UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

End-to-End Language-Guided Reinforcement Learning for Legged Robots

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

AUTHOR: Ojeda Gandía, Raúl

Tutor: Mestre Gascón, Antoni

Cotutor: Fons Cors, Joan Josep

ACADEMIC YEAR: 2024/2025

Abstract

The rapid advancement of large language models has created a significant gap between software-based intelligence and the physical capabilities of robotic systems. This thesis addresses this challenge by developing an end-to-end reinforcement learning framework that directly maps natural language instructions to continuous motor actions for legged robots, eliminating the need for intermediate symbolic representations. The proposed solution integrates multilingual sentence transformer embeddings directly into the observation space of an actor-critic policy. This policy is trained in a high-fidelity, GPU-accelerated physics simulator (Isaac Lab) using a comprehensive dataset of 922 multilingual commands. The trained policy was subsequently optimized with TensorRT and successfully deployed on a physical JetHexa hexapod robot, controlled in real-time by an embedded NVIDIA Jetson Nano via a distributed ROS architecture. Evaluation results demonstrate that the system can successfully generalize to novel commands not seen during training, achieving 55% directional accuracy on a diverse test set. This confirms that the model learned a genuine mapping from semantic intent to physical action. While velocity magnitude tracking was constrained by hardware limitations, the system's ability to correctly interpret movement direction validates the feasibility of direct language-to-motor grounding. This work contributes a scalable and computationally efficient approach for intuitive human-robot interaction, advancing the development of more generalist and accessible robotic systems.

Key words: Reinforcement learning, natural language, legged robot

Resumen

El rápido avance de los grandes modelos de lenguaje ha creado una brecha significativa entre la inteligencia basada en *software* y las capacidades físicas de los sistemas robóticos. Esta tesis aborda este desafío mediante el desarrollo de un marco de aprendizaje por refuerzo de extremo a extremo que traduce directamente instrucciones en lenguaje natural a acciones motoras continuas para robots con patas, eliminando la necesidad de representaciones simbólicas intermedias. La solución propuesta integra *embeddings* multilingües de un *sentence transformer* directamente en el espacio de observación de una política *actor-critic*. Dicha política se entrena en un simulador de física de alta fidelidad acelerado por GPU (Isaac Lab) utilizando un completo conjunto de datos de 922 comandos multilingües. La política entrenada fue optimizada posteriormente con TensorRT y desplegada con éxito en un robot hexápodo físico JetHexa, controlado en tiempo real por un sistema embebido NVIDIA Jetson Nano a través de una arquitectura ROS distribuida. Los resultados de la evaluación demuestran que el sistema puede generalizar con éxito a comandos novedosos no vistos durante el entrenamiento, alcanzando una precisión direccional del 55 % en un diverso conjunto de pruebas. Esto confirma que el modelo aprendió una correspondencia genuina entre la intención semántica y la acción física. Aunque el seguimiento de la magnitud de la velocidad se vio limitado por las restricciones del *hardware*, la capacidad del sistema para interpretar correctamente la dirección del movimiento valida la viabilidad de la conexión directa entre lenguaje y acción motora. Este trabajo aporta un enfoque escalable y computacionalmente eficiente para la interacción intuitiva humano-robot, avanzando en el desarrollo de sistemas robóticos más generalistas y accesibles.

Palabras clave: Aprendizaje por refuerzo, lenguaje natural, robot con patas

Resum

El ràpid avanç dels grans models de llenguatge ha creat una bretxa significativa entre la intel·ligència basada en programari i les capacitats físiques dels sistemes robòtics. Esta tesi aborda aquest desafiament mitjançant el desenvolupament d'un marc d'aprenentatge per reforç d'extrem a extrem que traduïx directament instruccions en llenguatge natural a accions motores contínues per a robots amb potes, eliminant la necessitat de representacions simbòliques intermèdies. La solució proposada integra *embeddings* multilingües d'un *sentence transformer* directament en l'espai d'observació d'una política *actor-critic*. Esta política s'entrena en un simulador de física d'alta fidelitat accelerat per GPU (Isaac Lab) utilitzant un complet conjunt de dades de 922 comandaments multilingües. La política entrenada va ser optimitzada posteriorment amb TensorRT i desplegada amb èxit en un robot hexàpode físic JetHexa, controlat en temps real per un sistema embegut NVIDIA Jetson Nano a través d'una arquitectura ROS distribuïda. Els resultats de l'avaluació demostren que el sistema pot generalitzar amb èxit a comandaments nous no vistos durant l'entrenament, aconseguint una precisió direccional del 55% en un divers conjunt de proves. Això confirma que el model va aprendre una correspondència genuïna entre la intenció semàntica i l'acció física. Encara que el seguiment de la magnitud de la velocitat es va veure limitat per les restriccions del maquinari, la capacitat del sistema per a interpretar correctament la direcció del moviment valida la viabilitat de la connexió directa entre llenguatge i acció motora. Este treball aporta un enfocament escalable i computacionalment eficient per a la interacció intuïtiva humà-robot, avançant en el desenvolupament de sistemes robòtics més generalistes i accessibles.

Paraules clau: Aprenentatge per reforç, llenguatge natural, robot amb potes

Contents

Contents v
List of Figures ix
List of Tables x

1 Introduction 1
1.1 Motivation . . . 1
1.1.1 Technical Motivation . . . 2
1.1.2 Professional Motivation . . . 2
1.2 Objectives . . . 3
1.3 Expected Impact . . . 3
1.3.1 Technical Impact . . . 4
1.3.2 Societal Impact . . . 4
1.3.3 Alignment with Sustainable Development Goals . . . 4
1.4 Structure . . . 5
2 State of the Art 7
2.1 Evolution of Legged Robot Locomotion . . . 7
2.1.1 Classical Foundations and Model-Based Approaches . . . 7
2.1.2 The Reinforcement Learning Revolution . . . 8
2.1.3 Sim-to-Real Transfer and Domain Randomization . . . 8
2.1.4 Multi-Embodiment and Unified Approaches . . . 8
2.2 Integrating Natural Language into Robotic Control . . . 9
2.2.1 Foundation Models and Large Language Models . . . 9
2.2.2 Vision-Language-Action Models . . . 9
2.2.3 Retrieval-Augmented Generation and Embodied Intelligence . . . 10
2.2.4 Multimodal Integration and Sensor Fusion . . . 10
2.3 Critique of the State of the Art . . . 10
2.3.1 The Hierarchical Bottleneck . . . 10
2.3.2 The Manipulation Bias . . . 11
2.3.3 Scalability and Data Requirements . . . 11
2.3.4 Safety and Reliability Concerns . . . 12
2.4 Proposal . . . 12
2.4.1 Direct Language-Motor Grounding . . . 12
2.4.2 Locomotion-Centered Design . . . 13
2.4.3 Efficiency and Accessibility . . . 13
2.4.4 Validation Through Real-World Deployment . . . 13
3 Problem Analysis 15
3.1 Requirements Specification . . . 15
3.1.1 Target Platform Overview . . . 15
3.1.2 Core Functional Requirements . . . 16
3.1.3 Performance and Resource Constraints . . . 16
3.2 Security Analysis . . . 17
3.2.1 Attack Surface Analysis . . . 17
3.2.2 Privacy and Access Control . . . 17

