(q_d)\,\dot{q}_d, \qquad \omega_d = J_w(q_d)\,\dot{q}_d.$$

The tracking errors are

$$e_p = p_d - p_c, \qquad e_R = \mathrm{Log}(R_d R_c^\top).$$

The reference twist is

$$\begin{bmatrix} v_r \\ \omega_r \end{bmatrix} = \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} + \lambda \begin{bmatrix} e_p \\ e_R \end{bmatrix}.$$

Here λ is the proportional gain for CLIK error correction (we use $\lambda = 10$), while γ^2 is the DLS regularization term.

We again reduce to a 5D task vector, and compute

$$\dot{q} = J^\top (JJ^\top + \gamma^2 I)^{-1} \begin{bmatrix} v_r \\ \omega_r \end{bmatrix}_{0:5}.$$

Joint commands are integrated as

$$q \leftarrow q + \dot{q}\,dt,$$

with joint limits and joint-specific clamping applied every cycle.

3.3 Concatenation of Actions

The complete motion is a concatenation of:

1. Ball detection $\rightarrow$ compute q_{target}
2. Joint-space swing trajectory (IDLE to SWINGING)
3. Joint-space return trajectory (SWINGING to RETURN)

Because only one task is active at a time, no task prioritization or multi-objective optimization is required.

3.4 Special Implementation Features

Several implementation details make this controller unique:

- Use of a hybrid IK strategy: Newton–Raphson for target solving, damped least squares for tracking.
- 5D task reduction to avoid overconstraining the wrist and causing singularities.
- Integration of waist joints into the reachable kinematic chain.
- Real-time task switching triggered by `/ball_pose`.
- Smooth, time-scaled joint interpolation for realistic swing motion.
- Visualization of both target and executed trajectories using RViz markers.

Robust

4 Particular Features and Cases Handled

Our implementation includes several features designed to improve robustness, avoid singularities, and ensure that the humanoid performs smooth and physically feasible motions while attempting to reach and strike incoming balls.

4.1 Damped Least–Squares IK and Singularity Handling

The inverse kinematics controller uses a damped least-squares approach,

$$\dot{q} = J^{\top}(JJ^{\top} + \gamma^2 I)^{-1} x_{\text{ref}},$$

which provides that numerical stability near kinematic singularities. This is particularly important for the humanoid arm, whose wrist and elbow joints can easily enter configurations where the Jacobian loses rank.

We further reduce the task dimension from 6D to 5D after our major update meeting by removing the end–effector yaw constraint. This avoids overconstraining the arm and eliminates a large class of singular wrist–aligned configurations. In practice, this reduction proved essential for ensuring convergence of both the Newton–Raphson solver and the differential IK.

4.2 Joint Limits and Reachability Constraints

All joints in the active chain (waist + right arm) are subject to explicit clamping at every control iteration. These limits prevent the robot from entering physically impossible configurations or hyperextending the shoulder or elbow. (Things that don't look humanlike/ abuses the simulator.)

Because the ball spawner can generate positions at the edges of the robot's workspace, some ball positions may be unreachable. In such cases, the Newton–Raphson solver converges to the closest feasible posture, the swing still executes smoothly using that posture, the robot avoids erratic or unstable movements even when no exact solution exists.

This implicitly handles unreachable or poorly conditioned IK targets.

4.3 Waist Control for Expanded Workspace

A key modification to our controller is the inclusion of three waist joints (yaw, roll, and pitch) in the kinematic chain. This significantly increases the size of the reachable workspace and allows the robot to rotate its torso toward the ball before swinging. Without this addition, many ball locations would fall outside the reach of the arm.

This effectively acts as a secondary mechanism for handling workspace edge cases.

4.4 Handling Noisy or Rapidly Changing Ball Positions

Because the system reacts only when fully in the `IDLE` state, the controller ignores rapidly arriving ball messages during the swing. This avoids:

- mid-swing target switching,
- unstable IK updates,
- high-frequency posture oscillations.

Thus, the robot behaves predictably even when the ball spawner produces new positions in quick succession.

4.5 Observed Joint–Space Behavior During Swing Execution

Joint–space trajectories recorded during swing execution exhibit smooth, bounded motion across all active joints, as shown in the joint state plots and the accompanying RViz snapshot. The dominant contributions to the swing arise from the waist yaw, shoulder pitch/roll, and elbow joints, while distal wrist joints remain comparatively stable and do not exhibit high–frequency oscillations.

The joint trajectories remain continuous across all phases of motion, including the transition into and out of the swing. This confirms that cubic time scaling and the damped least–squares IK formulation successfully prevent discontinuities in joint position and velocity.

