Integrating the Unitree D1 Robotic Arm with Obelisk for Loco-manipulation Research

Eloise Zeng, Zachary Olkin (Co-Mentor), Aaron Ames (Mentor)

***Abstract*—This project focuses on integrating the Unitree D1 robotic arm into the Caltech AMBER Lab's robotics research interface, Obelisk [1], with the goals of enabling loco-manipulation experiments on a quadruped-armed system. After extensive debugging of the Unitree D1 SDK and Obelisk, I developed a teleoperation system consisting of a controller, state estimator, and hardware interface. The controller performs inverse kinematics on the gripper, and I implemented a joystick-based interface for intuitive user control. Despite these efforts, the D1 arm failed to reliably follow commands, even after extensive troubleshooting and communication with Unitree Customer Service, ultimately leading to the conclusion that the hardware is defective. The project has pivoted toward preparation for future research in model predictive control and reinforcement learning.**

I. Introduction

The Caltech Advanced Mechanical Bipedal Experimental Robotics (AMBER) Lab focuses on developing robotic systems—such as robotic bipeds, quadrupeds, arms, and hands—that can interact with humans in everyday environments. In recent years, robotics labs around the world have purchased Unitree's robots because they are relatively cheap and robust. Furthermore, these labs can compare the performance of their software algorithms when they are deployed on the same robots. Although Unitree's robotic quadrupeds and humanoids have been popular, Unitree has also sold robotic arms with six degrees of freedom, equipped with grippers, which can be attached to the quadrupeds.

Our lab has a Unitree Go2 quadruped ($2800), the cheapest Unitree quadruped other than Go1 ($2700). One year ago, Unitree was selling the D1 robotic arm, which can be mounted onto the Go2 quadruped. The AMBER lab purchased the D1 arm around January 2024, however, Unitree no longer sold the arm by March 2025. There are no robotics papers that use the arm and only one organization unaffiliated with Unitree has posted a video of the D1 arm functioning [2]. Despite this, we believed that integrating the D1 arm with the Go2 quadruped would suffice for conducting loco-manipulation experiments.

Before conducting these experiments, we connected the arm to Obelisk and added documentation to it. Obelisk is an open-source platform created by the AMBER Lab in 2024 to simplify robot testing. Obelisk provides a unified API for interfacing with simulators and hardware, allowing the same control code to run on both. Without it, developers must write separate code for hardware integration and simulation in MuJoCo. Additionally, Obelisk offers a suite of tools—such as joystick interfacing and logging—that are easier to use than those provided by ROS2. Other labs such as ETH Zurich and the Robotics and AI Institute have developed such platforms, however, they are closed-source and customized for their research goals. Open-source platforms, like Dora (Dataflow-Oriented Robotic Architecture) [3], are relatively new and therefore unstable.

Obelisk's documentation, however, is a work in progress, making it challenging for newcomers to learn. Thus, I wrote a simple controller (where the joint positions vary sinusoidally), a state estimator, and a hardware interface for the D1 arm and added it to Obelisk. In the Obelisk Examples repository [17], I demonstrate how to use Obelisk as a library for more complex control: I implemented joystick-based gripper control and an autonomous trajectory-following demo where the arm moves its gripper to a user-specified position and orientation in Cartesian space.

The arm shakes significantly when controlled by a joystick and is unable to follow the joystick commands after a minute of execution. The arm shakes less during the autonomous demo, but it failed to execute the pose commands after a few minutes of operation. Unitree Customer Service could not resolve these issues because their D1 robotic arm group was "poached by malicious business competition." The only replacement would be the Unitree Z1 arm, which costs $15,999 and is compatible only with the Aliengo quadruped ($50,000) and the B1 quadruped ($100,000).

Therefore, I pivoted to a new project on July 30, 2025. I read several papers to better understand state-of-the-art robotics research. I learned about reinforcement learning (RL) techniques such as the advantages of PPO, privileged training [4], curriculum-based training, certain batch sizes [5], and hierarchical learning. I read how the MIT Cheetah 3 [6] (a robotic quadruped) was designed and how it differs from the ETH Zurich ANYmal quadruped.

