**Unitree G1 Overview**

**INDEX**

1. Camera & LiDAR (ros commands to launch drivers)
2. SSH Commands for Connectivity
3. ROS Versions
4. Odometry Data Source (ros topic, what odometry info we get from the robot)
5. Transforms Structure (Coordinate frames of the robot)
6. SLAM Video (trial of SLAM method, very short video\*)
7. Unitree G1 SDK & GitHub
8. Unitree RL Repo – Isaac Lab & Isaac Gym (Unitree’s Learning based approach for locomotion)
9. Additional Info
10. Control Overview of Unitree SDK Example (how Unitree controls the robot)

**Camera (Intel Realsense d435i)**

ros2 launch realsense2\_camera rs\_launch.py enable\_color:=true enable\_depth:=true

**Livox LiDAR Mid360**

~/livox\_ws$ source install/setup.bash

ros2 launch livox\_ros\_driver2 msg\_MID360\_launch.py

**ssh commands**

ssh -X unitree@192.168.124.100 --> for Wifi (.wificonnect.sh)

ssh -X unitree@192.168.123.164 --> for ethernet

**ROS Onboard**: ROS2 Foxy, ROS1 Noetic

**Odometry Data Source (Topic)**:

/odommodestate --> Position, velocity, IMU data (Orientation, acceleration, etc.)

**Transforms**:

pelvis

├── pelvis\_contour\_link     ──┐

├── left\_hip\_pitch\_link ───► left\_hip\_roll\_link ───► left\_hip\_yaw\_link

├── right\_hip\_pitch\_link ──► right\_hip\_roll\_link ──► right\_hip\_yaw\_link

├── waist\_yaw\_link ─────────► waist\_roll\_link ────► torso\_link

│                                                                                                             ├─► logo\_link

│                                                                                                             ├─► head\_link

│                                                                                                             ├─► waist\_support\_link

│                                                                                                             ├─► imu\_in\_torso

│                                                                                                             ├─► d435\_link

│                                                                                                             ├─► mid360\_link

│                                                                                                             ├─► left\_shoulder\_pitch\_link ──► left\_shoulder\_roll\_link

│                                                                                                              └─► right\_shoulder\_pitch\_link ─► right\_shoulder\_roll\_link

└── imu\_in\_pelvis

**SLAM Video (ORB SLAM3 trial on rosbag data of Unitree G1 by Me & Shalini):** [https://drive.google.com/file/d/1Ou4SMNP6buWFjMuDYDfJDBT81N5YQNke/view?usp=sharing](https://drive.google.com/file/d/1Ou4SMNP6buWFjMuDYDfJDBT81N5YQNke/view?usp=sharing%20)

**Unitree G1 online SDK:** <https://support.unitree.com/home/en/G1_developer/get_sdk>

**GitHub page:** <https://github.com/unitreerobotics>

**Unitree RL Repo’s**

**ISAAC LAB based**

<https://github.com/unitreerobotics/unitree_rl_lab>

NOT A SENSOR BASED APPROACH\*

PROPRIOCEPTIVE FEEDBACK based on:

* Joint positions and velocities
* IMU data (angular velocity, gravity direction)
* Contact force sensors on feet

**Primary RL Algorithm**:

RSL-RL (Robotic Systems Lab – Reinforcement Learning) Framework by ETH Zurich 🡪 core is PPO (Proximal Policy Optimization) algorithm -> Actor Critic architecture

File: scripts/rsl\_rl/train.py

Supports: Go2, H1, G1-29dof

**Core Files**:

source/unitree\_rl\_lab/ - Main library package

scripts/rsl\_rl/train.py - Training script

scripts/rsl\_rl/play.py - Inference/visualization

scripts/list\_envs.py - Environment listing

source/unitree\_rl\_lab/unitree\_rl\_lab/assets/robots/unitree.py - Robot definitions

Requires USD files from unitree\_model repository

**Deployment:**

deploy/robots/g1\_29dof/ - C++ controller for G1

Sim2Sim (unitree\_mujoco) → Sim2Real

**Training Approach**

* G1 learns locomotion in Isaac Lab with 4096 parallel environments, each episode lasting 20 seconds with 4x decimation (200Hz → 50Hz control).
* State Space (What G1 Observes)
* Policy observations (with 5-step history):
* base\_ang\_vel: Angular velocity (scaled 0.2x, with noise) 🡪 how fast robot torso is rotating
* projected\_gravity: Gravity direction in robot frame (with noise)
* velocity\_commands: Target velocities to achieve (x, y, yaw)
  + For training, values start with:
    - lin\_vel\_x: -0.1 to 0.1 m/s (forward/backward)
    - lin\_vel\_y: -0.1 to 0.1 m/s (left/right strafe)
    - ang\_vel\_z: -0.1 to 0.1 rad/s (turning)
* joint\_pos\_rel: Joint positions relative to default (with noise)
* joint\_vel\_rel: Joint velocities (scaled 0.05x, with noise)
* last\_action: Previous action for temporal consistency

Critic (PPO algorithm) gets additional info: base\_lin\_vel: Linear velocity (not available to deployed policy)

**Action Space**

Joint Position Control: Policy outputs joint position targets scaled by 0.25, added to default poses. Uses realistic actuator models (N7520, N5020, W4010) with torque-speed curves.

