GENG4412 Engineering Research Project Part 2

Final Report

Design and Integration of an End-to-End Autonomy Stack: Voice, Vision, Manipulation, Locomotion, Navigation Stack on the Unitree G1 Humanoid Robot

Cameron Hoi-Shun Tsang
23149388
School of Engineering, University of Western Australia

Supervisor: Professor Thomas Bräunl
School of Engineering, University of Western Australia

Word count: 8044

School of Engineering
University of Western Australia

Submitted: 20 May 2026

Project Summary

Recent advancements in artificial intelligence have rapidly progressed the sub-disciplines that underpin humanoid robotic autonomy – computer vision, natural language processing, navigation – yet there is limited academic work regarding the integration of these sub-disciplines into a cohesively working end-to-end autonomy stack.

The objective of this thesis was to design, deploy, and test a complete end-to-end autonomy stack on the university Unitree G1 EDU humanoid that supports voice-driven communication and commands, object detection and spatial memory, autonomous navigation, dexterous manipulation, and gesture performance – operating entirely on the robot's onboard computer. The system architecture was designed to be modular and upgradable, consisting of seven integrated sub-systems built on ROS 2 and the Unitree SDK. The enabling technologies include Qwen 2.5 3B (a compact, on-board large language model), YOLOv11 (live object detection), Pinocchio (inverse kinematics), Piper TTS (neural-network-based voice synthesiser), and Unitree native SLAM (LiDAR-based mapping and navigation). The integrated stack was demonstrated through two tests consisting of (1) a pre-scripted autonomous laboratory humanoid tour guide function – comprising of the humanoid navigating to six pre-defined locations, whilst talking about the laboratory robots and including gestures to the tour presentation, and (2) a voice-driven end-to-end exercise testing each sub-system capability.

Contributions include the completed stack released as a publicly available repository for future students and the wider robotics community, tour guide functionality for the UWA Robotics and Automation Laboratory demonstration, and a documented architectural solution to ROS 2 and Vendor-SDK DDS incompatibility.

Future work recommended includes integrating the hierarchical task planner, integrating LLM-driven Q&A functionality for the scripted laboratory tour, expanding the YOLO class detection set through training, and generally updating individual sub-systems for small, but iterative system improvements.

Acknowledgements

I would like to express my gratitude towards my thesis supervisor, Professor Thomas Bräunl, for his guidance throughout this thesis and for the opportunity to take on an exciting and novel humanoid research and development project. I would also like to thank PhD student, Oliver Zhang for the many continuous discussions about research advice and humanoid work.

Finally, I would like to extend my deepest appreciation for my family, who supported me through life, university and all the challenges in between; my girlfriend Chloe – for keeping me company through the countless very late nights at the lab; and my late grandfather Chi-Ping Tsang, a UWA computer-science professor and researcher, who I would have really enjoyed working on this thesis with.

Table of Contents

Signed Declaration i
Project Summary ii
Acknowledgements iii
Table of Contents iv
List of Figures v
List of Tables v
Nomenclature vi
1. Introduction 1
2. Literature Review 2
   2.1 Current Frontier Humanoid Development 2
   2.2 Communication and Cognition for Robotics 2
   2.3 ROS 2 and SDK Middleware 2
   2.4 Vision and Object Detection Models 3
3. Project Objectives 4
   3.1 Intent-Routing Architecture 4
   3.2 Communication Subsystem 4
   3.3 Vision-Based Perception with Spatial Memory 5
   3.4 Locomotion and Navigation Subsystem 5
   3.5 Object Pick-Up Subsystem 6
   3.6 Communicative Gestures 6
   3.7 Demonstration: Scripted Lab Tour 6
   3.8 Complete Demonstration of Integrated Subsystems 6
   3.9 Benefits to Interested Parties 7
4. Design Process 7
   4.1 Design Constraints 7
   4.2 Design Approach 8
   4.3 System Architecture Overview 8
   4.4 Architectural Pattern: ROS 2 / SDK Process Isolation 9
   4.5 Key Design Decisions 10
      4.5.1 Communication: LLM Selection 10
      4.5.2 Nav2 Pivot 10
      4.5.3 Pre-Scripted Tour Orchestration 11
   4.6 Codes and Standards 11
   4.7 Safety Protocols and Safety-Informed Design Decisions 11
5. Final Design, Results and Discussion 13
   5.1 Final Integrated System Overview 13
   5.2 Subsystem Design and Results 15
      5.2.1 Communication and Cognition Pipeline 15
      5.2.2 Vision and Spatial Memory Stack 16
      5.2.3 Locomotion and Navigation 19
      5.2.4 Dexterous Manipulation Subsystem 20
      5.2.5 Gesture Subsystem 22
      5.2.6 Intent-Routing Subsystem 23

5.3 Integrated Demonstration.. 24
5.4 Discussion and Implications 25
6. Conclusions and Future Work 26
References.. 27
Appendices... 29
Appendix A: Example Title Pages.. 29

List of Figures

Figure 4.3.1: Abstraction layers and communication methodologies.. 8
Figure 5.1.1: Flow diagram of end-to-end routing for the utterance "Hey Stuart, go to the kitchen" 14
Figure 5.2.2.1: RGB frame from the RealSense D435i with YOLOv11 detection overlay and confidence — water bottle at 0.91 confidence... 14
Figure 5.2.2.2: Depth visualisation used to project the YOLO detection centroid into 3D camera coordinates 17
Figure 5.2.2.3: Unitree camera position documentation... 17
Figure 5.2.3.1: web_nav_gui map visualisation — named-marker positions (green) and live humanoid odometry (blue arrow)... 18
Figure 5.2.3.2: web_nav_gui during testing — global-frame coordinates input and target navigation.. 19
Figure 5.2.4.1: Humanoid performing vision-based pick-up — pre-grasp approach pose.. 20
Figure 5.2.4.2: Humanoid performing vision-based pick-up — grasp and lift............................ 21
Figure 5.2.5.1: joint_slider_gui_v2.py — joint control sliders, waypoint recording, playback, save and load interface 22
Figure 5.2.5.2: joint_slider_gui_v2.py — gesture playback in progress 23
Figure 5.3.1: Humanoid gesturing while communicating during the laboratory tour guide operation.. 24

List of Tables

Table 5.1.1: Autonomy stack components organised by abstraction layer, with corresponding subsystem and responsibility. 14

Nomenclature

Abbreviation	Expansion
API	Application Programming Interface
AS	Australian Standard
ASR	Automatic Speech Recognition
CLIP	Contrastive Language-Image Pre-training
COCO	Common Objects in Context (dataset)
DDS	Data Distribution Service
GPT	Generative Pre-trained Transformer
GUI	Graphical User Interface
HTN	Hierarchical Task Network
HTTP	HyperText Transfer Protocol
IK	Inverse Kinematics
ISO	International Organisation for Standardisation
JSON	JavaScript Object Notation
LiDAR	Light Detection and Ranging
LLM	Large Language Model
NN	Neural Network
PaLM-E	Pathways Language Model - Embodied
QoS	Quality of Service
R&D	Research and Development
R-CNN	Region-based Convolutional Neural Network
REP	ROS Enhancement Proposal
RGB	Red, Green, Blue (image data type)
RGB-D	Red, Green, Blue + Depth (image data type)
ROS	Robot Operating System
SAM	Segment Anything Model
SDK	Software Development Kit
SLAM	Simultaneous Localisation and Mapping
TTS	Text-to-Speech
USB	Universal Serial Bus
VLA	Vision-Language-Action (model)
YAML	YAML Ain't Markup Language (file type)
YOLO	You Only Look Once (model)

1. Introduction

Recent years have seen rapid advancements in artificial intelligence, driven by frontier research in deep neural networks and reinforcement learning. These advances directly affect robotic autonomy development by catalysing research for foundational robotics sub-disciplines including computer vision (e.g. YOLO), SLAM and navigation, and natural language processing (large language models).

Humanoid robotics R&D fall into three broad categories – private humanoid hardware companies (Figure AI, Tesla, Unitree), private AI research labs developing embodied-intelligence (NVIDIA GEAR, Google DeepMind Robotics, Physical Intelligence (pi)), and academic research work (universities, government). Private R&D typically pursue fully integrated, commercially oriented autonomy stacks, whereas academic research tends to advance individual humanoid sub-systems in isolation.

This disparity highlights a clear gap; the component technologies are well established, highly documented and publicly available for implementation in robotic systems via open-source libraries, and yet integration of these technologies for the humanoid platform are rare outside the private R&D programs.

This thesis hypothesises that the constituent technologies SLAM, navigation algorithms, conversational large language models (LLMs), text-to-speech synthesis, NN-based object detection, stereo depth and inverse kinematics – can be integrated into a cohesive end-to-end autonomy stack deployable on a humanoid. It pursues the development of a fully autonomous humanoid robot through the design and implementation of an end-to-end language comprehension, perception, autonomous navigation, and dexterous manipulation stack on the Unitree G1. Upon completion, the thesis work on the humanoid will be publicly accessible via a GitHub repository and be reproducible as a baseline for future work.