I learned about the rewards that have been used to train quadrupeds and humanoids to traverse the wild [7] and do parkour [8] [9]. Some papers reward the robot for following the correct direction, conserving energy, and staying alive (e.g., the height of the robot is above a threshold). In addition, they train the robots on uneven terrain to compel them to raise their feet. Other papers use several more rewards, including those that encourage robots to lift their feet or imitate motion-capture videos of human beings.

I learned that while RL is more robust than MPC, MPC outperforms RL in environments where there are few valid footholds [13] [14] [15]. Therefore, some labs have experimented with combining RL with MPC [16].

My mentor proposed that I train the Unitree G1 humanoid to run using pure RL, so I followed Isaac Lab tutorials on training

manager-based RL policies. I increased the commanded base velocity in the walking base code, however, the robots were unable to learn to walk, let alone run. The reward function became quite unstable around the 6000th iteration.

II. Methods/Results

A. Hardware Set-up

The D1 arm was initially set up independently from the Go2 quadruped. The D1 arm was fastened to a piece of wood, which was clamped to a desk. The ALITOVE 24V Power Supply was purchased to power the arm. The arm was connected to the MINISFORUM UM870 Slim Mini PC via Ethernet cable. The D1 SDK was downloaded onto the Ubuntu 24.04 PC. The D1 SDK example code was debugged, so the PC can send joint position commands to the arm and receive arm feedback (i.e, the joint positions, power-on status, enable-discharge status) from the arm. The bug fixes were to set the network interface name (instructions to find the network interface name are specified in the Go2 documentation [18]) when initializing the DDS Channel Factory and to edit the arm feedback topic name to be "rt/arm_Feedback" instead of "arm_Feedback."

B. Added D1 arm functionalities to Obelisk

I was unable to set up the most recent version of Obelisk on the PC, so I set up version 3 of Obelisk and documented the issues when setting up both versions here [19].

I wrote example code to demonstrate how to use Obelisk to control the arm in simulation and real life along with visualizing the arm in Mujoco, Rviz, and Foxglove (make sure to toggle the "mesh up-axis" to be Z-up). This mainly involved writing the following files:

- Controller (Python): This sends Obelisk commands to the D1 interface. The arm initializes to its zero position, then its joint positions vary sinusoidally.
- D1 interface (C++): This converts the Obelisk commands into commands that the D1 Arm can parse. The interface converts the arm feedback messages into Obelisk feedback messages, which the estimator can parse.
- Estimator (Python and C++): This estimates the joint positions of the arm given the Obelisk feedback messages.
- YAML file: This specifies how to launch all the Obelisk nodes.
- URDF: This was provided by the D1 SDK, however, the URDF was edited to say 3.1415926535 instead of 3.14 and 1.57079632679 instead of 1.57 to facilitate computing inverse kinematics. The rotation axes were edited for two joints, so the model matched reality. Foxglove uses the URDF file to visualize the robot in its estimated state.
- XML: The URDF was converted into an XML file by following these steps [20] (which show how the Z1 Arm URDF was converted into an XML file). Then the kp and kd gains were edited to better match reality. Mujoco uses the XML file to simulate a robot obeying the controller's commands.
- Gripper utility: The URDF and XML files simulate the gripper as two prismatic joints even though it is controlled by one servo motor on the hardware. Hence, the controller publishes the control inputs in meters for the prismatic joints. The D1 arm, however, expects the gripper command to be the servomotor angle in degrees. Thus, a gripper utility file was written to convert the control inputs from meters to degrees by using the law of cosines. Other files were debugged as well.

C. Debugged how Obelisk launches nodes

I wrote code for the Obelisk Examples repository to demonstrate how to use Obelisk as a library to control the arm. I implemented inverse kinematics in the controller by using the method taught in the Caltech course ME 133a Robotics. This involved using a KinematicChain node that subscribes to the /robot_description topic to which the URDF model is published.

When I launched the stack (i.e. all the nodes), the Python Mujoco simulator node sometimes failed to transition from the unconfigured state to the active state. My mentor informed me that the node was deprecated and slower than the C++ Mujoco simulator node. I updated the Python Mujoco simulator node to raise an error if a user tries to initialize it. Then I switched to the C++ Mujoco simulator node.