**Reward Function Design**

Primary locomotion rewards:

* + track\_lin\_vel\_xy: Track commanded X/Y velocity (weight: 1.0)
  + track\_ang\_vel\_z: Track commanded yaw rate (weight: 0.5)
  + gait: Reward proper bipedal walking pattern, 0.8s period (weight: 0.5)
  + feet\_clearance: Encourage foot lifting during swing (weight: 1.0)

Stability & efficiency penalties:

* base\_height: Maintain 0.78m height (weight: -10.0)
* flat\_orientation: Stay upright (weight: -5.0)
* energy: Minimize power consumption (weight: -2e-5)
* joint\_deviation\_\*: Keep arms/waist near default poses (varying weights)
* feet\_slide: Prevent foot slipping (weight: -0.2)
* undesired\_contacts: Avoid non-foot body contacts (weight: -1.0)

Domain Randomization

* Mass randomization: Torso mass ±1-3kg
* Friction randomization: 0.3-1.0 range
* External disturbances: Periodic velocity pushes every 5s
* Sensor noise: Added to all observations

**ISAAC GYM**

<https://github.com/unitreerobotics/unitree_rl_gym>

Worklfow: Train → Play (Isaac Gym) → Sim2Sim (test in Mujoco) → Sim2Real (deploy to hardware)

legged\_gym/scripts/train.py - Main RL training script for Isaac Gym

legged\_gym/scripts/play.py - Visualization/testing of trained policies

Support for 4 robot types: go2, g1, h1

**Deploying Scripts:**

deploy/deploy\_mujoco/deploy\_mujoco.py - Sim2Sim testing in Mujoco

deploy/deploy\_real/deploy\_real.py - Sim2Real hardware deployment

deploy/deploy\_real/cpp\_g1/ - C++ implementation for G1 real-time deployment

**Configuration Files:**

deploy/deploy\_mujoco/configs/ - Mujoco deployment configs

deploy/deploy\_real/configs/ - Hardware deployment configs (g1.yaml, h1.yaml, h1\_2.yaml)

**Models:**

deploy/pre\_train/{robot}/motion.pt - Pre-trained models for each robot

logs/{experiment\_name}/exported/policies/ - Custom trained model exports

policy\_1.pt (MLP networks)

policy\_lstm\_1.pt (RNN networks)

**Paper I referenced:**

**Humanoid Locomotion and Manipulation: Current Progress**

**and Challenges in Control, Planning, and Learning**

**https://arxiv.org/abs/2501.02116**

**UC San Diego AMO work link**: <https://amo-humanoid.github.io/>

**Additional Content**

**Actuators:**

* **Motor Type**: Low inertia, high-speed internal rotor permanent magnet synchronous motor.
* **Maximum Torque**:
  + **Knee joint**: 90 Nm (EDU model: 120 Nm).

**Degrees of Freedom (DoF):**

* **Total DoF**: 23
  + **Arms**: 5 DoF x 2
  + **Legs**: 6 DoF x 2
  + **Waist**: 1 DoF

**2 Special Joints** (https://support.unitree.com/home/en/G1\_developer/basic\_motion\_routine)

* **Waist Joint –** 3 DoF but can lock roll and pitch to make it 1 DoF (controlling only yaw)
* **Ankle Joint**
  + **PR Mode**
    - Controls Pitch (P) and Roll (R) motors – default mode
  + **AB Mode**
    - Controls A and B motors
    - Need to calculate the parallel mechanism kinematics ourselves

**A diagram of a leg and foot

AI-generated content may be incorrect.**

**Software Stack & Control:**

* **Control Type**: Force-position hybrid control for dexterous manipulation with three-finger hand.
* **Learning Methods**:
  + Imitation learning
  + Reinforcement learning
* **Framework**: UnifoLM (Unified Large Model) by Unitree
* **Compatibility**:
  + ROS 2
  + Simulation tools: Isaac Sim, MuJoCo

**Power:**

* **Battery**: 9,000 mAh lithium battery.
* **Operational Runtime**: 2 hours.