2. Literature Review

2.1 Current Frontier Humanoid Development

The Unitree G1 EDU falls under the small category of commercial research-grade platforms which allow low-level access to sensor data and actuator control through their dedicated SDK. Other humanoids in this category include the Booster T1 and Fourier GR-1. Industrial humanoids are available such as Tesla Optimus and Figure 02, but these humanoids run on propriety integrated stacks that are not publicly accessible for research. Consequently, academic research available is limited and typically focused on applied individual system functionality. For example, Radosavovic et al., (2024) deployed reinforcement learning trained locomotion to a humanoid; Chi et al., (2023) deployed a diffusion policy model for manipulation tasks on to a humanoid. The research advances individual sub-system capabilities; however, they do not address the integration challenges that arise when voice, vision, navigation, and manipulation operate concurrently.

2.2 Communication and Cognition for robotics

LLMs and VLAs that are utilised as a primary reasoning engine for robotic action can be classified as foundation-model-based robot autonomy systems (Firoozi et al., 2025).
SayCan (Ahn et al., 2022) was the first major demonstration of LLMs grounding robotic actions in natural language. Given a command, the LLM proposed high-level actions and a pre-trained value function would score its feasibility. SayCan does not scale effectively as the value function must be retrained as the action library increases. RT-2 (Brohan et al., 2023), PaLM-E (Driess et al., 2023), and VoxPoser (Huang et al., 2023) are foundation-model approaches where: RT-2 maps language to output action tokens, PaLM-E inputs robotic sensor data for context aware reasoning, and VoxPoser generates 3D value maps for classical planner execution. Each system uses compute intensive neural networks, at 55B, 528B, GPT-4 respectively, and do not fit within latency and computation constraints of this thesis. An alternative approach is taken by TidyBot (We et al., 2023), which employs a compact LLM for narrow reasoning capabilities and defers action execution to deterministic pre-coded routings. Scaling-law (Kaplan et al., 2020; Hoffman et al., 2022) also characterises the sub-3B parameter LLMs, having lower latency, less memory requirement, and still has functional language ability, with the trade-off being reductions in complex reasoning, long context coherence, breadth of knowledge for smaller. It can be gathered from these papers that, for a scenario which primarily utilises the LLM for simple tasks and conversations and constrained in compute, the compact LLM would work effectively.

2.3 ROS 2 and SDK middleware

ROS 2 is the industry standard for autonomous robotics middleware development, supporting publish and subscribe structured messaging services over DDS transports. (Macenski et al., 2022). It is a published issue (Unitree Robotics, 2024), that when ROS 2 is run within the same process as the vendor SDKs that rely on DDS communication– such as Unitree SDK – the ROS 2 participant configuration can become corrupted. Mitigation strategies have been examined (Maruyama et al., 2016) such as QoS configuration and DDS tuning, but there is no documented fix.

2.4 Vision and Object Detection Models

The YOLO family object detection models (Joher et al., 2023) utilise a single-stage architecture, effective for real-time inference on edge hardware. However, it is limited by its closed set detection with 80 COCO classes. Two-stage detection systems exist, such as Faster R-CNN (Ren et al., 2015) and DETR (Carion et al., 2020), which is transformer based. The two-stage detection systems have higher detection accuracy but require more compute and have increased latency when compared to single-stage systems.

Persistent and dynamic object-level spatial memory is an active field of research. ConceptGraph (Gu et al.,, 2023) constructs open vocabulary 3D graph fusing CLIP features. ConceptFusion (Jatavallabhula et al.,, 2023) creates dense open-set feature maps in 3D, and Hydra (Hughes et al.,, 2022) uses hierarchical closed-set graphs for rooms. These architectures support LLM queries with highly connected features, but they all require heavy computation as they rely on neural networks such as CLIP, SAM or dense semantic segmentation.

The current literature for humanoid robotics primarily focuses on individual capabilities in isolation; the gap addressed in this thesis is the integration and deployment of a single, autonomous voice, vision, manipulation, locomotion and navigation stack on the university‘s G1 humanoid, while also constrained by local compute and with safety at the forefront of testing.

3. Project Objectives

This thesis is fundamentally an integration project, delivering a complete humanoid autonomy stack across voice, vision, manipulation, locomotion, and navigation sub-systems. Given the breadth of sub-systems integrated within a single-student, two-semester scope, a qualitative evaluation approach was adopted for objective acceptance over quantitative metrics. An objective is considered successfully met when two conditions are satisfied: (1) all listed design features are implemented, and (2) the system demonstrates consistent and repeated functionality with success. Additional performance metrics are specified where applicable.

3.1 Intent-routing architecture

The intent-routing architecture is the basis of the humanoid voice-driven autonomy. It parses incoming speech into executable intents and dispatches the intent to the relevant sub-system – navigation, manipulation, gesture, or conversational – through topic publishing.

Design features:
- Deterministic regex-based intent classification
- LLM-based parameter extraction (objects and location entities)
- TaskIntent message publishing
- Routing to subsystem action interfaces
- Non-blocking conversational fallback

Additional performance metrics:
- Accurate intent classification across a varied set of test inputs
- Dispatch latency below 5s
- Low false-positive rate on non-command input

3.2 Communication subsystem

The communication sub-system primarily utilises an LLM capable of dynamically handling diverse conversational input, and implements the following design features:

Design features:
- Wake-word detection
- Onboard ASR
- LLM response generation
- Short-term dialogue memory
- Vision-context injection
- Static laboratory information injection
- Text-to-speech and speaker system interfacing

Additional performance metrics:
- End-to-end conversational latency below four seconds

- Wake-word activation reliability of over 90% across various test inputs
- Accurate vision-context recall
- Demonstration of contextual memory across spoken input
- Successful integration with the gesture subsystem.

3.3 Vision-based perception with spatial memory

The vision-based perception sub-system provides the humanoid with 3D environmental awareness through the detection and localisation of objects in real time whilst also maintaining a persistent but decaying spatial memory of their positions.

Design features:

- Live RGB-D frame streaming
- NN-based object detection
- Depth projection and object-depth association
- Camera-to-torso frame transformation
- Torso-to-global frame transformation
- 3D object position estimation and spatial marker management
- Spatial memory retention and decay
- Detection information published for downstream sub-systems

Additional performance metrics:

- Position error less than 5cm for torso-frame detections
- Consistent detections across multiple objects

3.4 Locomotion and navigation sub-system

Voice-triggered mobility within a pre-mapped environment, supporting the design features below.

Design features:

- Named-marker navigation (kitchen, bathroom, etc.)
- Global-frame coordinate navigation (x, y, yaw)
- Body-frame relative motion (forward, back, turn)
- Locomotion with dynamic obstacle avoidance
- Arrival detection
- Completion-handoff TTS

Additional performance metrics:

- Positional accuracy within 0.5m of the goal
- Yaw accuracy with 20 degrees of target bearing
- Consistent obstacle avoidance and navigation success

3.5 Object pick-up

Voice-triggered detection, localisation and dexterous manipulation of a named object.

Design features:

- Object localisation via perception sub-system
- IK-based reach planning
- Low-level motor control for arm and hand movement
- Intermediate arm poses for arm trajectory execution
- Dexterous hand control (grasp and release)

3.6 Communicative gestures

Pre-defined arm gestures performed alongside conversational responses, supporting design features below:

Design features:

- Pre-defined gesture library (Unitree SDK)
- Record and playback functionality for new gestures
- LLM dispatch for gesture-inclusive dialogue
- Concurrent gesture execution with TTS

3.7 Demonstration: scripted lab tour

A scripted six-station laboratory tour, validating end-to-end integration of locomotion, navigation, communication, gesture, and intent-routing sub-systems.

Acceptance criteria:

- Sequence of six pre-defined waypoints (representing laboratory stations)
- Pre-made scripts and pre-defined gestures for each station
- Autonomous navigation between stations with scripted speech and gestures
- No operator intervention required.

3.8 Complete demonstration of integrated sub-systems

Voice-invoked operation of each subsystem objective, demonstrating complete autonomous functionality and validating robustness of all subsystems within the end-to-end stack architecture.

Design features:

- Voice-invoked navigation to multiple named locations
- Humanoid conversation with integrated gestures when necessary
- Voice-invoked dexterous pick-up of a named object
- Stable system across consecutive voice commands without user intervention

3.9 Benefits to interested parties

This thesis contributes directly to the UWA Robotics and Automation Laboratory by providing a stable baseline for future ROS 2-based sensor integration and conversational interaction on the Unitree G1 EDU. The Laboratory will additionally use the humanoid tour-guide functionality as a public-facing tool to showcase the University's humanoid development and the broader robotics projects undertaken within the lab. The stack is transferable to other commercial humanoid platforms and remains available for extension by future students. The full implementation will be released publicly via GitHub.

4. Design Process

4.1 Design Constraints

The design and architecture of the system was developed under fixed hardware, software and time constraints. These constraints directed the engineering decisions and bounded the design space as described throughout Section 4.

The Unitree G1 humanoid robot has 43 degrees of freedom including the 3-fingered dexterous hand. The humanoid has a weight of 35kg, peak knee torque of 120N.m, and have documented reports of unexpected, dangerous movement during testing. A two-student minimum testing protocol was implemented and a fall arrest gantry that was utilised for deployment trials.

There are two onboard computers – a NVIDIA Jetson Orin 16GB for deploying and running code and a DDS based motion controller. Realtime onboard execution of the software stack was required as an implicit safety constraint. Wireless data transmission latency was slow and unreliable, thus incompatible with live data-dependent control loops required for safe locomotion and manipulation. Robot perception was bounded by the fixed, onboard sensor array including the Livox MID-360 LiDAR, Intel RealSense D435i stereo depth camera, 4-microphone array, and 5W speaker.