3.3 Energy and Algorithmic Efficiency Analysis . . . 17
3.3.1 Power Budget Analysis . . . 17
3.3.2 Computational Optimization . . . 18
3.4 Legal and Ethical Framework Analysis . . . 18
3.4.1 Liability and Safety Considerations . . . 18
3.4.2 Bias and Accessibility . . . 18
3.5 Risk Analysis . . . 18
3.5.1 Critical Risk Assessment . . . 19
3.5.2 Safety and Operational Risks . . . 19
3.6 Identification and Analysis of Possible Solutions . . . 19
3.6.1 Alternative Architecture Comparison . . . 19
3.6.2 End-to-End Approach Justification . . . 19

4 Proposed Solution **21**
4.1 Solution Overview . . . 21
4.2 Work Plan . . . 21
4.2.1 Development Phases and Timeline . . . 21
4.2.2 Critical Path Analysis and Risk Management . . . 23
4.3 Budget . . . 23
4.3.1 Hardware and Software Resources . . . 23
4.4 Solution Design . . . 24
4.4.1 System Architecture . . . 24
4.4.2 Detailed Design . . . 25
4.5 Technology Used . . . 28
4.5.1 Development Environment and Tools . . . 28
4.5.2 Hardware Platform Selection . . . 28
4.5.3 Software Architecture Decisions . . . 29
4.5.4 Optimization and Performance Strategies . . . 29

5 Implementation **31**
5.1 Development Methodology . . . 31
5.2 Phase 1: Dataset Creation and Language Processing Pipeline . . . 31
5.2.1 Multilingual Command Dataset Development . . . 31
5.2.2 Dataset Generation Using Large Language Models . . . 32
5.2.3 Test Set Design for Generalization Evaluation . . . 35
5.2.4 Principal Component Analysis Optimization . . . 37
5.3 Phase 2: Simulation Environment Development . . . 39
5.3.1 Isaac Lab Framework Architecture . . . 39
5.3.2 JetHexa Robot Model Integration . . . 39
5.3.3 Language Command System Integration . . . 40
5.3.4 Observation Space Architecture . . . 41
5.3.5 Terrain Generation and Domain Randomization . . . 42
5.3.6 Foundation from Existing Velocity Locomotion Task . . . 43
5.4 Phase 3: Reinforcement Learning Training . . . 46
5.4.1 Proximal Policy Optimization Algorithm . . . 46
5.4.2 Training Infrastructure and Parallelization . . . 46
5.4.3 Neural Network Architecture and Control Loop Implementation . . . 47
5.4.4 Reward Function Engineering . . . 50
5.4.5 Training Monitoring and Evaluation . . . 53
5.5 Phase 4: Real-World Deployment System Architecture . . . 53
5.5.1 System Architecture and Information Flow . . . 54
5.5.2 Project File Structure and Organization . . . 55
5.5.3 ROS Communication Infrastructure . . . 55
5.5.4 Language Command Node: NLP Pipeline Implementation . . . 56

5.5.5 RL Inference Node: Real-Time Control Loop . . . 56
5.5.6 Robot Driver Node: Hardware Interface and Servo Control . . . 57
5.5.7 System Integration and Operational Procedures . . . 59
5.6 Phase 5: Performance Evaluation and System Validation . . . 59
5.6.1 Training Campaign Overview and Model Development History . . . 59
5.6.2 Final Model Analysis and Training Characteristics . . . 60
5.6.3 Systematic Evaluation Methodology . . . 66
5.6.4 Real-World Deployment Validation Strategy . . . 67
5.6.5 Integrated Evaluation Pipeline . . . 68

6 Results **69**
6.1 Final Model Training Performance . . . 69
6.2 Language Understanding Baseline Evaluation . . . 69
6.2.1 Baseline Architecture and Results . . . 70
6.3 Reinforcement Learning Agent Performance . . . 70
6.3.1 Evaluation Methodology . . . 70
6.3.2 Overall Performance Metrics . . . 70
6.3.3 Performance Gap Analysis . . . 71
6.3.4 Command Category Performance Analysis . . . 71
6.3.5 Training vs. Test Performance . . . 74
6.3.6 Direction Performance . . . 75
6.3.7 Direction Accuracy Baseline Comparison . . . 76
6.4 Real-World Deployment Validation . . . 76
6.4.1 System Integration Performance . . . 76
6.4.2 Validation Criteria Assessment . . . 77

7 Conclusions **79**
7.1 Fulfillment of Objectives . . . 79
7.1.1 Primary Objectives . . . 79
7.1.2 Secondary Objectives . . . 80
7.2 Reflection on the Work Realized . . . 80
7.2.1 Problems Encountered and Solutions . . . 80
7.2.2 Errors Committed and Lessons Learned . . . 81
7.2.3 Personal and Professional Learning . . . 81
7.3 Relation of the Work to Master's Studies . . . 81
7.4 Future Work . . . 82
7.4.1 Improvements and Short-Term Extensions . . . 82
7.4.2 New Research Directions . . . 82
7.4.3 Paths to Avoid . . . 83

Bibliography **85**

Appendices

Appendices

A JetHexa Platform Specifications **89**
A.1 Platform Overview . . . 89
A.2 Hardware Specifications . . . 90
A.2.1 Physical Characteristics . . . 90
A.2.2 Actuator System . . . 90
A.2.3 Computing Platform . . . 91
A.2.4 Expansion Board . . . 92
A.3 Sensor Suite . . . 92
A.3.1 Inertial Measurement . . . 92

A.3.2 3D Depth Camera . . . 93
A.3.3 LiDAR System . . . 93
A.3.4 Microphone Array . . . 94
A.4 Power System . . . 95
A.4.1 Battery Configuration . . . 95
A.4.2 Power Distribution . . . 95
A.5 Software Architecture . . . 96
A.5.1 Operating System and Framework . . . 96
A.5.2 Control Interfaces . . . 96
A.6 Locomotion Capabilities . . . 96
A.6.1 Gait Patterns . . . 96
A.7 Research Applications . . . 97
A.8 Limitations and Considerations . . . 97
B Humanoid G1 Language Locomotion: ARA Course Project 99
B.1 Project Overview Slides . . . 99
B.2 Technical Implementation Details . . . 101
B.3 Experimental Results . . . 101
B.4 Project Conclusions and Implications . . . 102
B.5 Relationship to Main Research . . . 102
C Objetivos De Desarrollo Sostenible 103