Following resource source: <https://docs.westonrobot.com/tutorial/unitree/g1_dev_guide>

**Electrical Connectivity**

A diagram of a computer

AI-generated content may be incorrect.

**Operation and Control Computing Unit (Locomotion Computer):**

* Function: Communicates with motor drivers and low-level devices, running the Unitree locomotion controller.
* User Access: Not accessible to users.

**Development Computing Unit:**

* Function: User can run custom code, including low-level (joint control) and high-level (speed control, mobile base) functions.

**Communication and APIs:**

* Middleware: Provides APIs for user interaction via CycloneDDS.
* unitree\_sdk2: Enables direct communication using DDS.
* unitree\_ros2: Integrates with ROS2, as both are DDS-compatible.

A screenshot of a computer

AI-generated content may be incorrect.

**Robotic Hands & Peripherals:**

* Third-party Add-ons: Robotic hands and peripherals (e.g., Livox Lidar) require separate driver setup.

**Disabling Locomotion Controller (Debug Mode):**

* Entering Debug Mode:
  + Suspend G1 and put it in damping state.
  + Press L2 + R2 on the remote to enter Debug Mode.
* Position Mode:
  + Press L2 + A to enter Position Mode, posing the robot in a specific diagnostic position.
* Returning to Damping State:
  + Press L2 + B to return G1 to damping state, confirming successful entry into Debug Mode.

**CONTROL OVERVIEW OF UNITREE SDK EXAMPLES**

Unitree Repo has some running examples to display on the robot, where we just run the python scripts. Following is the analyzed control aspect of one such example, basically how they control their robot. \*\*

<https://github.com/unitreerobotics/unitree_sdk2_python/blob/master/example/g1/low_level/g1_low_level_example.py>

1. **Control Framework**:
   1. Utilizes the **Unitree SDK** to interact with the robot's **locomotion computer** for real-time control.
   2. Communication is done via **DDS middleware**, enabling interaction with both low-level and high-level control systems.
2. **Control Modes**:
   1. **Pitch/Roll Mode (PR Mode):** Primarily controls rotational movements (pitch/roll) for joints like legs and hips.
   2. **A/B Mode:** Used for more complex control, especially for joints with more independent motion (e.g., wrists, fingers).
3. **Motor Command Structure**:
   1. **Enable/Disable**: Modes - 1 for enable, 0 for disable
   2. **Torque Control**: Typically set to 0 for **position control** -> achieve specific position with 0 torque (force-position control).
   3. **Position Control**: The joint positions are specified by **q** (target position), adjusted dynamically in real-time.
   4. **Velocity Control**: **dq** value is typically set to 0 for pure position control.
   5. **PD Gains**:
      1. **Kp** (Proportional): difference between the joint's current position and the desired position 🡪 larger value -> large response
      2. **Kd** (Derivative): dampen the movement to avoid overshooting the target position.
4. **Control Workflow**:
   1. **Zero Posture Initialization**: Neutral or default posture (they start with all joints at 0 radians)
   2. **Dynamic Movement**: Movements are defined by periodic (e.g., sinusoidal) motion patterns for various joints, enabling tasks like walking, swinging, or manipulation (don’t know the details\*)
   3. **Mode Switching**: Switch b/w PR and AB mode depending on the task.
5. **Communication**:
   1. **DDS Publisher**: Sends **LowCmd\_** (low-level commands) to the robot, defining motor states and movements.

For each 29 joints, following data is being published

self.low\_cmd.motor\_cmd[i].mode = 1 # Enable/disable motor

self.low\_cmd.motor\_cmd[i].q = angle # Target position (rad)

self.low\_cmd.motor\_cmd[i].dq = velocity # Target velocity (rad/s)

self.low\_cmd.motor\_cmd[i].tau = torque # Target torque (N⋅m)

self.low\_cmd.motor\_cmd[i].kp = stiffness # Position gain

self.low\_cmd.motor\_cmd[i].kd = damping # Velocity gain

self.low\_cmd.mode\_pr = Mode.PR # Ankle control mode

self.low\_cmd.mode\_machine = mode # Overall robot mode

self.low\_cmd.crc = checksum # Error detection

* 1. **DDS Subscriber**: Receives **LowState\_** (low-level state) data, providing feedback on robot’s current position, velocity, and other sensor data.

Data received:

self.low\_state.motor\_state[i].q *# Current joint angle*

self.low\_state.imu\_state.rpy *# Current body orientation*

self.low\_state.mode\_machine *# Current robot mode*

* 1. **CRC**: A **cyclic redundancy check** ensures the integrity of control messages sent to the robot.

1. **Feedback** – monitoring robot specifics (joint positions, velocity, etc.)