The development of the stack was constrained by the requirement of both Unitree SDK and ROS 2 as middleware. ROS 2 is an industry-standard development middleware for robotics research and provides the necessary infrastructure for sensor-driven, autonomous robotics stack. The Unitree SDK was important as it had access to the low-level controllers and unitree-based sub-systems these include Unitree SLAM and locomotion, Unitree gesture presets, speaker audio, and motor control of arm and dex3 hands. Additionally, ROS 2 and the Unitree SDK both rely on CycloneDDS for robot interfacing and commands but cannot share processes without middleware corruption.

4.2 Design Approach

Design choices throughout this thesis were driven by three priorities: Implementing every design feature described in Section 3, complete demonstration of humanoid capability objectives through the end-to-end autonomy stack deployment within the two-semester time constraint and operating within the hardware and middleware constraints described in Section 4.1. Where multiple implementations were available – the option offering the best balance of latency, modularity, and on device feasibility was selected. Additionally, well-established libraries were leveraged where available, in preference to reimplementation.

4.3 System Architecture Overview

The system architecture comprises 15 ROS 2 nodes and 4 Python daemons structured into four layers. Each layer has a specific role within the stack, and the communication between layers is restricted to three specific channels, detailed below.

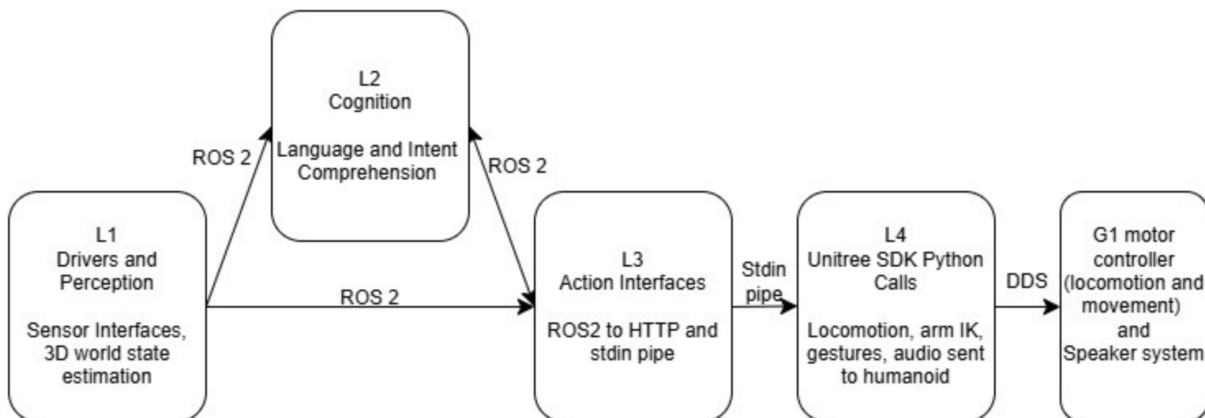

Figure 4.3.1. Abstraction Layers and Communication Methodologies.

The L1 (Drivers and Perception) layer handles physical sensor data and translates it into structured information for other ROS 2 nodes to utilise. To illustrate, camera_node (L1) consumes the USB camera stream and publishes the raw image and depth data; yolo_node (L1) takes the raw image feed and produces object detections and 2D bounding box.

L2 Cognition handles language comprehension and high-level intent-based decision-making. To illustrate, a local LLM in llm_node.py (L2) is utilised to determine intent from speech and to generate humanoid response. Router_node.py initiates the downstream nodes' actions dependent on intent.

The L3 (Action Interfaces) layer translates the upstream ROS 2 commands into stdin pipe commands and HTTP calls, and handles relaying status information back to ROS 2 via topics. As an example, web_nav_gui runs a live, interactive visualisation of humanoid locomotion on a port 8081 web server and sends movement commands to (L4) slam_client through stdin pipeline.

The L4 (SDK Isolation Daemon) layer interfaces with the humanoid through the Unitree SDK via python processes outside ROS 2. The python processes are invoked by (L3) ROS 2 nodes through stdin pipe. For instance, tts_daemon takes lines of text through stdin pipe and produces audio output through the onboard speakers. As a secondary example, pick_daemon receives stdin commands from (L3) pick_node.py and translates command into humanoid arm motion.

Communication between nodes and daemons utilises three channels: ROS 2 publish/subscribe topics connect L1, L2, and L3; OS-based stdin/stdout pipes connect L3 to L4; and L4 to humanoid output connection through Unitree SDK via DDS. The independent communication between layers allow for independent testing for each layer -> simulated input and output verification.

4.4 Architectural Pattern: ROS 2 / SDK Process Isolation

The mechanism for humanoid interfacing is through DDS communication; where both ROS 2 and Unitree SDK depend on CycloneDDS. When run in separate processes, ROS 2 and Unitree SDK function as intended, but when collocated in the same process an SDK call corrupts the ROS 2 middleware. Upon initialisation the SDK overwrites the participant configuration originally set by ROS 2, but ROS 2’s internal participant records do not get refreshed upon overwrite. The misconfiguration leads to a range of issues including topic communication failure and service calls becoming unresponsive. This incompatibility is acknowledged within the Unitree documentation (Unitree Robotics, 2024).

Migration strategies were explored to work around the issue. DDS has isolation mechanisms such as domain ID, namespace and QoS profile configuration to allow for process separation. Domain ID adjustment allows for separate network traffic but does not eliminate memory corruption. Namespaces control ROS 2 client-side naming for node, service, and action names, but do not affect the communication CycloneDDS. The QoS configuration affects participant behaviour once registered, but the participant records are the corrupted state of the CycloneDDS collocation issue, so QoS reconfiguration is ineffective. These limitations of DDS isolation under a shared single process are consistent with the experimental observations of Maruyama et al., (2016).

Given there was no structural fix to the ROS 2/ SDK incompatibility; the alternative approach was to have the ROS 2 node spawn a separate child process in a clean virtual environment which ran the unitree SDK operations. The daemon allows CycloneDDS to load into its own virtual memory, isolated from the ROS 2 CycloneDDS memory. Communication between daemons and nodes were through the OS based, stdin/stdout pipeline, which does not route through middleware affecting DDS. Communication between all daemons and nodes followed the same structure; upon initialisation the daemon sent READY (stdout), node sends a one-line command (stdin), and confirmation msg by daemon is sent back (stdout). The node tracks the state of the daemon. The communication pattern is consistent between tts_daemon, gesture_daemon, pick_daemon, and slam_client.
This OS-level pipe solution is not specific to the Unitree G1 hardware and can be utilised for any robotics platform whose vendor SDK creates conflicts with the ROS 2 DDS collation. No publicly documented implementation of this pattern on the G1 was identified during this work. This thesis contributes one through its open-source release, providing a baseline architecture for further development.

This solution can be utilised across robotics platforms with ROS2, vendor SDK incompatibility – as the solution isolates both SDK and ROS 2, with only OS level communication.

4.5 Key Design Decisions

4.5.1 Communication: LLM selection

As described in Section 3.2, the communication system handles command recognition, dynamic conversation responses, contextual injection and gesture association with responses, and must perform these tasks under the computation constraint and a latency budget of approximately four seconds. Foundation-model approaches surveyed in recent embodied-AI robotics literature (Ahn et al.,, 2022; Brohan et al.,, 2023; Driess et al.,, 2023) require compute infrastructure that exceeds the capabilities of the onboard Jetson. Small-parameter models including Llama 3.2 3B, Phi-3.5 mini, DeepSeek-R1 1.7B and Qwen 2.5 3B were evaluated. To minimise latency and computation requirements, two design decisions were made – (1) Qwen 2.5 3B at Q4_K_M quantisation was chosen as the onboard LLM as the sub-3B parameter class was identified in literature (Kaplan et al., 2020) to have latency-quality balance for on-board computation. (2) a three-stage filter was implemented to handle low-latency intent routing comprised of (i) a deterministic regex-based intent classification filter, (ii) a Qwen inference with context injection filter, and (iii) a response classification and gesture striping filter. The three stage filter and compact LLM architecture follows the precedent established by TidyBot (Wu et al., 2023), which satisfies the on-device compute limitation, latency budget, and modularity design approach described in Section 4.2.

4.5.2 Nav 2 pivot

Nav2 is an industry standard for deploying navigation systems across robotic platforms that run ROS 2 (Macenski et al., 2022) . The Nav2 system has built-in functionality for global planning, layered cost mapping, and dynamic navigation. It integrates data sent from SLAM toolbox for mapping. Utilising Nav2 and SLAM toolbox as premade, industry standard integration material, was a design criterion described in Section 4.2. A cmd_vel bridge node was created specifically for translating the twist commands from nav2 to Unitree commands. The intended dataflow was: SLAM toolbox -> Nav2 planner -> cmd_vel_bridge -> humanoid locomotion.
There were three compounding failures that occurred and forced a system redesign away from the Nav2 implementation. The MID360 LiDAR scans in 3D, the problem is that it captures the floor as part of the scan, and Nav2 associates the floor scan as an obstacle, updating the cost map and marking goals unreachable. The velocities sent through the cmd_vel_bridge were below 0.025m/s and the unitree locomotion controller requires a minimum of 0.3m/s before movement occurs, so no motion was produced. The behaviour tree plugins are incomplete within the ROS 2 Foxy distribution, which limits the features of Nav2.