When I launched the stack again, the controller node failed to transition from the unconfigured state to the active state 50% of the time. If I commented out the KinematicChain node, then the code failed only 10% of the time. Thus, my mentor suggested that I use Pinocchio, a library for computing the inverse kinematics of a robot model, instead of the method taught in ME 133a.

I reimplemented the inverse kinematics in Pinocchio, however, the controller node failed to make the transition 80% of the time. I traced the source of the error back to the Obelisk launch file. Here, a global node is initialized in its unconfigured state. Once the global node configures, it notifies all the other nodes to configure, and it begins to activate. Once the global node is in its active state, it notifies all the other nodes to activate. If the global node has activated before a node has finished configuring, then that node will fail to transition from the unconfigured state to the active state.

I modified the Obelisk launch file: If the user autostarts the stack (i.e., the user wants the nodes to configure and activate automatically, then the nodes will transition through their lifecycles independent of the state of the global node.

I likely encountered this bug because I used a computer that did not have a GPU. Perhaps, the controller node could not configure itself quickly enough since it was written in Python instead of C++.

D. Debugged how Obelisk shuts down nodes

Obelisk also failed to cleanly shut down all nodes. It would ask several nodes to shut down when they were already shutting down. Thus, I fixed the code so that almost all nodes would shut down cleanly. I could not figure out how to nicely

shut down the Mujoco node, though. Even if the stack no longer appears to run (i.e., I can type a new command in the terminal), if I type "ros2 node list," the Mujoco node is still listed. The node takes several seconds before it deactivates. If I relaunch the stack before the node has deactivated, the Mujoco node fails to change from the active state to the configured state. (This likely caused the code to fail 10% of the time when the KinematicChain code was commented out.)

E. Inverse Kinematics Algorithm

Algorithm 1 Inverse Kinematics

Require: number of iterations $i = 0$, initial joint position guess q, damping factor γ

while True **do**
  current_pose $\leftarrow$ FORWARDKINEMATICS(q) {get current gripper pose}
  $e \leftarrow$ DIFFERENCE(desired_pose, current_pose) {Compute error between desired pose and current pose}
  if $\|e\| <$ MIN_ERROR_THRESHOLD or $i \geq$ MAX_ITERATIONS **then**
    break
  end if
  $J \leftarrow$ COMPUTEJACOBIAN(q) {Get the relationship between the joint velocities and the gripper velocity}
  $v \leftarrow -J^\top (JJ^\top + \gamma I)^{-1} e$ {Get the update factor to approach the desired pose}
  $q \leftarrow q + v \cdot \delta t$ {Update the joint position guess (integrate)}
  $i \leftarrow i + 1$ {Update the number of iterations}
end while

After the while loop has ended, q should approximately be the joint positions for the gripper to reach the desired pose.

F. Attempted to smooth the robot's movements for the joystick-based controller

I implemented a joystick-based controller using the Obelisk library. The joystick commands the gripper at 10 Hz to move in the x-y-z direction, rotate about the x-y-z axes, modify its claw stroke (i.e., distance between the gripper's clamps), or reinitialize. The arm fails to achieve the commanded servo positions and does not follow any commands after a few minutes of execution. I tried to debug this by doing the following.

I wrote a simple program called initialize_arm.cpp in C++ that sends commands at 10 Hz using mode 1 to move the arm from one position to another using a joint spline. Unitree says 10 Hz is the control cycle of the arm [21]. The arm, however, moves very shakily. After a few minutes, the arm would only partially move toward the commanded servo positions. Shortly after, the arm would not respond to any commands.

I sent commands at 100 Hz and the arm appeared to move more smoothly. After a minute, however, the arm would fail in the same manner as above.

I reduced the control frequency to 10 Hz and sent commands using mode 0, which Unitree specified is for the "small smoothing of 10 Hz data," whereas mode 1 is for the "large smoothing of trajectory-use." The arm was even shakier and promptly failed.