List of Figures

3.1 ROS Hexapod Robot JetHexa 16

4.1 End-to-end system architecture showing direct integration of language embeddings into the reinforcement learning control loop 24
4.2 Detailed language processing pipeline with dimensionality reduction . . . 25
4.3 Policy network architecture with observation space decomposition 26
4.4 JetHexa Hexapod Robot 27
4.5 Distributed ROS deployment architecture with performance specifications 27

5.1 Cumulative explained variance as a function of principal components for language command embeddings. The plot shows the trade-off between dimensionality reduction and information preservation, with key decision points at 90% and 95% variance thresholds. 37
5.2 JetHexa USD model visualization on Isaac Sim 40
5.3 Rough terrain environment used in training. 42
5.4 Parallel training of G1 robot following multiple movement commands . . 45
5.5 G1 robot following the command "proceed in a forward direction" downstairs 45
5.6 JetHexa robots following multiple commands in parallel on a rough terrain. 53
5.7 Information flow through the distributed ROS architecture showing critical timing and data transformations 54
5.8 Mean episode reward progression over 10,000 training iterations, showing rapid initial learning followed by stable convergence around 40 points with peak performance of 44.75 at iteration 4,470 61
5.9 Linear velocity tracking reward progression, demonstrating rapid convergence to near-optimal performance (2.11 final value) with excellent stability (convergence ratio: 0.0458) 62
5.10 Angular velocity tracking reward progression, showing rapid early convergence to peak performance (1.46) at iteration 975, followed by stable maintenance around 1.40 62
5.11 Policy noise standard deviation evolution, demonstrating stable exploration maintenance (final value: 1.05) with appropriate adaptation throughout training 63
5.12 Policy entropy loss progression from 25.51 to 23.59, showing appropriate policy convergence while maintaining exploration capability 64
5.13 Coxa joint deviation penalty progression, stabilizing at -0.198 while maintaining acceptable postural control during language-guided locomotion . 64
5.14 Feet air time reward progression, improving from -0.026 to -0.088 with excellent convergence (0.084 ratio), indicating successful tripod gait development 65
5.15 Feet sliding penalty stabilizing at -0.067 with excellent convergence, demonstrating effective prevention of unrealistic foot motion 65
5.16 Action smoothness penalty showing variability (convergence ratio: 2.53) while maintaining regularization function for control quality 66

6.1 Predictions using MLP Baseline on test set . . . 70
6.2 Sequence of real JetHexa robot performing the test set instruction "glide to the left" . . . 77

A.1 JetHexa hexapod robot . . . 89
A.2 HX-35H intelligent serial bus servo used in JetHexa joints . . . 90
A.3 NVIDIA Jetson Nano B01 development board integrated in JetHexa . . . 91
A.4 Multi-functional Expansion Board . . . 92
A.5 3D depth camera mounted on JetHexa for visual perception . . . 93
A.6 EAI G4 Lidar . . . 93
A.7 6-channel circular microphone array for audio processing . . . 94
A.8 11.1V LiPo battery pack with integrated charging system . . . 95

B.1 Project motivation and objectives . . . 99
B.2 G1 task description and dataset composition . . . 100
B.3 G1 feature extraction and observation space architecture . . . 101
B.4 G1 project performance results by category and movement type . . . 101
B.5 Project conclusions and implications for generalist robotics . . . 102

List of Tables

3.1 Critical Performance Requirements . . . 16
3.2 JetHexa Power Distribution . . . 18
3.3 Primary Risk Analysis . . . 19
3.4 Solution Architecture Comparison . . . 19

4.1 Project Development Phases and Time Allocation . . . 21
4.2 Hardware and Software Budget . . . 23
4.3 Development Environment and Toolchain . . . 28

5.1 Dataset language statistics . . . 34
5.2 Dataset movement distribution . . . 34
5.3 PCA Dimensionality Reduction Analysis . . . 38
5.4 ROS Topic Communication Specifications . . . 55

6.1 Final Model Training Performance Summary . . . 69
6.2 MLP Baseline Performance - Language Understanding Limits . . . 70
6.3 RL Agent Performance Comparison with Baselines (MAE) . . . 71
6.4 Performance by Command Category (MAE) . . . 72
6.5 Performance by Movement Type and Relevant Velocity Component . . . 72
6.6 Training vs. Test Set Performance Comparison (MAE) . . . 74
6.7 Direction Accuracy by Movement Type . . . 75
6.8 Direction Accuracy Comparison Across Methods . . . 76
6.9 Real-World System Performance Metrics . . . 77

A.1 JetHexa Physical Specifications . . . 90
A.2 HX-35H Servo Specifications . . . 91
A.3 Jetson Nano Computing Specifications . . . 91

A.4 3D Depth Camera Specifications 93
A.5 LiDAR Specifications (EAI G4 Lidar) 94
A.6 Microphone Array Specifications (iFLYTEK 6-Microphone Array) 94
A.7 Battery System Specifications 95
A.8 Power Consumption Analysis 95
A.9 Software Stack 96
A.10 Available Gait Patterns 96

CHAPTER 1
Introduction

The convergence of artificial intelligence and robotics has reached a pivotal moment in 2025, with humanoid robots transitioning from laboratory prototypes to commercial deployments. The humanoid robot market was valued at $2.03 billion in 2024 and is predicted to increase to more than $13 billion by 2029, signaling an unprecedented acceleration in the deployment of intelligent physical agents [9].

Yet despite this remarkable progress, a fundamental challenge persists: the gap between software intelligence and physical intelligence. Large Language Models (LLMs) have demonstrated near-human capabilities in reasoning, planning, and communication, but translating this semantic understanding into precise physical actions remains an unsolved problem [41]. This disconnect is particularly pronounced in legged locomotion, where the continuous, dynamic nature of movement and the need for precise temporal coordination across multiple degrees of freedom present unique challenges.

Traditional robotic control systems rely heavily on predefined motion primitives, carefully engineered state machines, or hierarchical architectures that separate high-level planning from low-level execution [24, 18]. While successful in constrained environments, these approaches face critical scalability barriers, semantic discontinuities, and limited adaptability that prevent their deployment in real-world scenarios where variability is the norm rather than the exception.

The integration of natural language understanding with robotic control systems represents one of the most promising pathways toward truly generalist robotics. Recent breakthroughs like LEGION (2025) and ELLMER (2025) have shown that robots can preserve and combine knowledge across tasks using language embeddings, enabling lifelong learning capabilities that mirror human knowledge acquisition [25, 26]. However, most existing language-guided robotic systems focus primarily on manipulation tasks, with limited attention to the equally important challenge of language-guided locomotion.