The design decision was made to pivot to the onboard Unitree SLAM and navigation, controlled via the DDS API SDK. The unitree navigation and locomotion included velocity commanding, navigation and obstacle awareness, which suited the scope of the thesis. As it was developed specifically for the robot, there was documentation available specifically for interfacing with the

client. The daemon web_nav_gui was developed to create a live interactable visualisation, allowing click-to-walk on LiDAR map, and persistent marker placement (to be navigated to).

The pivot from Nav2 meant losing spin-to-replan functionality, escaping local minima, and the general purpose Nav2 obstacle avoidance. However, the unitree-based locomotion, navigation, and SLAM generation and localisation is highly reliable, functional as the motion controller within the communication and intent stack. Given the benefits of the unitree solution, the trade-off is reasonable and the limitations that may come with the pivot away from Nav2 is accepted as and highlighted within the 5.4 (limitations)

4.5.3 Pre-scripted tour orchestration

The tour orchestration consists of a standalone ROS 2 node and a .yaml configuration file containing the predefined script, timing, gestures, and locations. Zero modifications were made to any other file for this implementation. It parses the script to audio_responder through /llm/response, including text with gesture tags. Navigation utilised predefined markers within the Unitree SLAM map.

The tour implementation is a pre-scripted YAML state machine, instead of an LLM-driven conversation. This design decision was made due to multiple reasons. First, the demo had to be consistent and reliable – the use of the LLM could pick up chatter from audience members. Additionally, LLM outputs are non-deterministic, especially if the Unitree ASR transcribes a question incorrectly. The tour specifications are relatively fixed, and so generating the perfect timing between the speech, walking and gesturing can be tuned to become very smooth. There is LLM-based Q&A functionality able to be implemented as a future implementation. The trade off for the design decision is less dynamic content for a more reliable and consistent demo.

4.6 Codes and standards

Australian standards do not explicitly have standards that govern for humanoid robotics testing – the closest relevant standard is AS4024 (Industrial Machinery). The stack development conforms to REP-103 ROS 2 conventions (Foote & Purvis, 2010) – standard for units of measure and coordinate conventions, and standard ROS 2 topic naming conventions – (std_msg, geometry_msgs, nav_msgs, sensor_msgs).

4.7 Safety protocols and safety informed designed decisions

Human safety was prioritised, informed design decisions, and safety protocol implementation, where live humanoid testing was determined as the significant safety risk. Two scenarios were determined where the humanoid functionality was considered non-deterministic and unpredictable – (1) Concurrent autonomous humanoid locomotion and failure of object avoidance mechanisms, and (2) misinformed dexterous manipulation (e.g. IK request from vision sub-system.)

Risk mitigation was implemented at two levels of the ISO 12100 hierarchy: engineering controls through two independent abort mechanisms (as described below), and administrative controls through a two-person testing protocol assigning each operator an abort strategy.
One individual would be responsible of software-based initialisation, testing, and controlled abort functionality - process termination that would stop all DDS humanoid communication and where the humanoid control system would defer to controller-only motion. This is the safe-exit methodology because the default position for controller-only humanoid motion is a maintained stationary up-right position. The other individual is responsible for the hard-abort controller-based e-stop – directly deenergises all motors where humanoid immediately crumples to the ground, unless suspended from the fall-arrest gantry. An additional safety protocol was utilised, where the humanoid was suspended from the fall-arrest gantry for all practically viable tests – e.g. stationary dexterous manipulation.

A safety-informed design decision was implemented to mitigate the unintended and unsafe arm motion, during misinformed dexterous manipulation. A software-based engineering control was integrated where the IK-based target arm pose was published and required a user confirmation before arm motion execution.

5. Final Design, Results and Discussion

5.1 Final Integrated System Overview

This thesis delivers a complete and functional end-to-end autonomy stack on the Unitree G1, where the humanoid can perform complex communication and command-invoked vision-manipulation tasks, locomotion and navigation tasks, and communication-associated gestures. The system architecture extends the classical three-layered abstraction architecture for robotics established by Gat (1998) through the addition of the SDK isolation layer. The SDK isolation layer mitigates the CycloneDDS conflict described in Section 4.4. Each layer has a distinct system responsibility and each operates at separate levels of abstraction. The software stack is built upon ROS 2 and Unitree. ROS 2 hosts the primary processing logic and inter-node communication – vision (YOLO), language comprehension (Qwen 2.5 3B), intent-routing, and decision-making implemented as nodes. The Unitree SDK handles low-level humanoid interfacing – sensors, actuators, and Unitree-specific services. The stack comprises of seven integrated sub-systems: communication, vision, navigation, manipulation, gesture, intent-dispatch, and tour guide.

The fully operational autonomy stack comprises of fifteen ROS 2 nodes and four Python daemons totalling over 4,500 lines of Python code. Auxiliary files include two YAML configuration files (tour guide script and lab-context injection), two JSON configuration files (gesture library and navigation marker library), and two Flask-based GUI files (joint control tool for gesture recording and a runtime visualisation-and-navigation interface). Additionally, a standalone hierarchical task planner C++ file to be deployed and integrated in the stack for future use. All the nodes and daemons run locally onboard the NVIDIA Jetson Orin.

To illustrate end-to-end stack functionality, consider the following input speech "Hey Stuart, go to the kitchen". The onboard microphone receives audio and transcribes audio through the Unitree ASR, publishing the text to */audio_msg*. *audio_responder* validates the "Hey Stuart" wake-phrase and publishes the cleaned utterance to */llm/input*. *llm_node* regex matches the speech pattern to *navigate_to_marker* intent, extracts the target "kitchen" and publishes a *TaskIntent* message to */llm/task_intent*. *router_node* receives task intent and produces a HTTP POST to *web_nav_gui* with the target "kitchen". *web_nav_gui* forwards the goal to *slam_client* via stdin pipeline. *slam_client* then executes locomotion navigation to the target coordinates through the Unitree SLAM service, and the humanoid walks to "kitchen" marker goal. The end-to-end dataflow traverses the four abstraction layers and all communication channels – ROS 2 topics, HTTP and stdin/stdout pipes.

Additional video recordings will be published and hosted on camtsang.com as well as the GitHub repository – and demonstrate the complete functionality of the autonomy stack, as well as individual sub-system operations.

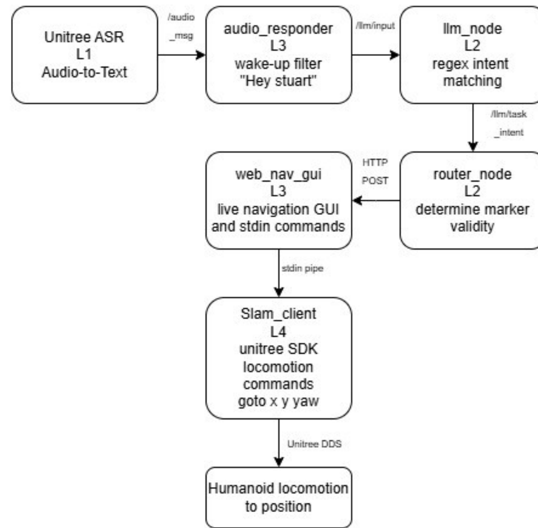

Figure 5.1.1. Flow diagram describing end-to-end routing for example "Hey Stuart, go to the kitchen"

Layer	Component	Subsystem	Responsibility
L1: Drivers and Perception	camera_node	Vision	USB camera feed to ROS topics
	yolo_node	Vision	Real-time YOLO object detection
	depth_projector_node	Vision	Torso-frame and world-frame transformations
L2: Cognition	llm_node	Communication	Qwen 2.5 inference – regex intent classification, context injection, gesture association
	router_node	Intent-dispatch	Intent routing to action interfaces
	spatial_memory_node	Vision	3D object position memory
	tour_guide_node	Tour guide	Tour guide state machine and orchestration
L3: Action Interfaces	audio_responder_node	Communication	Wake-word filtering, dispatching TTS, gesture routing
	pick_node	Manipulation	Pick sequence controller
	web_nav_gui	Navigation	HTTP endpoint for navigation, map, pose and marker visualisation
L4: SDK daemons	tts_daemon	Communication	Piper TTS synthesis, audio playback via SDK
	gesture_daemon	Gesture	Arm gesture execution via SDK
	pick_daemon	Manipulation	Pinocchio IK, arm trajectory generation, Dex3-1 hand control
	loco_daemon	Navigation	Walking and pose control via SDK

Table 5.1.1. Autonomy stack components organised by abstraction layer, with corresponding subsystem and responsibility.

5.2 Subsystem Design and Results

5.2.1 Communication and Cognition Pipeline

The communication sub-system implements the design features as defined in Section 3.2, through a five-stage pipeline, as described in Figure 5.2.1: Onboard Unitree ASR, Wake-phrase filtering (audio_responder), three-stage cognition filtering (llm_node), Piper TTS audio synthesis (tts_daemon) with gesture dispatch (gesture_daemon).