I ssh-ed into the robot to see the execution time of the commands sent at mode 0. I edited it to be 0.1 seconds instead of 0.04 seconds. However, the arm was still shaky. I also edited the execution time to be 0.01 seconds and tried sending commands at 100 Hz with mode 0, however, the arm failed as before.

In the D1 SDK example code, there is a file titled multiple_joint_angle_control.cpp, which publishes one command containing all seven servo positions. I recorded a video alternating between running initialize_arm.cpp and multiple_joint_angle_control.cpp. After a few minutes, the arm failed. I messaged Unitree Customer Service, but the representative was not allowed to look at my code and all his questions indicated that he was unfamiliar with the D1 arm. After a week of messaging, the representative asked me to fill out a Google Form to provide feedback on the arm. Nothing he said was helpful.

I asked if Unitree was still developing the D1 arm (since Unitree no longer sells it) or if they were focusing on the Z1 arm. The representative said, "the R&D team for the robotic arm group was poached as part of malicious business competition. A new team has now taken over. It will take them some time to restore the data, as it was also deleted due to malicious business competition."

G. Attempted to send one command every thirty seconds

In the D1 SDK, there is a file called arm_zero_control.cpp, which publishes one command to move the arm to its zero position. I alternated between running arm_zero_control.cpp and multiple_joint_angle_control.cpp about ten times, and the arm did not fail.

I edited my Python controller. When the stack is launched, one can view the arm and the goal frame in Foxglove. The joystick moves the goal frame in the x-y-z direction and rotates it about the x-y-z axes of the goal frame. The joystick can control the gripper position, reinitialize the robot, and emergency-stop the robot. Once the goal frame is in the desired pose, the user can command the arm to move the gripper to this pose by hitting the joystick's SHARE button. Since the robot receives fewer commands, the user can operate the robot for a longer duration (about five minutes) before it starts failing to achieve the commanded servo positions and stops executing any commands.

H. Connecting the arm to the Go2 quadruped

At this point, I had been testing the arm on my desk using a power supply I had purchased from Amazon. Perhaps, the power supply was defective, and that is why the arm was failing. I decided to attach the arm to the Go2 quadruped since the arm was meant to be powered by the dog. However, the arm failed as before. A few servomotors were burning hot as

well. We concluded that the arm is defective and we cannot fix it.

I. Attempted to train G1 to run

To develop the running baseline, I modified the walking baseline, so the commanded base velocity ranged from -3 to 3 m/s. After 3000 iterations, all robots stood (in comparison, after 3000 iterations of training the walking policy, all robots walked). After 6000 more iterations, almost all the robots stood, but some walked diagonally or walked in place by periodically raising their feet about an inch off the ground. The mean reward became very unstable around the 6000th iteration due to the action rate reward becoming unstable.

III. Discussion

Although I discovered that the arm malfunctions six weeks into my SURF project, I still gained a lot from the experience. I learned C++ and practiced writing cleaner Python code. I learned how to use Obelisk, Mujoco, Foxglove, and Pinocchio. I improved my debugging skills by debugging Unitree and Obelisk code. I learned that it is suspicious when a robotic arm is no longer sold by a company. I got the opportunity to learn more about state-of-the-art robotics research and how to use Isaac Lab to train RL agents.

If the lab purchases another 6-DOF arm with a gripper, they could easily set it up by basing it off my code. The main difference would be the interface between Obelisk and the hardware.

Acknowledgments

I thank Prof. Aaron Ames and Zach Olkin for their mentorship, and the DaRin Butz Foundation for its financial support.

References

[1] Caltech AMBER Lab. “Documentation for Obelisk.” Github.io, 2024, caltech-amber.github.io/obelisk/index.html. Accessed 19 Aug. 2025.

[2] chen37058. “GitHub - Chen37058/Grasp-With-The-Unitree-D1: Demonstrations of Using the Unitree D1 Robotic Arm to Open Cabinet Doors and Grasp Objects.” GitHub, 2024, github.com/chen37058/Grasp-with-the-Unitree-D1. Accessed 19 Aug. 2025.

[3] dora-rs. “Dora-Rs.” Dora-Rs.ai, 2024, dora-rs.ai/. Accessed 19 Aug. 2025.