1.1 Motivation

The motivation for this research stems from several critical observations about the current state of robotics and artificial intelligence that create both technical challenges and unprecedented opportunities for advancement.

1.1.1. Technical Motivation

The disparity between language-based intelligence and embodied intelligence has become increasingly apparent as AI systems demonstrate remarkable capabilities in digital domains while struggling with basic physical tasks. Figure's Helix system represents the first Vision-Language-Action model to directly control an entire humanoid upper body from natural language, capable of generating long-horizon, collaborative, dexterous manipulation on the fly without task-specific demonstrations [13]. This achievement highlights both the potential and the current limitations of language-guided robotics.

Current approaches to language-guided robotics face several fundamental limitations:

Hierarchical Complexity: Most existing approaches rely on hierarchical architectures that separate language understanding, task planning, and motor execution. While this modular design offers interpretability, it introduces multiple points of failure and limits the system's ability to discover novel solutions that span multiple levels of the hierarchy [2].

Limited Direct Grounding: Few approaches demonstrate direct grounding of language in continuous motor actions. Most systems use language for high-level task specification while relying on pre-programmed primitives for actual execution [35].

Locomotion Gap: The majority of language-guided robotics research focuses on manipulation tasks, with limited work addressing the challenge of language-guided locomotion. Locomotion presents unique challenges due to its continuous, dynamic nature and the need for precise temporal coordination.

1.1.2. Professional Motivation

The professional motivation for this work is driven by the recognition that natural language represents humanity's most sophisticated interface for expressing complex intentions and coordinating collaborative activities. The ability to communicate with robots using natural language could fundamentally transform human-robot interaction across multiple domains:

Industrial Applications: Factory workers could direct robots using natural language commands, enabling more flexible manufacturing processes and reducing the need for specialized programming expertise. This could accelerate the adoption of robotic assistance in small and medium enterprises that lack extensive technical resources.

Assistive Technologies: Language-guided robots could provide intuitive interfaces for elderly individuals or those with mobility impairments, enabling commands like "walk me to the kitchen slowly" or "help me navigate around these obstacles."

Emergency Response: Robots capable of understanding natural language commands like "navigate to the collapsed structure and search for survivors" could enable remote coordination in dangerous environments where traditional teleoperation interfaces are impractical.

The economic implications are significant: the global market for reinforcement learning technologies was over $52B in 2024 and is projected to reach $32T by 2037, growing at around 65% CAGR during 2025-2037 [11].

1.2 Objectives

The primary objective of this thesis is to develop an end-to-end neural network approach that can directly map natural language instructions to motor actions for legged robots without requiring intermediate symbolic representations or predefined motion primitives. This overarching goal is achieved through several specific objectives:

Primary Objectives:

1. **Develop Direct Language-Motor Grounding:** Create a unified framework that integrates language understanding directly into the reinforcement learning observation space, enabling direct associations between natural language semantics and motor control signals.
2. **Achieve Multilingual Capability:** Design and validate a system capable of understanding and executing locomotion commands across multiple languages (English, Spanish, and potentially others), demonstrating the universal nature of embodied intelligence.
3. **Implement End-to-End Learning:** Train a single neural network policy capable of interpreting diverse natural language instructions and executing corresponding motor behaviors without hierarchical task decomposition.
4. **Validate Real-World Deployment:** Demonstrate the approach through both high-fidelity simulation training and real-world deployment on a hexapod robot platform, proving the practical feasibility of the proposed method.

Secondary Objectives:

1. **Demonstrate Emergent Semantic Understanding:** Validate the robot's ability to generalize to novel command combinations and variations not explicitly present in training data.
2. **Achieve Computational Efficiency:** Develop optimization strategies that enable real-time performance on embedded hardware suitable for mobile robotic platforms.
3. **Establish Evaluation Metrics:** Create comprehensive benchmarks for assessing language-guided locomotion performance across different command types and complexity levels.
4. **Enable Scalable Extension:** Design the architecture to support the addition of new command types without requiring complete retraining.

These objectives are measurable through quantitative metrics including command execution accuracy, response latency, stability measures, and generalization performance on held-out test sets.

1.3 Expected Impact

This research is expected to make significant contributions across multiple dimensions, from technical advancement to broader societal implications.

1.3.1. Technical Impact

Advancement in Embodied AI: This work contributes to the broader field of embodied artificial intelligence by demonstrating how language understanding can be directly grounded in physical action without explicit symbolic reasoning, potentially influencing future research directions in the field.

Scalable Robot Control: The end-to-end approach eliminates the need for manually designed intermediate representations, potentially enabling more scalable deployment of language-controlled robotic systems across diverse applications and environments.

Methodological Innovation: The integration of dimensionality-reduced language embeddings directly into the RL observation space provides a computationally efficient method for incorporating semantic information into motor control policies, which could be adapted for other robotic control tasks.

1.3.2. Societal Impact

Democratization of Robotics: By enabling direct natural language control, this work reduces the technical expertise required for robot operation, making robotic systems more accessible to non-expert users and potentially accelerating adoption across diverse sectors.

Environmental Benefits: General-purpose robots that can adapt to multiple tasks through language commands could replace numerous specialized systems, reducing manufacturing overhead and resource consumption while extending the useful lifecycle of robotic platforms.

Workforce Enhancement: Language-guided robots could work alongside humans in various industries, requiring minimal retraining when tasks change and enabling more flexible human-robot collaboration without displacing human workers but rather augmenting their capabilities.

Educational Applications: The natural language interface could make robotics more accessible for educational applications, allowing students to program robots using natural language rather than traditional programming languages, potentially inspiring a new generation of robotics researchers and practitioners.

1.3.3. Alignment with Sustainable Development Goals

This research aligns with several United Nations Sustainable Development Goals:

Goal 9 - Industry, Innovation and Infrastructure: By advancing robotic technologies that can be easily deployed and reconfigured, this work contributes to building resilient infrastructure and promoting inclusive and sustainable industrialization.

Goal 8 - Decent Work and Economic Growth: Language-guided robots can enhance workplace safety and efficiency while creating new job categories focused on human-robot collaboration rather than replacing human workers.

Goal 4 - Quality Education: The intuitive interface developed in this work can make robotics education more accessible and engaging for students across different backgrounds and technical expertise levels.

1.4 Structure

This thesis is organized into seven main chapters, each addressing specific aspects of the end-to-end language-guided locomotion system:

Chapter 1 - Introduction: Provides the foundational context and motivation for this research, establishing the problem statement, objectives, and expected contributions. This chapter positions the work within the broader landscape of artificial intelligence and robotics while highlighting the specific challenges addressed.

Chapter 2 - State of the Art: Presents a comprehensive review of related work in language-guided robotics, reinforcement learning for legged locomotion, and end-to-end learning approaches. This chapter identifies gaps in current approaches and positions our contribution within the existing literature, with particular focus on recent advances in Vision-Language-Action models and their applications to robotics.