Audio capture and transcription is handled by Unitree ASR and continuously streams completed utterances to /audio_msg. audio_responder subscribes to /audio_msg and text matches every incoming utterance against a set of 'Hey Stuart' variations. The set contains spelling variations of Stuart, such as 'Stuwart', 'Stewart', and 'Swart' and is implemented due to common ASR transcription errors and phonetic confusion for the name. After the wake-phrase match, audio_responder strips 'Hey Stuart' and publishes the cleaned message to /llm/input - the reduction in input words, reduces the LLM compute amount, thus lower conversational latency.

The llm_node passes the utterance through a three-stage filter chain. The Stage 1 filter is a regex-based intent classifier, where the transcript is matched against eight regex patterns covering all command options (navigate_to_marker, move_relative, turn_by_angle, turn_around, pick_up, gesture, stop, find). Additionally, command-specific parameters are extracted from the transcript via regex. A TaskIntent message is built from the confirmed regex matches and is published to /llm/task_intent. A confirmation message containing TaskIntent information is compiled and dispatched to /llm/response (E.g. "Navigating to Kitchen"). Unmatched utterances flow to filter two.
Stage two filter constructs an LLM prompt combining five context sources (static system prompt, static lab knowledge from dedicated YAML, keyword-triggered vision-context, twenty message conversation history, and the current utterance), submits it to the LLM, and produces a response which may include response-appropriate gesture tags.
The static system prompt is the primary context window for the LLM and is the baseline for every call. It contains the humanoid identity grounding, "You are Stuart, a friendly humanoid assistant", a behaviour rule set such as, "keep responses under three sentences", robotics lab information and facts– such as the current research projects and student robotics projects ongoing, gestures in the gesture library and the gesture tag format. Robotics laboratory information context are in a dedicated YAML file that is always injected in the static system prompt. This allows simple and non-invasive revisions that can be implemented in the YAML instead of the node Python file.
The vision context injection is triggered when the user message is matched against a set of key words used for vision responses. Some of the key word/phrases are, "looking at", "what's there", "what do you see". If the pattern matches, the filter 2 will read the latest /object_detections from yolo_node and add it to the inject it into the prompt. If the user message does not match the vision key words, injection is skipped.

Stage 3 filter is a deterministic gesture emission filter which takes the LLM generated response, applies substring matching and compares the response with a set of key-word triggers which imply a social connotation. If no social connotation is implied, the response is stripped of the gesture tags.

The final response is published to /llm/response, where audio_responder removes gesture tags from the response (if it exists) and dispatches the gestures to gesture_daemon. The speech text is sent to tts_daemon for Piper-based vocal synthesis, outputted on the humanoid speakers.

For testing, the pipeline operated as designed. Wake-phrase detection was reliable, intent classification through the regex filtering functioned – correctly classifying and associating the eight commands with no misclassifications observed during testing. Fuzzy substring matching against marker names within the marker library resolved accurately for navigation commands (e.g. "the bathroom" resolved to bathroom marker). End-to-end latency from the end-of-user speech to start-of-humanoid speech fell within the 4 second budget – with 2-3 second latency for regex classified commands, and 3-4 second latency for conversational speech. Vision-context conversation was successful, injecting live /object_detections data into the LLM prompt with humanoid speech accurately describing objects in-view. Recollection of conversation history context, spanning twenty previous messages, functioned appropriately. Response-appropriate gestures executed concurrently with humanoid speech.

Two failure modes were recorded: (1) Environments with continuous background noise reduced the effectiveness of Unitree ASR and in rare instances, wake-phrase transcriptions fit outside the set of valid "Stuart" spelling variations, resulting in missed activations. (2) Qwen occasionally incorporated an unrelated gesture to the response. Although these invalid gestures were consistently filtered out by the Stage 3 filter, the behaviour highlights the limitations of compact-LLM intelligence.

5.2.2 Vision and Spatial Memory Stack

The vision and spatial memory sub-system implements the design features as defined in Section 3.3, through four ROS 2 nodes: camera_node, yolo_node, depth_projector_node, spatial_memory_node.

The camera_node receives data from the USB camera via the RealSense Python SDK, and streams live RGB image (approximately at 17Hz) and depth image (approximately at 5Hz) data feed to the ROS 2 topics camera/color/image_raw and camera/depth/image_raw.

The yolo_node loads the YOLOv11n model upon initialisation, subscribes to the camera feed data topic camera/color/image_raw, and feeds the colour image into the inference model. The inference model outputs object detections, which consist of an object label string, a bounding box (x, y, width, height) and confidence value between 0 and 1. The yolo_node filters detections by ignoring detections below a 0.75 confidence value and passes the detections above the threshold to the /detections topic.

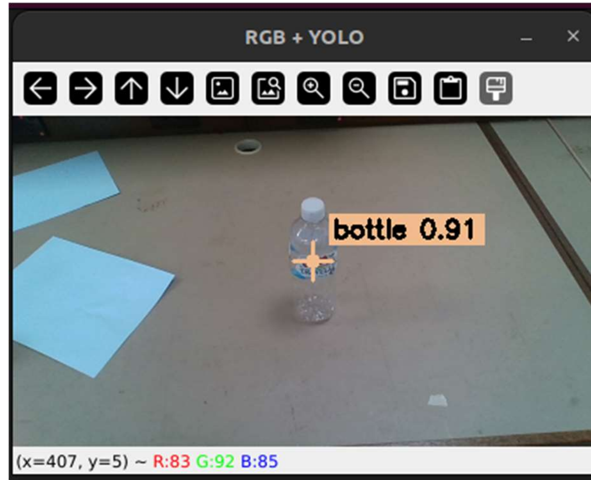

Figure 5.2.2.1. RGB frame from Realsense D435i with YOLOv11 detection overlay and confidence – Detecting a water bottle at 0.91 confidence.

The depth_projector_node projects the bounding box of the relevant detections over the live depth image, and samples the median depth value within the bounding box. The median depth value is taken as the 3D coordinate should be as central as possible to ensure the object's depth value is taken, as there is noise around the bounds of the bounding box.

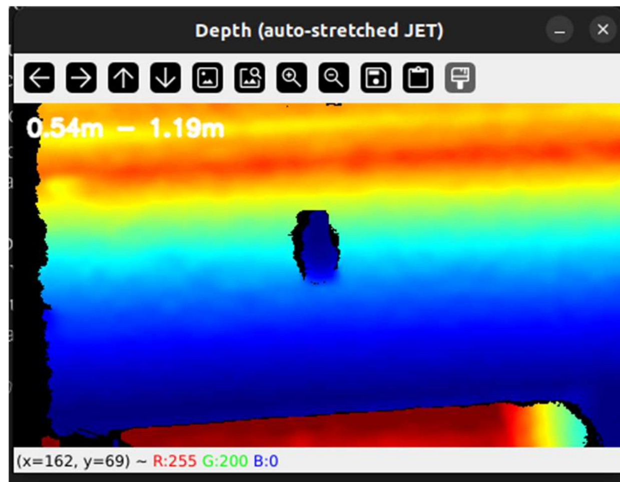

Figure 5.2.2.2. Depth Visualisation used to project the YOLO detection centroid into the 3D camera coordinates.

The resulting (x, y, depth) values are converted to 3D camera-frame coordinates by applying a transformation matrix based on the camera optical specifications.

$$\begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\sin\theta & \cos\theta \\ 0 & -\cos\theta & -\sin\theta \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} + \begin{bmatrix} 0 \\ d \\ h \end{bmatrix}$$

Equation 5.2.2.1. Camera-to-torso transformation – with θ = 48̊, d = 0.04764m, h= 0.46268m – as per the Unitree documentation.

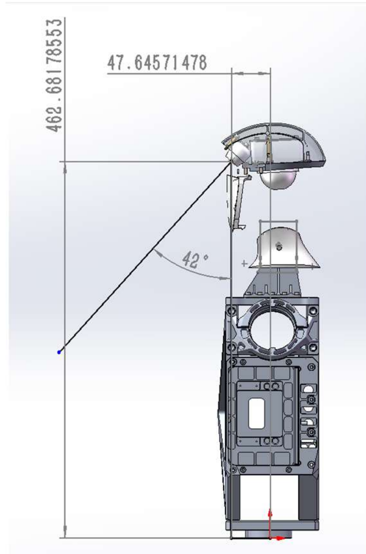

Figure 5.2.2.3. Unitree camera position documentation. (Unitree Robotics, 2024)

The Unitree documentation provides detailed schematic of the camera's location and orientation relative to the torso as shown in Figure 5.2.2.3. The camera-frame coordinates are transformed into torso-frame coordinates, by applying a transformation matrix based on the Unitree camera-mount documentation. The torso-frame object coordinates are published to /object_detections_body, and an additional transformation - based on live LiDAR odometry from /unitree/slam_relocation/odom - converts the coordinates into world_frame coordinates and these coordinates are published to /object_detections

spatial_memory_node maintains an object dictionary containing the object's persistent 3D global coordinates, with decaying and updatable confidence functionality. Live detections in /object_detections are checked against the existing objects within the library; if the detection is of the same object type and has a distance under 0.5m, the internal object has its position updated. The object positional update tracks the current detected position and slowly moves the object in memory to the new position with a smoothening function. If the objects in the dictionary have not been viewed for an arbitrary 30 seconds, the confidence values start decaying at a rate of 0.02 per second. Once an object in the dictionary falls below 0.05 confidence, it is removed. The spatial_memory_node publishes to two topics: /spatial_memory/summary – a JSON formatted summary highlighting all objects currently in the dictionary, with 3D global coordinate positions and confidence values – and /object_markers/<label>, which provides individual marker location for individual objects.