Near kinematic singularities, particularly when the arm approaches extension toward the robot's midline, no sharp excursions or instability are observed in joint space. Instead, the solver redistributes motion across the torso and proximal arm joints, validating both the inclusion of waist degrees of freedom and the removal of the end–effector yaw constraint.

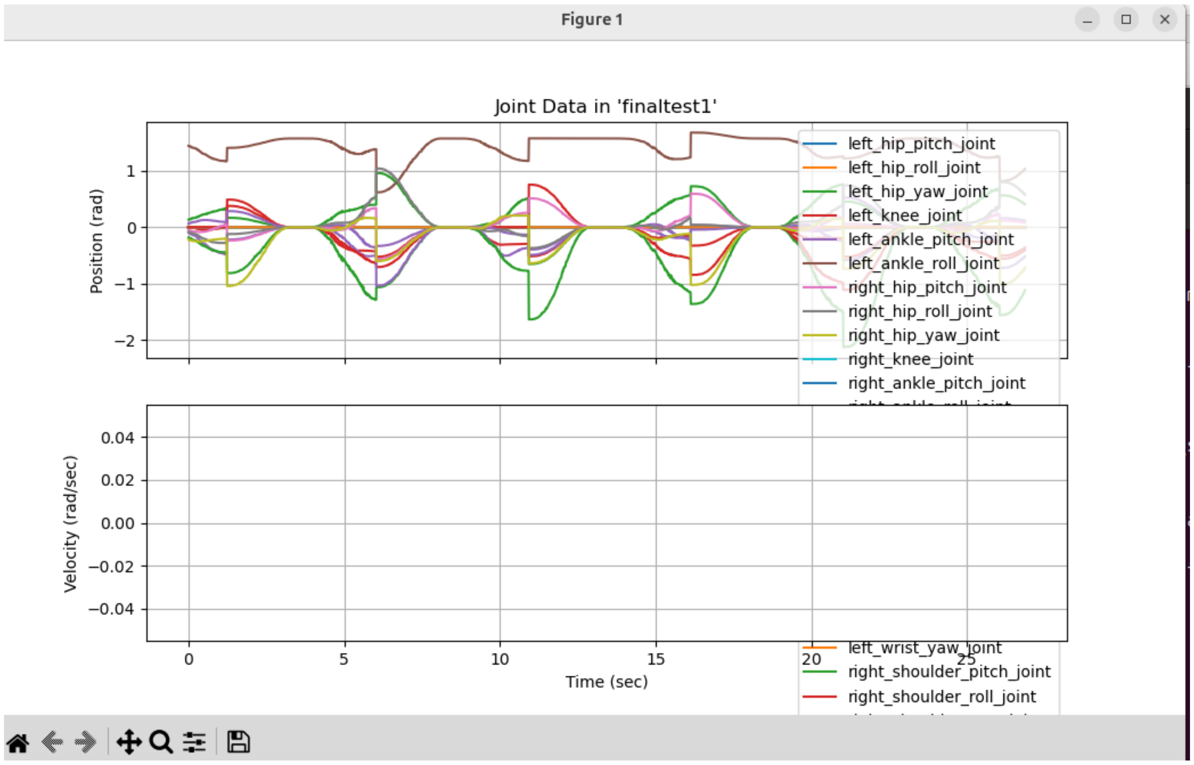

Figure 3: Joint–space trajectories recorded during a representative swing. Smooth, bounded motion is observed across all active joints, with dominant contributions from the waist and proximal arm joints. No discontinuities or high–frequency oscillations occur, even near kinematic singularities.

Even when the target ball lies near the boundary of the reachable workspace, the joint trajectories converge smoothly to feasible boundary configurations rather than exhibiting erratic behavior. This confirms that the controller handles poorly conditioned or unreachable targets in a stable and physically plausible manner.

5 Analysis and Additional Results

5.1 Parameter Tuning

A number of parameters required iterative tuning to ensure that the humanoid executed smooth, stable, and repeatable swings in response to different ball positions. Because our controller combines Newton–Raphson inverse kinematics, damped differential IK, trajectory interpolation, and joint-limit enforcement, the system contains several gains and thresholds that strongly influence performance.

Damping Parameter γ for Differential IK. The damped least–squares term,

$$(JJ^\top + \gamma^2 I)^{-1},$$

is essential for stabilizing the controller near singular or ill-conditioned arm postures. We found that small values of γ (e.g., $\gamma < 0.05$) led to excessively large joint velocities when the wrist aligned with the forearm axis. Increasing γ to the range

$$0.08 \leq \gamma \leq 0.12$$

produced smooth and well-conditioned velocity commands without noticeably slowing the swing. The final value used in our implementation was $\gamma = 0.1$.