Chapter 3 - Problem Analysis: Provides a detailed analysis of the research problem from multiple perspectives, including technical requirements, security considerations, energy efficiency, and ethical implications. This chapter examines alternative approaches and justifies the selection of the end-to-end methodology.

Chapter 4 - Proposed Solution: Details the system architecture, design decisions, and technologies employed in the implementation. This chapter covers the language processing pipeline, reinforcement learning integration, and real-world deployment architecture, providing sufficient detail for reproduction of the work.

Chapter 5 - Implementation: Provides a comprehensive description of the implementation process, covering dataset creation, simulation environment setup, neural network training, and real-world deployment. This chapter documents the various technical challenges encountered and the solutions developed to address them.

Chapter 6 - Results: Presents experimental results including quantitative performance metrics, qualitative behavioral analysis, and validation of the system's ability to generalize to novel commands. This chapter includes both simulation-based evaluation and real-world performance assessment.

Chapter 7 - Conclusions: Summarizes the key contributions of this research, discusses limitations and lessons learned, and outlines directions for future work. This chapter also reflects on the broader implications of the research for the field of embodied artificial intelligence.

CHAPTER 2
State of the Art

The field of language-guided robotics sits at the intersection of multiple rapidly evolving research domains, including natural language processing, computer vision, reinforcement learning, and robotic control. This chapter provides a comprehensive analysis of the current state of research across these interconnected areas, with particular emphasis on recent breakthroughs that have shaped the landscape of embodied artificial intelligence.

2.1 Evolution of Legged Robot Locomotion

The trajectory of legged robot locomotion research has undergone a fundamental paradigm shift over the past decade, evolving from model-based control approaches to data-driven learning methodologies that achieve unprecedented levels of agility and robustness.

2.1.1. Classical Foundations and Model-Based Approaches

Early developments in legged robotics were grounded in classical control theory and biomechanical insights. The Zero Moment Point (ZMP) criterion, introduced by Vukobratović and Borovac [40], provided the theoretical foundation for stable bipedal locomotion by ensuring that the ground reaction force remained within the support polygon. This approach, while providing theoretical guarantees, resulted in conservative, quasi-static gaits that were highly sensitive to model uncertainties and environmental variations.

The Spring-Loaded Inverted Pendulum (SLIP) model, proposed by Cavagna et al. and later formalized for robotics applications, offered a simplified yet effective framework for understanding the fundamental dynamics of running gaits [33]. This template-based approach enabled the development of energy-efficient hopping and running behaviors in platforms like the MIT Cheetah series, demonstrating the potential for dynamic locomotion in legged systems.

Model Predictive Control (MPC) emerged as a powerful framework for handling the complex, multi-contact dynamics of legged robots while incorporating constraints and optimization objectives. Recent implementations have demonstrated remarkable success in real-time applications, with systems achieving control frequencies of 1000 Hz while maintaining stability across diverse terrains [33]. However, these approaches require accurate dynamic models and struggle with unmodeled disturbances and terrain variations.

2.1.2. The Reinforcement Learning Revolution

The integration of deep reinforcement learning into legged robotics represents a watershed moment in the field. The seminal work by Hwangbo et al. [18] demonstrated that end-to-end reinforcement learning could produce highly dynamic and robust locomotion behaviors in quadruped robots, achieving performance that exceeded traditional model-based approaches. This breakthrough established the foundation for modern learning-based locomotion control by showing that policies trained in simulation could successfully transfer to real hardware.

Recent advances have pushed the boundaries of what is achievable through learning-based approaches. Margolis et al. [24] achieved record-breaking agility with their Mini Cheetah system, demonstrating sprinting speeds up to 6.0 m/s and high-speed turning maneuvers. Their approach achieved a Froude number of 2.9, substantially exceeding previous applications of reinforcement learning to legged locomotion and approaching the agility levels observed in biological systems.

The work by Radosavovic et al. [32] represents a significant milestone in humanoid locomotion, demonstrating successful reinforcement learning-based control of Agility Robotics' full-sized Digit humanoid robot. Their approach utilized transformer-based architectures to process historical observations and actions, enabling adaptive behavior that could handle outdoor environments, uneven terrain, and unexpected disturbances without explicit modeling of environmental properties.

2.1.3. Sim-to-Real Transfer and Domain Randomization

The challenge of transferring policies trained in simulation to real-world hardware has been addressed through sophisticated domain randomization and system identification techniques. Modern approaches randomize not only physical parameters such as mass, friction, and joint dynamics, but also include sensor noise, actuator delays, and environmental conditions [15].

Recent work has demonstrated that comprehensive domain randomization can achieve remarkable sim-to-real transfer performance. The study by Lee et al. [23] showed that policies trained with sufficient randomization could handle terrains and conditions not explicitly present in the training data, suggesting that the learned representations capture fundamental principles of locomotion rather than memorizing specific scenarios.

The emergence of differentiable simulation frameworks such as Brax and Isaac Gym has accelerated training speeds by orders of magnitude, enabling researchers to train policies with millions of environment interactions in hours rather than weeks [15]. This computational efficiency has made it feasible to explore more complex behaviors and larger policy networks.

2.1.4. Multi-Embodiment and Unified Approaches

A significant trend in recent research is the development of unified control frameworks that can handle multiple robot morphologies. The work by Bohlinger et al. [6] introduced URMA (Unified Robot Morphology Architecture), demonstrating that a single learning framework could control diverse legged platforms including quadrupeds, humanoids, and hexapods. This approach addresses the scalability challenge by enabling knowledge transfer across different robot designs.

The concept of morphology-agnostic representations has gained traction, with researchers developing observation and action spaces that can adapt to different numbers

of legs, joint configurations, and body geometries. This line of research suggests that fundamental locomotion principles may be universal across different legged embodiments, opening possibilities for more generalizable control systems.

2.2 Integrating Natural Language into Robotic Control

The integration of natural language understanding with robotic control represents a convergence of advances in large language models, multimodal learning, and embodied AI. This section examines the evolution from early symbolic approaches to current end-to-end learning systems.

2.2.1. Foundation Models and Large Language Models

The breakthrough success of large language models has catalyzed significant interest in their application to robotics. The work by Wang et al. [41] provides a comprehensive survey of opportunities and challenges in leveraging LLMs for robotic applications, identifying key areas where language models can enhance robot capabilities including task planning, human-robot communication, and semantic understanding.

Recent work has demonstrated that pre-trained language models contain rich representations of physical knowledge that can be leveraged for robotic tasks. Huang et al. [17] showed that GPT-3 and similar models could generate detailed task plans and motion descriptions when prompted with robotic scenarios, suggesting that the extensive text training of these models captures implicit understanding of physical interactions.