For testing, position estimation in the torso frame was reliable with around 2cm of error – well within the 5cm acceptance target described in Section 3.3. Detection consistency was maintained across multiple object classifications, and spatial object memory was achieved with global-frame 3D object marker persistence and decay.

Three failure modes have been documented: (1) YOLOv11n is closed set detection with only 80 detectable objects, and thus additional training is required for more detectable object classes, (2) when two objects of the same class are present within 0.5m, their positions can merge due to the

reassociation functionality, and (3) transparent or reflective objects can cause inaccurate depth measurements.

5.2.3 Locomotion and Navigation

The navigation and locomotion sub-system is built upon Unitree SLAM relocalisation, SLAM mapping, and locomotion services. The stack contribution by this thesis comprises of map generation, the web_nav_gui interactive map GUI interface, the slam_client daemon, the intent dispatch via router_node, and llm_node regex-based intent coordination.

Before SLAM based navigation occurred, the map was generated with Unitree's onboard SLAM mapping service - where controller-based locomotion moved the robot around the space while the mapping service accumulated LiDAR and odometry data. Once mapping is completed, the .pcd file is saved and called when initialising the unitree SLAM relocation. slam_client is a Unitree-SDK daemon that communicates with web_nav_gui via the stdin/stdout pipeline. web_nav_gui hosts a Flask web server on port 8081 with a GUI displaying the loaded LiDAR map, live humanoid position and bearing, and location markers stored in named_markers JSON file. web_nav_gui allows marker management (add, rename, delete, navigate to), and click-to-walk functionality - clicking a space on the map and setting a goal bearing. Additionally, web_nav_gui allows voice-driven navigation and locomotion by exposing its HTTP endpoint that router_node posts to.

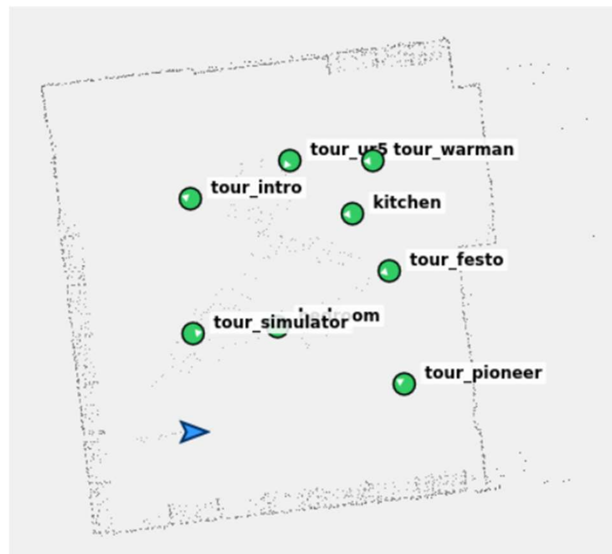

Figure 5.2.3.1. web_nav_gui map visualisation: Green points are named markers position and bearing, and the blue arrow represents the current humanoid odometry

The router_node receives the TaskIntent messages from llm_node and is responsible for dispatching the intent- for navigation, commands are sent via HTTP POST to web_nav_gui. The router node receives TaskIntent message and, specifically for 'navigate to' commands, a fuzzy search is applied against the named_markers JSON file to the TaskIntent declared marker name parameter. For example, TaskIntent: action = "navigate_to_marker" and target= "the kitchen", the fuzzy substring match associates the marker 'kitchen' and resolves the position to (x=2.1, y=1.3, yaw=-1.57). This becomes a HTTP POST to web_nav_gui, also the router_node would directly publish a confirmation string to /llm/response for the humanoid to 'speak'.

For testing, the subsystem met design feature requirements defined in Section 3.4. Voice-commanded navigation to described markers (e.g. "Move to the kitchen") and body-frame relative motion (e.g. "Turn 30 degrees left") saw repeatable and consistent success across varied voice commands. Global-frame coordinate navigation was achieved consistently through web_nav_gui click-to-move functionality. Arrival was detected through position convergence and triggered a completion flag and completion phrase via TTS. Positional accuracy at target location was within the defined 0.5m target and yaw accuracy within 20 degrees.

Figure 5.2.3.2. web_nav_gui map use during testing, where global-frame coordinates are input and the robot navigates to the target coordinates and bearing.

Two failure modes were documented: (1) the sub-system only accepts individual goals at a time – navigation commands during traversal cancel in-progress motion, rather than queuing navigation goals, and (2) SLAM relocalisation performance degrades in areas of the map that are feature poor – such as uniform corridors.

5.2.4 Dexterous Manipulation sub-system

The manipulation sub-system implements project objective as defined in Section 3.5, through two primary components: pick_node, and pick_daemon, with intent-routing performed by llm_node and router_node. This sub-section combines perception, intent-routing and manipulation functionality into a voice-drive pick sequence.

The TaskIntent should look like, action "pick_up", target = "bottle". Router node then dispatches the target from TaskIntent to the /pick_target topic.

The pick_node subscribes to the /pick_target and receives the object label. It compares it with the current detections from /object_detections_body, searches through a matching bottle. If no detected object is matched, the pick_node sends an error status along the /pick_status topic. If there is a successful match, the 3D position is extracted from the /object_detections_body topic in an (x, y, z) format. This 3D object position relative to torso frame, is imputed into Pinocchio IK, and a grasp pose is computed. The grasp pose is displayed and waits for a user to verify the coordinates and publish "confirm" to the /pick_target topic. If the user confirms the pick-up function, the pick command is sent to pick_daemon, which has low-level control of the arm joint positions.

The IK and arm control occurs in pick_daemon, running as a subprocess outside the ROS 2 middleware. The pick_daemon node is responsible for the initialisation of the Pinocchio IK solver, the Unitree Arm Controller SDK, the Unitree Dex3-1 Hand Controller SDK, reading commands from the stdin pipeline, executing motion sequences, and writing status reports to stdout. Command list consists of pick x y z, open, close, home, ready, DONE, OK, READY.

The pick is executed in two stages: first, the arm interpolates to a pre-grasp pose with a 10 cm vertical offset above the target and an 8 cm offset in front of the target. Then the arm moves downwards (along the z axis) closer to the target, the hand closes and grasps the object, then the arm lifts the object above the current grasp pose.

For testing, the complete end-to-end chain from voice command to successful grasp was demonstrated consistently with object variations and position variations. The fixed-pose IK seeding eliminated switching between multiple arm configurations that occurred in earlier testing.

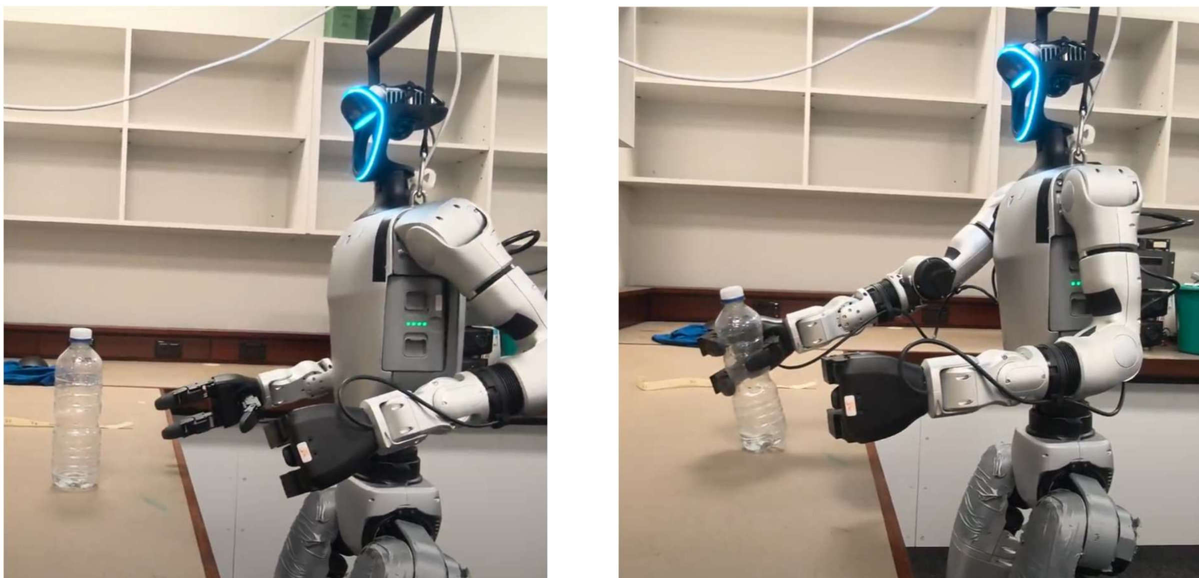

Figure 5.2.4.1 (left) and Figure 5.2.4.2 (right). Humanoid performing vision-based pick-up functionality

Four failure modes were recorded: (1) the system inherits YOLO's closed-set limitation of only 80 detectable classes, where objects outside the set cannot be addressed by name – thus the voice-based pick-up cannot occur, (2) the pick-up functionality is bounded by the reach space of the robot and IK coordinates can fall outside the reach space, (3) the pick-up wrist orientation is fixed, thus

objects requiring non-axis-aligned grasps cannot be picked up (4) when arm trajectories are set, objects in the way are not considered.