[4] Chen, Dian, et al. “Learning by Cheating.” ArXiv:1912.12294 [Cs], 27 Dec. 2019, arxiv.org/abs/1912.12294.

[5] Rudin, Nikita, et al. “Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning.” ArXiv.org, 19 Aug. 2022, arxiv.org/abs/2109.11978.

[6] Bledt, Gerado et al. ”MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot.” IEEE International Conference on Intelligent Robots and Systems, October 2018, Madrid Spain, Institute of Electrical and Electronics Engineers, January 2019. © 2018 IEEE.

[7] Miki, Takahiro, et al. “Learning Robust Perceptive Locomotion for Quadrupedal Robots in the Wild.” Science Robotics, vol. 7, no. 62, 19 Jan. 2022, https://doi.org/10.1126/scirobotics.abk2822.

[8] Hoeller, David, et al. “ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots.” ArXiv.org, 2023, arxiv.org/abs/2306.14874. Accessed 19 Aug. 2025.

[9] Zhuang, Ziwen, et al. “Humanoid Parkour Learning.” ArXiv.org, 15 June 2024, arxiv.org/abs/2406.10759.

[10] Fu, Zipeng, et al. “Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots.” ArXiv.org, 25 Oct. 2021, arxiv.org/abs/2111.01674.

[11] Zhuang, Ziwen, et al. “Robot Parkour Learning.” ArXiv.org, 11 Sept. 2023, arxiv.org/abs/2309.05665.

[12] He, Tairan, et al. “ASAP: Aligning Simulation and Real-World Physics for Learning Agile Humanoid Whole-Body Skills.” ArXiv.org, 2025, arxiv.org/abs/2502.01143.

[13] Grandia, Ruben, et al. “Perceptive Locomotion through Nonlinear Model-Predictive Control.” IEEE Transactions on Robotics, vol. 39, no. 5, 1 Oct. 2023, pp. 3402–3421, https://doi.org/10.1109/tro.2023.3275384. Accessed 18 Apr. 2024.

[14] Li, He, and Patrick M Wensing. “Cafe-Mpc: A Cascaded-Fidelity Model Predictive Control Framework with Tuning-Free Whole-Body Control.” ArXiv.org, 2024, arxiv.org/abs/2403.03995. Accessed 19 Aug. 2025.

[15] Olkin, Zachary, and Aaron D Ames. “Locomotion on Constrained Footholds via Layered Architectures and Model Predictive Control.” ArXiv.org, 2025, arxiv.org/abs/2506.09979. Accessed 19 Aug. 2025.

[16] Jenelten, Fabian, et al. “DTC: Deep Tracking Control.” Science Robotics, vol. 9, no. 86, 17 Jan. 2024, https://doi.org/10.1126/scirobotics.adh5401.

[17] Caltech-AMBER. ”GitHub - Caltech-AMBER/Obelisk_examples at D1_arm.” GitHub, 2025, https://github.com/Caltech-AMBER/obelisk_examples/tree/d1_arm. Accessed 19 Aug. 2025.

[18] HangZhou YuShu TECHNOLOGY CO.,LTD. “Quick Start.” Unitree.com, 2025, https://support.unitree.com/home/en/developer/Quick_start. Accessed 23 Aug. 2025.

[19] Eloise Zeng. “Issues Encountered While Opening Dev Container for Current Repo and Version 3 (Especially When Using Wayland instead of X11 as Display Server).” GitHub, 30 June 2025, https://github.com/Caltech-AMBER/obelisk/issues/143. Accessed 23 Aug. 2025.

[20] google-deepmind. “Google-Deepmind/Mujoco_menagerie/Unitree_z1.” GitHub, 2022, https://github.com/google-deepmind/mujoco_menagerie/tree/main/unitree_z1. Accessed 23 Aug. 2025.

[21] HangZhou YuShu TECHNOLOGY CO., LTD. “D1 Mechanical Arm Services Interface.” Unitree.com, 2025, https://support.unitree.com/home/en/developer/D1Arm_services. Accessed 23 Aug. 2025.