The development of multimodal foundation models has further advanced the field. PaLM-E [12] represents a significant milestone in embodied language models, combining visual perception with language understanding to enable robots to ground natural language instructions in visual observations. This work demonstrated that large-scale multimodal training could produce models capable of following complex, multi-step instructions in real-world environments.

2.2.2. Vision-Language-Action Models

The emergence of Vision-Language-Action (VLA) models represents the current state-of-the-art in language-guided robotics. These systems integrate visual perception, language understanding, and action generation into unified neural architectures that can be trained end-to-end.

The RT-1 system by Brohan et al. [7] pioneered the application of transformer architectures to robotic control, demonstrating that a single model could learn diverse manipulation skills from natural language instructions. The subsequent RT-2 work [8] showed that incorporating web-scale vision-language pre-training significantly improved generalization to novel objects and scenarios.

Figure AI's Helix system [13] represents the most advanced VLA model to date, capable of controlling an entire humanoid upper body from natural language instructions. Helix demonstrates several breakthrough capabilities including zero-shot manipulation of novel objects, multi-robot coordination, and complex dexterous manipulation tasks. The system uses a hierarchical architecture with a 7B parameter "System 2" for high-level reasoning and an 80M parameter "System 1" for reactive control, achieving real-time performance on embedded hardware.

The development of RoboPoint [45] addresses the challenge of spatial reasoning in VLA models by enabling precise pointing and spatial affordance prediction. This work demonstrates that vision-language models can be instruction-tuned to predict specific spatial locations for robotic actions, addressing one of the key limitations of purely language-based control systems.

2.2.3. Retrieval-Augmented Generation and Embodied Intelligence

Recent advances have explored the integration of retrieval-augmented generation (RAG) with robotic control systems. The ELLMER framework by Mon-Williams et. al. [26], published in Nature Machine Intelligence, utilizes GPT-4 and a retrieval-augmented generation infrastructure to enable robots to complete long-horizon tasks in unpredictable settings. This approach extracts contextually relevant examples from a knowledge base, producing action plans that incorporate force and visual feedback while enabling adaptation to changing conditions.

The LEGION framework by Meng et al. [25] demonstrates how robots can achieve lifelong learning capabilities by preserving and combining knowledge across sequential tasks. This work addresses the critical challenge of continual learning in robotics, showing how language embeddings can serve as a universal interface for knowledge representation and transfer.

2.2.4. Multimodal Integration and Sensor Fusion

Modern language-guided robotic systems increasingly rely on sophisticated multimodal integration approaches that combine language, vision, touch, and proprioceptive information. The survey by HAN et al. [16] provides a comprehensive overview of multimodal fusion techniques for robotic applications, highlighting the importance of attention mechanisms and cross-modal alignment for effective sensor integration.

Recent work has explored the integration of tactile and language information for manipulation tasks. The Octopi system demonstrates how large tactile-language models can enable robots to understand and reason about object properties through touch, complementing visual and linguistic information [29].

The development of 3D vision-language models represents an active area of research, with systems like ConceptFusion and ConceptGraphs enabling robots to build semantic 3D representations of environments that can be queried using natural language [16]. These approaches address the challenge of grounding language understanding in three-dimensional spatial representations.

2.3 Critique of the State of the Art

Despite remarkable progress in both language understanding and robotic control, current approaches face several fundamental limitations that constrain their real-world deployment and scalability.

2.3.1. The Hierarchical Bottleneck

Most existing language-guided robotic systems rely on hierarchical architectures that separate language understanding, task planning, and motor execution into distinct mod-

ules. While this modular design offers interpretability and enables the integration of specialized components, it introduces several critical limitations:

Semantic Gaps: Information is lost at each abstraction boundary, limiting the system's ability to leverage the rich semantic knowledge embedded in language models. The translation from natural language to symbolic representations and then to motor commands introduces opportunities for misinterpretation and reduces the system's ability to handle nuanced instructions.

Error Propagation: Failures in any component of the hierarchy can cascade through the entire system. Language understanding errors, planning failures, or execution problems can all lead to complete task failure, reducing overall system reliability.

Limited Adaptability: Hierarchical systems struggle to adapt to situations that require coordination across multiple levels of abstraction. Novel behaviors that span linguistic understanding and motor execution are difficult to discover through this decomposed approach.

2.3.2. The Manipulation Bias

A critical limitation in current research is the overwhelming focus on manipulation tasks at the expense of locomotion applications. This bias stems from several factors:

Evaluation Simplicity: Manipulation tasks often have clear success criteria and can be evaluated in controlled laboratory environments, making them attractive for research purposes.

Hardware Availability: Robotic arms are more accessible and standardized than legged platforms, lowering barriers to entry for research groups.

Safety Considerations: Stationary manipulators are generally safer and easier to work with than dynamic legged robots, particularly in academic settings.

However, this focus on manipulation has left significant gaps in our understanding of how language can guide continuous, dynamic behaviors like locomotion. Locomotion presents unique challenges that are not addressed by existing manipulation-focused approaches:

Temporal Dynamics: Locomotion requires precise temporal coordination across multiple degrees of freedom, with timing constraints that are more stringent than typical manipulation tasks.

Continuous Control: Unlike discrete manipulation actions, locomotion involves continuous control signals that must be generated at high frequency while maintaining stability.

Environmental Interaction: Locomotion involves complex interactions with terrain and environmental features that are typically abstracted away in manipulation scenarios.

2.3.3. Scalability and Data Requirements

Current language-guided robotic systems face significant scalability challenges that limit their practical deployment:

Data Hunger: Most successful systems require extensive training data, with some approaches using hundreds of thousands or millions of demonstrations. This data requirement makes it impractical to deploy these approaches to new tasks or domains without substantial data collection efforts.

Task Specificity: Despite claims of generalization, most systems demonstrate strong performance only on tasks similar to those seen during training. True zero-shot generalization to genuinely novel tasks remains elusive.

Computational Requirements: State-of-the-art VLA models require significant computational resources for both training and inference. While recent work has made progress on efficiency [14], real-time deployment on resource-constrained robotic platforms remains challenging.

2.3.4. Safety and Reliability Concerns

The deployment of language-guided robotic systems raises significant safety and reliability concerns that are not adequately addressed by current research:

Adversarial Inputs: Language-guided systems may be vulnerable to adversarial inputs designed to cause unsafe behaviors. The flexibility that makes natural language attractive as an interface also creates opportunities for misuse.

Error Modes: Unlike traditional robotic systems with well-understood failure modes, language-guided systems may fail in unpredictable ways when faced with ambiguous or malformed instructions.

Verification Challenges: The black-box nature of learned language-motor mappings makes it difficult to verify system behavior or provide guarantees about safe operation.