5.2.5 Gesture Sub-System

The gesture sub-system achieves the project objective 3.6, through the implementation of three components: gesture_daemon (SDK communication), audio_responder (gesture dispatch to gesture_daemon), and joint_slider_gui_v2 (development tool).

Gestures are initiated by the llm_node which incorporates appropriate gesture tags into its response and the process is described in detail in Section 5.2.1. audio_responder receives the LLM response with appropriate gesture tags integrated from the /llm/response topic and forwards the gesture tags to the gesture_daemon via stdin pipe. The gesture_daemon interfaces with the Unitree SDK gestures API – which contains a library of Unitree developed gestures (handshake, high-five, wave, hug, etc.). Additionally, custom gestures can be recorded through joint_slider_gui_v2 – a Flask-based web GUI that enables individual arm and hand joint control through an array of connected sliders, record, playback and save functionality for arm poses. Concatenated arm poses are saved as gesture JSON files in a motion_sequences folder for future playback.

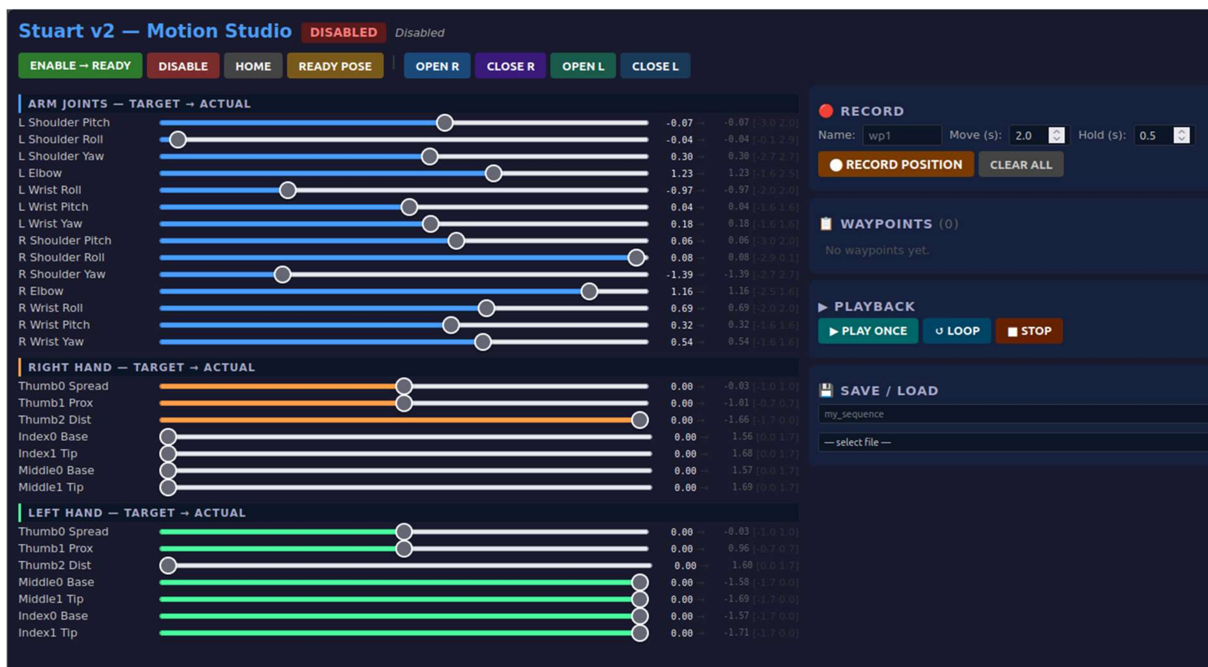

Figure 5.2.5.1. Joint_slider_gui_v2.py visualisation of joint control with sliders. Waypoint recording, playback, save and load interface to the right.

Figure 5.2.5.2. Joint_slider_gui_v2.py visualisation functionality in progress.

For testing, the Unitree-based gestures executed reliably when dispatched via voice command and ran concurrently with humanoid speech (e.g. robot waving while it introduces itself), custom-recorded gesture sequences could be recorded and accurately played back, and gestures could repeatably and successfully be invoked through speech – satisfying all design features defined in Section 3.6.

Two failure modes were documented: (1) the LLM gesture emission was not reliable and the tertiary deterministic filter was heavily relied upon (as described in Section 5.2.1), and (2) use of user-recorded and named gestures was not yet implemented into the intent subsystem.

5.2.6 Intent-Routing Sub-System

The intent-routing architecture is the backbone of the voice-driven autonomy, linking language comprehension to action execution. The implementation comprises two stages: (1) llm_node regex classification (as described in Section 5.2.1) and dispatch logic in router_node.

Stage 1 uses regex-based classification to filter for commands but also their corresponding parameters. These parameters are compiled into a TaskIntent message which contains fields: action, target, x, y, yaw, and raw_text, and is published to /llm/task_intent. The message is consumed by router_node which then dispatches the parameters within TaskIntent to the relevant subsystem through the specific communication pipeline, (i.e. navigate_to_marker, move_relative, turn_by_angle, turn_around) are dispatched via HTTP pipeline to web_nav_gui; pick-up intent is published to /pick_target for pick_node to utilise; gesture intent is forwarded to audio_responder and gesture_daemon. Utterances that are not classified by regex move to Stage 2 llm_node filter for conversational handling.

In testing, all eight intent classes were correctly classified across multiple tests with speech variations and no misclassifications occurred. End-to-end dispatch latency was well below the 5-second target, as the deterministic characteristic for classification was sub-100ms.

5.3 Integrated Demonstration

The integrated lab tour was architected as a standalone ROS 2 node (tour_guide_node) and driven by a YAML configuration – containing predefined station scripts, gestures, and station markers. The node functioned as a linear state machine where the cycle was followed until completion: arrive at marker, speak script segment via /llm/response publish, dispatch relevant gesture tags to gesture_daemon, and wait for completion. The tour_guide_node functions without any additional modifications to other nodes and purely produces output by publishing directly to the relevant topics (/llm/response for speech, /llm/task_intent for navigation and gesture).

Figure 5.3.1. Images of humanoid gesturing while communicating during the laboratory tour guide operation

Once the system is initialised and the tour guide functionality begins, the robot successfully navigates between the six predefined laboratory markers, whilst also delivering speech and performing gesture - all of which are fully autonomous. This functionality is consistent across multiple runs. An additional design feature is that when a person steps in front of the humanoid during locomotion, the object avoidance mechanism will stop the humanoid and wait for the path to clear before continuing. A failure mode that exists is the lack of dynamic audience interaction, where the tour guide cannot react to audience questions during execution. The integration of LLM for question answering is recommended for future work.

5.4 Discussion and Implications

All humanoid capability objectives and their corresponding design features defined in Section 3 were met through subsystem integration and testing on the Unitree G1 humanoid. This thesis demonstrates the successful deployment of an end-to-end autonomy stack on the G1 humanoid with capabilities including conversational language comprehension and response, neural object detection, persistent spatial memory, autonomous navigation, and dexterous manipulation.

As discussed in Section 2, there are only a few examples of publicly available repositories that encompass a humanoid integrated stack. The current academic literature typically advances isolated sub-systems for robotics platforms and end-to-end industry-based stacks are typically closed and proprietary. This thesis addresses this gap by publishing the completed humanoid autonomy stack on a publicly available GitHub repository. This work is reproducible, deployable, and "field-tested" as a baseline for future work, and its modular architecture provides a testbed for isolated sub-systems to be tested within an end-to-end communication-enabled intent-driven stack.

Additionally, this thesis contributes a documented and generalisable solution to a recurring deployment problem in commodity robotics: the conflict between ROS 2 and vendor SDKs. The sub-process spawning and stdin/stdout communication isolation mitigation strategy and architecture is platform agnostic and can be applied to any robotics platform encountering the same conflict issue.

The integrated autonomy stack and the documented solution to ROS 2 / vendor-SDK conflict are intended as resources for future students and the wider robotics community, while the tour-guide functionality provides direct utility to the UWA Robotics and Automation Laboratory as a demonstration and outreach platform.

6. Conclusions and Future Work

This design-and-build thesis successfully integrated and implemented a multimodal end-to-end stack, incorporating voice, vision, locomotion, navigation, manipulation on the Unitree G1 humanoid. The autonomy stack provides the humanoid with the capability to achieve all eight capability objectives as defined in Section 3, validated against the qualitative acceptance criteria defined by the design features. Intent-routing (Section 5.2.6), Communication and cognition (5.2.1), vision-based perception with spatial memory (5.2.2), locomotion and navigation (5.2.3), dexterous manipulation (5.2.4), and communicative gestures (5.2.5) were validated as individual subsystems. The scripted tour guide operation (5.3) and the integrated end-to-end demonstration (5.3) validated their unified functionality as a single autonomy stack.

This thesis contributes a modular stack that runs can be adapted to most commercial humanoid platforms. The stack is modular in the sense that ROS 2 nodes are highly independent, allowing for non-intrusive upgrade pathways. The stack encompasses a baseline architecture that supports SDK and ROS 2 processes running in parallel. Importantly the stack highlights the extent of autonomous humanoid functionality through integrating novel technologies. The thesis explores the limitations of computation on edge hardware. The stack also creates a foundation for additional functionality, including HTN-based task planning and action-policy model deployment. The tour-guide functionality also contributes to the UWA Robotics and Automation Laboratory as a public-facing demonstration platform.

