Xsens

TECHNICAL CASE STUDY

From Motion Capture to Live Humanoid Teleoperation

A technical case study of the Xsens Humanoid Teleoperation Demo Pipeline, and a guide to building your own

Overview

Abstract

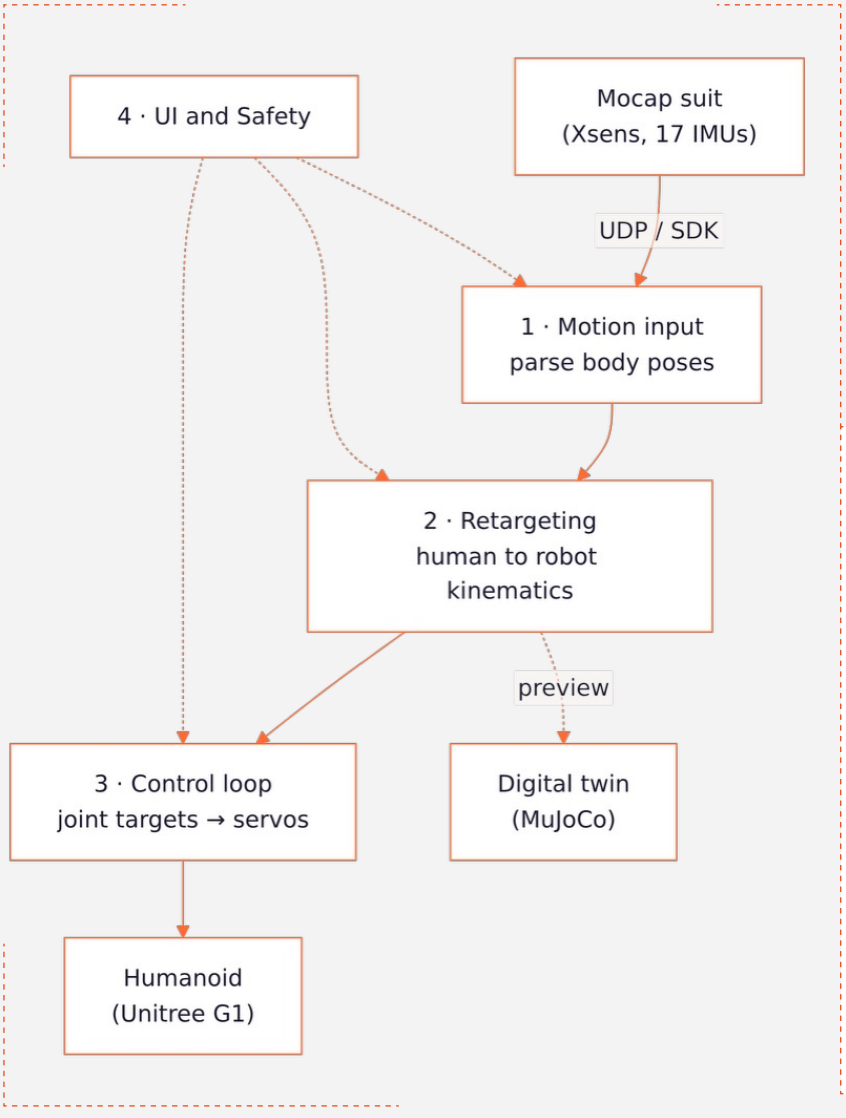
Teleoperating a humanoid robot with your own body is one of the most direct human-robot interfaces available: an operator moves, and the robot moves with them. At Xsens we built a demo pipeline, the ***Xsens Humanoid Teleoperation Demo Pipeline:*** *an operator workstation that turns live Xsens motion capture into real-time joint commands for a Unitree G1 humanoid.*

*This paper describes the architecture at a high level and maps open-source building blocks as a guidance for creating a similar pipeline. We use the Unitree G1 as a concrete example, but the structure (****motion input - retargeting - control loop,*** *wrapped in a UI and a safety layer) is humanoid-agnostic.*

A technical case study of the Xsens Humanoid Teleoperation Demo Pipeline, and a guide to building your own - Xsens · August 2026

1 System overview

A motion-capture teleoperation pipeline is four stages and two cross-cutting concerns:



System diagram

1. **Motion input:** receive a stream of human body poses.
2. **Retargeting:** convert human motion into robot joint configurations.
3. **Control loop:** deliver joint targets to the robot at servo rate, safely.
4. **UI and safety:** the layer that makes it operable and trustworthy.

The first three stages represent the pipeline. The fourth is an important difference between a research artifact and something an operator can run. Each section below covers what the stage does, the options, and the open-source repositories to start from, collected in the Appendix at the end of this document.

2 Motion input

2.1 What the data looks like

The Xsens Humanoid system pairs Xsens Link hardware (an inertial suit with 17 IMUs) with the Xsens Humanoid software, which fuses sensor data into a full-body skeleton: on the order of 23 segments, each with a 3D position and an orientation quaternion, updated at a high, steady rate (well above what a robot control loop needs).

One important detail matters: calibration. The operator's limb lengths and rest pose differ from the robot's. A short calibration step (e.g. capturing a known reference pose) lets you track orientation deltas from that reference rather than absolute geometry, decoupling operator body proportions from robot kinematics.