2.4 Proposal

To address the limitations identified in current approaches, this thesis proposes a novel end-to-end learning framework that directly maps natural language instructions to motor actions for legged robots. Our approach makes several key contributions that distinguish it from existing work:

2.4.1. Direct Language-Motor Grounding

Unlike hierarchical approaches that introduce multiple abstraction layers, our system learns direct associations between language embeddings and motor commands through reinforcement learning. This end-to-end approach offers several advantages:

Semantic Preservation: By eliminating intermediate symbolic representations, our approach preserves the rich semantic information present in language embeddings throughout the control pipeline.

Emergent Behavior Discovery: The direct optimization approach enables the system to discover novel language-motor mappings that might not be captured by predefined hierarchical decompositions.

Reduced Complexity: The unified architecture eliminates the need for careful integration of multiple specialized components, reducing system complexity and potential failure points.

Scalability: New tasks can be added to the system without modifying the model or fully retraining the system, for example, it is possible to modify the reward system and add a new command to make the robot jump.

2.4.2. Locomotion-Centered Design

This work specifically addresses the locomotion gap in current language-guided robotics research by designing our approach around the unique requirements of continuous motor control:

Temporal Coordination: Our system is designed to handle the precise temporal coordination required for stable legged locomotion, with control frequencies suitable for dynamic balance.

Continuous Action Spaces: Unlike manipulation systems that often operate with discrete action spaces, our approach generates continuous joint angle commands optimized for smooth, stable locomotion.

Environmental Adaptation: The system is trained to handle terrain variations and environmental disturbances that are fundamental challenges in legged locomotion.

2.4.3. Efficiency and Accessibility

To address scalability concerns, our approach emphasizes computational efficiency and reduced data requirements:

Compact Architecture: Our policy network is designed to be lightweight enough for real-time inference on embedded hardware while maintaining expressiveness for complex language-motor mappings.

Pre-computed Embeddings: Language processing is performed offline and cached during training, eliminating the need for real-time transformer inference.

2.4.4. Validation Through Real-World Deployment

A critical limitation of much current research is the focus on simulation-based evaluation or controlled laboratory demonstrations. Our approach includes comprehensive real-world validation:

Physical Platform: We demonstrate our approach on a physical hexapod robot platform, validating sim-to-real transfer and real-world performance.

The combination of these contributions represents a significant advance in language-guided robotics, addressing key limitations in current approaches while opening new possibilities for intuitive human-robot interaction. The end-to-end learning approach offers a pathway to more scalable, efficient, and accessible language-guided robotic systems that can operate effectively in real-world environments.

This work builds upon the substantial progress made by the robotics and AI communities while addressing critical gaps that limit the practical deployment of language-guided robotic systems. By focusing specifically on the locomotion domain and utilizing an end-to-end learning approach, we aim to demonstrate that natural language can serve as a direct, effective interface for continuous robotic control, paving the way for more intuitive and capable robotic systems.

CHAPTER 3
Problem Analysis

The development of an end-to-end language-guided reinforcement learning system for hexapod robots requires systematic analysis of technical requirements, constraints, and implementation challenges. This chapter examines the core problems that must be solved to achieve natural language control of the JetHexa platform.

3.1 Requirements Specification

The system must bridge the gap between natural language understanding and robotic motor control while operating within the constraints of embedded hardware deployment.

3.1.1. Target Platform Overview

This research is conducted using the Hiwonder JetHexa hexapod robot platform, a commercially available research and educational robotics system. The JetHexa combines an NVIDIA Jetson Nano B01 computing module with 18 degrees of freedom actuated through HX-35H intelligent serial bus servos, creating a capable platform for advanced locomotion research.

The platform weighs 2.5kg with an anodized aluminum alloy frame and operates on an 11.1V 3500mAh lithium polymer battery providing 60-90 minutes of autonomous operation. The default hexapod configuration employs tripod gait patterns coordinated through ROS Melodic running on Ubuntu 18.04 LTS. Integrated sensors include a 9-axis IMU for orientation feedback, 3D depth camera for visual perception, LiDAR for mapping and navigation, and a 6-channel microphone array for audio processing.

The computational constraints of the Jetson Nano platform (ARM Cortex-A57 quad-core CPU, 4GB shared memory, 128-core Maxwell GPU) directly inform the system architecture requirements, necessitating efficient algorithms capable of real-time performance on embedded hardware. Detailed technical specifications and capabilities are provided in Appendix A.

Figure 3.1: ROS Hexapod Robot JetHexa

3.1.2. Core Functional Requirements

The primary challenge is creating a system that can interpret diverse natural language commands and execute corresponding hexapod locomotion behaviors. The system must process commands in multiple languages while maintaining semantic consistency. Commands range from simple directional instructions like *"move forward"* to complex combinations such as *"turn left slowly while moving backward"*.

The locomotion system must coordinate all 18 degrees of freedom of the JetHexa hexapod to produce stable, efficient gaits. This requires maintaining dynamic balance during movement, executing smooth transitions between different locomotion patterns, and adapting to the gait pattern produced by the RL policy.

Integration requirements center on real-time deployment. The system must operate at 10Hz control frequency, transfer effectively from Isaac Lab simulation to real hardware, and integrate seamlessly with the ROS Melodic ecosystem running on the Jetson Nano platform.

3.1.3. Performance and Resource Constraints

Table 3.1: Critical Performance Requirements

Metric	Target	Justification
Language processing latency	<100ms	Interactive responsiveness
Policy inference time	<50ms	Real-time control stability
Memory footprint	<1GB	Jetson Nano hardware limit
Battery operational time	60-90min	JetHexa power constraints

The Jetson Nano platform imposes strict computational limits with its ARM Cortex-A57 processor and 4GB shared memory. The system must achieve real-time performance while maintaining acceptable accuracy across diverse command types and environmental conditions.

3.2 Security Analysis

Language-guided robotic systems introduce distinct security challenges, as they combine the interpretive flexibility of AI with the capacity to directly actuate physical hardware.

3.2.1. Attack Surface Analysis

A primary security concern is command injection through adversarial or malicious language inputs that induce unsafe behaviors. Unlike traditional robotic systems with a constrained set of predefined commands, natural language interfaces present a broader attack surface where subtle linguistic variations may bypass safety mechanisms.

In the present system, this risk is partially mitigated through hardware-enforced joint limits at the servo level, ensuring that physical movements remain within safe ranges regardless of high-level instructions. Also, during training the system modifies certain input values of the observation, so that the model is ready to handle malformed inputs. Additional defenses can be layered, including reinforcement learning–based safety boundaries that constrain behavior during training and run time, as well as input validation mechanisms that flag or reject obviously harmful commands, and speed limits to avoid harmful movements.