When compared against current frontier humanoid and robotics research, the overarching goals remain consistent – having a fully autonomous and versatile humanoid robot able to execute a large range of tasks from precision manipulation to dynamic full body motion, deep cognitive ability for communication and essentially the ability to do anything a human is able to do. The frontier humanoid and robotics research heavily focus on artificial intelligence based and neural network approaches such as VLAs, Deep Reinforcement Learning policy training, World Action Models, Teleoperation-based imitation learning etc. These AI-based approaches are heavily dependent on access to large computation clusters and multitudes of testing robotics. My thesis deploys a functional autonomous multi-modal stack that has the ability to various commands, and the approach is utilisation of neural network based AI in separated processes when necessary given compute constraints. The benefit of this approach is that further AI-driven components can be non-destructively integrated into the existing stack

For future work, the spatial memory functionality could be integrated into the complete stack, where objects 3D global coordinates are instantiated as markers in web_nav_gui that are able to be navigated to upon command – "move to the bottle". Additionally, spatial memory could be exposed to the LLM (through context injection), allowing for location-related queries.
A limitation of the routing layer, as highlighted in section 5.2.6, is that it can only handle a single command per utterance. The existing stubbed Hierarchical Task Planner node could be completed to decompose compound commands into sequential intents. Further testing of LLM deployment on edge hardware, potentially with hyper-distilled models that have less computation requirements and deeper reasoning ability. Deep Reinforcement Learning policies could be trained in simulation and deployed on the robot as intent-invokable behaviours.

References

Ahn, M., et al. (2022). Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. CoRL 2022. https://arxiv.org/abs/2204.01691

Brohan, A., et al. (2023). RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. CoRL 2023. https://arxiv.org/abs/2307.15818

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. In European Conference on Computer Vision (ECCV). https://arxiv.org/abs/2005.12872

Chi, C., et al. (2023). Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. RSS 2023. https://arxiv.org/abs/2303.04137

Driess, D., et al. (2023). PaLM-E: An Embodied Multimodal Language Model. ICML 2023. https://arxiv.org/abs/2303.03378

Firoozi, R., et al. (2025). Foundation Models in Robotics: Applications, Challenges, and the Future. International Journal of Robotics Research. https://arxiv.org/abs/2312.07843

Foote, T., & Purvis, M. (2010). REP 103 — Standard Units of Measure and Coordinate Conventions. ROS Enhancement Proposal. https://www.ros.org/reps/rep-0103.html

Gat, E. (1998). On three-layer architectures. In D. Kortenkamp, R. P. Bonasso, & R. Murphy (Eds.), Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems (pp. 195–210). MIT Press.

Gu, Q., Kuwajerwala, A., Morin, S., Jatavallabhula, K. M., et al. (2023). ConceptGraphs: Open-vocabulary 3D scene graphs for perception and planning. arXiv. https://arxiv.org/abs/2309.16650

Hoffmann, J., et al. (2022). Training Compute-Optimal Large Language Models. NeurIPS 2022. https://arxiv.org/abs/2203.15556

Huang, W., et al. (2023). VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. CoRL 2023. https://arxiv.org/abs/2307.05973

Hughes, N., Chang, Y., & Carlone, L. (2022). Hydra: A real-time spatial perception system for 3D scene graph construction and optimization. In Robotics: Science and Systems (RSS). https://arxiv.org/abs/2201.13360

Jatavallabhula, K. M., Kuwajerwala, A., Gu, Q., et al. (2023). ConceptFusion: Open-set multimodal 3D mapping. In Robotics: Science and Systems (RSS). https://arxiv.org/abs/2302.07241

Jocher, G., et al. (2023). Ultralytics YOLO (v8+). https://github.com/ultralytics/ultralytics

Kaplan, J., et al. (2020). Scaling Laws for Neural Language Models. https://arxiv.org/abs/2001.08361

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics, 7(66). https://doi.org/10.1126/scirobotics.abm6074

Maruyama, Y., Kato, S., & Azumi, T. (2016). Exploring the Performance of ROS2. EMSOFT 2016. https://doi.org/10.1145/2968478.2968502

Radosavovic, I., et al. (2024). Real-World Humanoid Locomotion with Reinforcement Learning. Science Robotics, 9(89). https://arxiv.org/abs/2303.03381

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in Neural Information Processing Systems (NeurIPS) 28. https://arxiv.org/abs/1506.01497

Unitree Robotics (2024). Unitree G1 SDK Documentation. https://support.unitree.com/home/en/G1_developer

Wu, J., et al. (2023). TidyBot: Personalized Robot Assistance with Large Language Models. Autonomous Robots. https://arxiv.org/abs/2305.05658

Appendix A: System Initialisation Sequence

This appendix A is AI-Claude generated for the purpose of explicit documentation. This appendix documents the complete initialisation procedure for the autonomy stack on the Unitree G1. The sequence is mandatory before any voice-driven operation and assumes the robot is in a de-energised state.

A.1 Hardware Boot-Up

The humanoid is initialised from a de-energised state through the wireless controller in the following sequence:

Press the power button on the G1 battery (short press, then long press).
Power on the wireless controller.
L1 + B — enter damping mode (motors soft, robot remains seated).
L2 + UP — enter ready state (motors energised, joints stiffen).
Lower the robot to the ground with gantry support.
R2 + A — enter motion mode (robot stands and is ready to receive commands).

The robot must be positioned in a clear area with a minimum 1 m radius around it; otherwise the Unitree locomotion controller raises error 421 ("obstacles nearby") and refuses motion commands.
Emergency stop: L1 + A at any time returns the robot to damping mode, de-energising the motors. The robot will fall unless suspended from the fall-arrest gantry.

A.2 Network Connection

The development laptop connects to the Jetson Orin via Ethernet on interface enp3s0. Default addresses are listed below.

Device	IP address
Jetson Orin (onboard)	192.168.123.164
Development laptop	192.168.123.99

SSH access to the Jetson from the laptop:

```
ssh unitree@192.168.123.164
```

All subsequent commands are executed on the Jetson over SSH.

A.3 Stack Initialisation

Each ROS 2 node runs in its own terminal. Every terminal begins with the same environment sourcing block. The leading "exec bash --norc --noprofile" is required to bypass the shared .bashrc, which sources conflicting ROS distributions:

```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
```

Terminals must be launched in the order below. Wait for the indicated success message before proceeding.

Terminal 1 — Load SLAM map (one-shot, terminal can be closed after success):

```
exec bash --norc --noprofile
python3 ~/slam_client.py load_map /home/unitree/lab_map_v1.pcd
```

Wait for: Successfully started re-location.

Terminal 2 — Web navigation GUI:

```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
python3 ~/web_nav_gui.py
```

Wait for: Web GUI on http://0.0.0.0:8081. The GUI auto-spawns the slam_client daemon as a child process.
Terminal 3 — Router node:
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron router_node
```
Wait for: router_node ready.
Terminal 4 — LLM node (Qwen 2.5 3B; model load takes 10–20 seconds):
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run llm_node llm_node
```
Wait for: Stuart is ready. Listening on /llm/input.
Terminal 5 — Audio responder (auto-spawns tts_daemon and gesture_daemon):
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron audio_responder
```
Wait for: Audio responder ready — say "Hey Stuart" to activate.
Terminal 6 — Camera node:
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron camera_node
```
Terminal 7 — YOLO detector:
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron yolo_node
```
Terminal 8 — Depth projector:
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron depth_projector_node
```
Terminal 9 — Spatial memory:
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron spatial_memory_node
```
Terminal 10 — Pick node (only required if dexterous manipulation is to be exercised):
```
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron pick_node
```
The pick node auto-spawns the pick_daemon subprocess on the first pick request.

A.4 Demonstration Mode

For the scripted laboratory tour (Section 5.3), the tour_guide_node replaces the LLM and router nodes. To stop those nodes:
```
pkill -f "ros2 run llm_node"
pkill -f "ros2 run cameron router_node"
```
Then launch the tour node in a fresh terminal:
```
exec bash --norc --noprofile
```

source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 run cameron tour_guide_node
The audio responder, web navigation GUI, and slam_client must remain running from the standard initialisation.
A.5 Verification
Confirm the stack is healthy by inspecting key topics in a fresh terminal:
exec bash --norc --noprofile
source ~/setup_cameron.sh && source ~/cam_ws/install/setup.bash
ros2 topic hz /audio_msg
ros2 topic hz /object_detections
ros2 topic echo /llm/response --once
The first two confirm the ASR and vision pipelines are streaming; the third confirms the cognition pipeline is reachable.
To test the cognition pipeline without using the microphone:
ros2 topic pub /llm/input std_msgs/msg/String "{data: 'what do you see'}" --once
A spoken response should be produced within four seconds.
A.6 Shutdown
To terminate all stack nodes cleanly:
pkill -f
"llm_node|audio_responder|camera_node|yolo_node|depth_projector_node|spatial_memory_node|router_node|web_nav_gui|pick_node|tour_guide_node"
To return the robot to a safe de-energised state, press L1 + A on the controller, then power down the battery via long-press of the power button.

Appendix B

Boot-up sequence for robot.

Testing and running functionality and complete stack initialisation sequence consisted of the following steps. The humanoid boots up in a de-energised state and must enter ready mode through the Unitree controller (L2+B -> L2+UP -> R2->A)