2.2 Four ways in

1. **UDP streaming integration —** the Xsens Humanoid software streams a documented datagram protocol on a configurable port, covering both live capture and .mvn file playback. The bytes need to be parsed (~200 lines for the datagrams required), which keeps the integration language-agnostic and independent of any SDK version.

2. **SDK integration —** the official Xsens XME SDK (C++/Python, licensed download). Direct API access gives it the lowest latency, and adds device control on top of live streaming and .mvn playback.

3. **ROS UDP streaming —** the stream node of the official **Xsens ROS 2 driver** receives the same UDP stream and publishes it into the ROS 2 graph: joint_states (joint angles for all 23 joints), link_states (pose, velocity, and acceleration for all 24 body segments), the centre of mass, and TF transforms.

4. **ROS SDK integration —** the XME node of the same driver connects directly to the suit hardware via the XME SDK (no separate streaming PC needed) and adds calibration services and hardware diagnostics on top of the same topics. A companion URDF publisher node generates a body-scaled model of the subject for visualization.

Xsens' choice (example): the live demo path is **UDP streaming.** It is the fastest way to a working feed and keeps the pipeline independent of SDK versions

Start here: parse the UDP quaternion datagram. The building blocks for each stage are listed in the **Appendix.**

3 Retargeting

Retargeting is the heart of the problem and where most of the engineering effort goes. The human and robot share neither proportions nor joint layout, so a pose that looks right on a person must be translated into joint angles that are reachable, balanced, and safe on the robot.

3.1 The landscape

The retargeting problem can be solved in three different directions: differential/analytic IK, optimization-based solving, and learned/policy-based approaches.

Family	How it works	Strengths	Costs
Differential / analytic IK	Per-frame solve of joint angles from end-effector + posture targets (e.g. on MuJoCo or Pinocchio)	Fast, transparent, easy to start, runs on CPU	Can struggle with balance, self-collision, and aggressive motions
Optimization-based	Solve a constrained optimization per frame (limits, collisions, contacts)	Higher quality, handles constraints explicitly	Heavier; cost-function tuning is robot- and task-specific
Learned / policy-based	A trained model maps reference motion to robust control	Best fidelity & robustness under hard/noisy motion	Needs data, training, and infrastructure; the model is the IP

3.2 Recommendation

For a newcomer, **start with differential inverse kinematics (IK),** building on an open-source solver such as Mink or Pink rather than writing your own: define an end-effector task for each hand (and feet/pelvis as needed) plus a posture regularizer, and solve each frame. You will have a robot mirroring an operator within a day, and you will learn exactly where analytic IK breaks: balance under fast motion, self-collision, and the wrists. When you hit those walls, the path forward is the **learned / policy-based** family: first learned retargeting (real-time retargeters tuned for RL tracking), and ultimately policy-based whole-body control, where a trained policy converts the reference motion into stable, balanced whole-body commands on the robot.

This staged path (analytic - learned) works well in practice: ship the IK version to prove the loop end-to-end, then move to the learned stack where higher fidelity and robustness are needed.

Start here: Mink (IK on MuJoCo) or Pink (IK on Pinocchio); graduate to GMR (General Motion Retargeting) and policy methods. See the **Appendix.**

4 Control loop

4.1 Bridging two rates

Your retargeter produces joint targets at one rate (say ~100–200 Hz). The robot's servos require faster and smoother commands: discontinuities become jerks, and jerks become unsafe.
The standard fix is a **dual-rate loop:** run retargeting at its natural rate, then **interpolate** (linear or higher-order) and **velocity-clip** up to the servo command rate. The same dual-rate pattern is used when deploying reinforcement-learning policies on these robots, so it carries over unchanged if you later move to a learned controller.

4.2 Transport: DDS and ROS 2

For the G1 we use the **native DDS** path: Unitree's SDK2 over CycloneDDS. For a single, well-characterized robot it gives the tightest, most predictable loop and the fullest access to the robot's command interface (including the arm-control blend that ramps authority from 0 - 1 at engage time).

The architecture is not tied to it, though: the same control layer can also publish over **ROS 2** / ros2_control when the wider ROS ecosystem or portability across robot platforms matters.

On the input side, the **Xsens ROS 2 driver** brings the suit data into the same ROS 2 graph, so the whole pipeline can be ROS-native.
Both ride DDS underneath, so moving between them is a transport choice rather than a rewrite; our system supports either.

> **Start here:** Unitree_sdk2_python over CycloneDDS, or Ros2_control .
> The building blocks are listed in the **Appendix.**

5 UI and safety

> **On top of the pipeline we built a web operator UI and a backend state machine: the parts that make it safe and easy to operate.**

The **UI** runs in the browser: a live 3D digital twin, camera feeds, and the operator controls, with operator, read-only observer, and Quest-3 VR views from a single codebase (laptop, booth screen, or headset, nothing to install).
New views can be added from the same codebase.