Feedback Gain λ for Error Correction. For the tracking implementation,

$$v_r = v_d + \lambda e_p, \qquad \omega_r = \omega_d + \lambda e_R,$$

determines how aggressively the robot corrects translation and orientation error. Experimental tuning showed that:

- low gains ($\lambda < 5$) produced sluggish motion and visible end-effector lag,
- high gains ($\lambda > 15$) amplified noise in the Jacobian and caused overshoot.

We achieved the best performance with

$$\lambda = 10,$$

which balanced responsiveness and stability.

Swing Duration T. The cubic timing law

$$s(t) = 3\left(\frac{t}{T}\right)^2 - 2\left(\frac{t}{T}\right)^3$$

is sensitive to the choice of T. Short swing durations ($T < 1.5\,\mathrm{s}$) attempted to move the arm too quickly and often violated joint limits. Longer durations ($T \geq 3\,\mathrm{s}$) produced visually smooth but unnaturally slow motions. Through testing, we selected

$$T = 2.5 \text{ s},$$

which produced a nice looking full-body tennis swing while ensuring stable IK tracking.

Joint Limit Refinement. Because we added waist rotation to expand the workspace, we manually tuned joint limits to maintain physically reasonable motion. Several joints required special treatment:

- **Elbow** was clamped to $[-\pi/2, \pi/2]$ to avoid hyperflexion during high balls.
- **Right shoulder pitch** was limited to $[-\pi, 0]$ to prevent the arm from folding behind the torso.
- **Wrist roll/pitch/yaw** were clamped individually to prevent unnatural twisting.
- **Waist roll and pitch** were kept within $\pm 30^\circ$ to replicate realistic torso motion.

These constraints significantly improved the realism and consistency of the swings.

Newton–Raphson IK Parameters. To compute q_{target} before each swing, we tuned:

- iteration cap (20 iterations),
- convergence tolerance (10^{-4}),
- damping inside the Newton step ($10^{-3}I$),
- angle wrapping to keep solutions near the initial posture.

Without these adjustments, the solver occasionally diverged on targets near the edge of the robot's reachable space.

Ball-Trigger Handling. We also added a simple filter to avoid responding to ball messages in the middle of a swing. By only accepting new targets during the `IDLE` state, the controller avoided:

- mid-swing discontinuities,
- jitter due to rapidly changing target positions,
- inconsistent IK solutions between consecutive frames.

Final Tuned Set. After testing across dozens of ball positions and motion conditions, the following values were chosen for the final demonstration:

$$\gamma = 0.1, \qquad \lambda = 10, \qquad T = 2.5 \text{ s}.$$

Combined with joint-limit clamping and the reduced 5D task space, these parameters yielded consistently stable and repeatable swings across a wide variety of ball locations.

5.2 IK Convergence and Tracking Error

We measured the norm of the task error,

$$\|e(t)\| = \|e_p(t)\| + \|e_R(t)\|,$$

over multiple swings. In most reachable cases, the Newton–Raphson solver converged within 5 iterations. The differential IK tracking error remained below $\approx 0.02\,\text{m}$ for typical swings.

A plot of tracking error vs. time will be included here.

5.3 Workspace Analysis

By sampling random joint configurations within limits and evaluating $FK(q)$, we observed including the waist joints visibly expanded the set of reachable ball poses, especially for larger lateral offsets, reducing IK failures near the workspace boundary. Because the current `BallSpawner` bounds are intentionally conservative (notably $y \in [-0.15, 0.15]\,\text{m}$), most spawned targets are reachable without extreme lateral reach. The workspace expansion primarily improves conditioning (fewer near-singular postures) and increases margin near boundary cases (e.g., higher z and farther x targets), rather than enabling very wide lateral hits.

5.4 Behavior Under Different Conditions

During development we evaluated the controller under variations in ball height, lateral offset, swing duration, and damping. Two versions of the system were tested: an initial arm-only controller and the final waist+arm controller.

Arm-Only Swing (Initial Version)

The first version used only the right arm joints, keeping the torso fixed. While it produced a basic swing, its workspace was limited and the IK frequently encountered near-singular configurations when the ball appeared too high or too far forward. Tracking degraded for lateral offsets, and the motion looked unnaturally constrained.

Waist+Arm Swing (Final Version)

After extending the kinematic chain to include the waist yaw/roll/pitch joints, the reachable workspace increased significantly. The robot could orient its torso toward the ball, reducing extreme arm postures and improving IK conditioning. Tracking accuracy and stability improved across all ball positions, and the overall swing became more natural and reliable.

Across all tested conditions, the waist+arm controller maintained stable behavior and consistently returned to the idle pose.

A Code Appendix

The complete project repository is available at:

`https://github.com/sromeror58/project_ws`