The **state machine** governs the session in the backend. It defines the allowed states and transitions (preflight, calibrate, engage, live teleop, pause, disengage, and an emergency stop reachable at any time) and hands control to the robot gradually through an engage ramp rather

commanded from a known, valid state. The state machine is not part of the UI: it lives in the backend and governs the session regardless of which interface (or how many) is attached; the UI only mirrors it.

6 Where to start

The quickest way to a working prototype is to wire the three stages together from off-the-shelf parts:

1. **Read the motion.** Enable UDP streaming in the Xsens Humanoid software and parse the incoming pose datagrams (a position and orientation per body segment). This creates a live skeleton to work from.

2. **Retarget.** Load a MuJoCo Menagerie model of your robot and set up an inverse-kinematics solve with Mink: an end-effector task per hand (and feet/pelvis if needed) that follows the operator's segments, plus a posture task to keep the result natural. Solving each frame produces joint angles.

3. **Send it.** Publish those joint angles to the robot with Unitree_sdk2_python, smoothing and rate-limiting them up to the servo rate.

That is a complete loop: operator moves, robot follows. If you'd rather not build the IK yourself, the **GMR** pipeline handles the retargeting for you: feed it the motion and it outputs robot joint targets directly.

The full, grouped list of projects for every stage is in the **Appendix.**

7 Practical tips

1

Get the loop closed end-to-end first, even badly.

A jittery robot mirroring an arm teaches you more than a perfect IK solver in isolation.

2

Invest in the safety state machine early.

Retrofitting safety onto a system that assumed success is painful.

3

Calibration and smoothing matter more than the solver.

Most "the robot looks bad" problems are reference-frame, tuning and rate issues, not IK problems.

8 Beyond motion capture: Xsens MTi

Xsens provides more than the motion-capture suit. The **MTi line** of industrial motion trackers (IMU, VRU, AHRS, and GNSS/INS modules) is widely used on the robot side of systems.
In humanoids the MTi is mostly used for **stabilization**, feeding the balance controller with high-rate, low-latency orientation and angular velocity, alongside odometry, state estimation, and localization.

The MTi line has official ROS support: the **Xsens MTi ROS node** brings MTi sensor data (orientation, angular velocity, acceleration, and GNSS) into the ROS graph.

In a ROS-native pipeline the two meet naturally: the same pipeline for the operator's motion (via the Xsens ROS 2 driver) can also be used for the robot's inertial state (via the MTi driver).

9 References

- **Xsens for humanoid robotics —** *https://www.xsens.com/humanoids*
- **Xsens XME SDK documentation —** *https://base.xsens.com/s/article/MVN-SDK-Download-and-reference-documentation*
- **Xsens GitHub —** *https://github.com/xsens*

Appendix — Reference implementations

Open-source building blocks for each stage of the pipeline.
★ *marks a sensible starting point for a newcomer.*

A1 · Motion input

Project / interface	What it is	Use it for
★ Xsens Humanoid network streaming (UDP)	Documented UDP datagram protocol from the Xsens Humanoid software	Lowest-friction live feed and .mvn playback; parse it yourself, no SDK linkage
Xsens XME SDK	Official C++/Python SDK (licensed download)	Live streaming and recorded .mvn playback, deeper device control
Xsens MVN ROS 2 Driver	Official Xsens MVN ROS 2 driver	ROS 2 nodes for streaming and URDF publishing

A2 · Retargeting

Project	Approach	Use it for
★ Mink	Differential IK on MuJoCo	The accessible on-ramp (Apache-2.0)
Pink	Differential IK on Pinocchio + QP	Same idea on the Pinocchio stack
Pinocchio	Rigid-body kinematics/dynamics	The engine under many IK stacks
GMR	General Motion Retargeting (realtime), by Yanjie Ze	Higher-fidelity, multi-robot; the retargeter behind TWIST
PHC / PHC_MJX	Learned physics-based control	Robust tracking from noisy input (research frontier)

A3 · Control loop

Project	Stack	Use it for
★ Unitree_sdk2_python	Unitree SDK2 over DDS	Direct low-latency control; the path used in this pipeline
Unitree_sdk2	Unitree SDK2 (C++)	Lowest latency, fullest API surface
Eclipse CycloneDDS	DDS middleware	The transport under SDK2 and ROS 2
Ros2_control	ROS 2 control framework	Portable control across many robots
Unitree_ros2	Unitree - ROS 2 bridge	Drive Unitree hardware from a ROS 2 graph

A4 · Simulation & models

Project	What it is	Use it for
★ MuJoCo	Physics engine + analytic Jacobians	IK substrate and a digital twin
MuJoCo Menagerie	Curated robot models	Drop-in MJCF for the Unitree G1 and others

A5 · Xsens ecosystem

All the official Xsens GitHub locations in one place:

Project	What it is	Use it for
Xsens GitHub org	Official Xsens open-source home	Entry point for all Xsens reference code
Xsens ROS 2 Driver	MVN ROS 2 driver: stream + XME nodes, URDF publisher	Motion-capture data in a ROS 2 graph
Xsens mti ROS node	Official MTi ROS node	Robot-side stabilization and IMU / GNSS-INS state estimation

Repository links verified July 2026. Owners and licenses change, so check before depending on one, especially for research code and robot models.