Beyond language-based inputs, the use of the Robot Operating System (ROS1) introduces further considerations. ROS1 was not designed with built-in authentication or encryption; any device connected to the same network as the ROS master node can, in principle, publish motor commands, override sensor data, or subscribe to private data streams. Consequently, the communication framework itself constitutes a significant attack vector unless access is tightly controlled at the network level.

3.2.2. Privacy and Access Control

All language interpretation is performed locally on the Jetson Nano, eliminating dependence on cloud services and reducing exposure to external interception risks. The system processes commands in real time without persistent logging, thereby minimizing data retention and associated privacy concerns.

However, the distributed ROS architecture necessitates strong access control measures. Since ROS1 lacks native mechanisms for authentication and confidentiality, security must be enforced at the network layer. In practice, this is achieved by isolating the robot on a restricted LAN or VLAN, applying firewalls to limit access to specific hosts, and using SSH tunneling when remote connections are required.

3.3 Energy and Algorithmic Efficiency Analysis

Energy efficiency directly impacts system viability, as the JetHexa's 3500mAh battery must power both computation and locomotion for practical operational periods.

3.3.1. Power Budget Analysis

The optimization strategy focuses on three areas: computational efficiency through TensorRT model optimization and PCA dimensionality reduction, motor efficiency through RL-trained smooth trajectories, and intelligent power management that scales computational resources based on task requirements.

Table 3.2: JetHexa Power Distribution

Component	Power Draw	Optimization Strategy
18× HX-35H Servos	18-54W	Smooth trajectory generation
Jetson Nano	5-10W	TensorRT optimization
Sensors (IMU, Camera)	3-6W	Selective activation
Total	**26-70W**	**Target: 25-30W average**

3.3.2. Computational Optimization

The critical bottleneck is real-time inference on resource-constrained hardware. Precomputing language embeddings eliminates transformer inference overhead when training, while PCA reduction from 384 to 128 dimensions enables effective RL learning. TensorRT optimization provides a speedup over standard PyTorch inference, making 10Hz control achievable.

Memory optimization uses pre-allocated buffers, efficient data structures, and model quantization to operate within the 4GB limit while maintaining multiple system components simultaneously.

3.4 Legal and Ethical Framework Analysis

The deployment of language-guided robots raises questions about responsibility attribution, safety standards, and societal impact that must be addressed proactively.

3.4.1. Liability and Safety Considerations

The system creates a complex responsibility chain from user commands through AI interpretation to physical actions. Clear documentation of system capabilities and limitations becomes essential for establishing appropriate liability frameworks. The implementation includes multiple safety layers: hardware joint limits, learned behavioral boundaries, and emergency stop mechanisms.

Compliance with emerging robotics safety standards requires systematic hazard analysis and risk mitigation. The system design emphasizes predictable failure modes and graceful degradation rather than unpredictable AI behavior.

3.4.2. Bias and Accessibility

Language models can exhibit cultural and linguistic biases that affect robot behavior. The multilingual training approach helps identify and mitigate these biases, while inclusive design principles ensure accessibility across different user groups and language proficiencies. The natural language interface has potential to democratize robot operation by reducing technical barriers.

3.5 Risk Analysis

System deployment faces technical, operational, and safety risks that require comprehensive mitigation strategies.

3.5.1. Critical Risk Assessment

Table 3.3: Primary Risk Analysis

Risk	Probability	Impact	Mitigation
Sim-to-real failure	Medium	High	Domain randomization
Real-time performance	Low	High	Profiling
Language misinterpretation	Medium	Medium	Training
Hardware failure	Low	Medium	Software limits
User misuse	Medium	Low	Domain randomization

The highest-impact risk is simulation-to-reality transfer failure, which could render the entire approach ineffective. Comprehensive domain randomization during training, systematic calibration procedures, and progressive hardware validation address this concern. Real-time performance risks are mitigated through extensive optimization and continuous monitoring.

3.5.2. Safety and Operational Risks

Hardware failures represent manageable risks through redundant safety systems and graceful degradation protocols. User misunderstanding poses moderate risks that clear documentation, progressive capability disclosure and good model training can address.

3.6 Identification and Analysis of Possible Solutions

Multiple architectural approaches could address language-guided locomotion, each with distinct trade-offs in complexity, performance, and capability.

3.6.1. Alternative Architecture Comparison

Table 3.4: Solution Architecture Comparison

Approach	Complexity	Flexibility	Performance	Interpretability
Hierarchical Planning	High	Low	Medium	High
LLM Integration	Very High	Very High	Low	Medium
End-to-End Learning	Medium	High	High	Low

Hierarchical approaches separate language understanding from motor control through symbolic representations. While offering interpretability, they suffer from information loss at abstraction boundaries and limited flexibility for novel commands.

Large language model integration offers superior language understanding but faces prohibitive computational requirements for embedded deployment. The models' text-based outputs are poorly suited for continuous motor control tasks.

3.6.2. End-to-End Approach Justification

The selected end-to-end approach directly integrates language embeddings into the reinforcement learning observation space, enabling unified optimization of language under-

standing and motor control. This architecture preserves semantic information throughout the pipeline while maintaining computational efficiency for embedded deployment.

Key advantages include adaptive discovery of novel language-motor mappings, semantic preservation without abstraction losses, and deployment efficiency through compact models. The approach aligns with research objectives of demonstrating direct language-guided control while meeting practical constraints of real-time embedded operation.

The training complexity and interpretability challenges are acceptable trade-offs given the innovation potential and superior performance characteristics. Comprehensive testing and validation procedures address the verification challenges inherent in learned systems.

This analysis establishes the technical foundation for the proposed end-to-end solution, identifying key requirements, constraints, and design decisions that inform the implementation approach detailed in the following chapters.

CHAPTER 4
Proposed Solution

This chapter presents the comprehensive solution for developing an end-to-end language-guided reinforcement learning system for hexapod robot locomotion. The proposed approach addresses the fundamental challenge of directly mapping natural language instructions to motor actions without requiring intermediate symbolic representations or hierarchical planning modules.

4.1 Solution Overview

The chosen solution implements a unified end-to-end learning paradigm that integrates natural language understanding directly into the reinforcement learning control loop. This approach eliminates the information loss and error propagation inherent in traditional hierarchical architectures while enabling real-time deployment on resource-constrained embedded hardware.

The solution consists of three main phases: simulation-based training using Isaac Lab with comprehensive domain randomization, systematic dataset development with multilingual coverage, and real-world deployment through a distributed ROS architecture optimized for the JetHexa hexapod platform.

4.2 Work Plan

4.2.1. Development Phases and Timeline

The project development was structured into five distinct phases, each with specific objectives, deliverables, and time allocations:

Table 4.1: Project Development Phases and Time Allocation

Phase	Objectives	Time
Phase 1	Dataset creation and language processing	Month 1
Phase 2	Simulation environment setup	Month 2
Phase 3	RL training and optimization	Month 3
Phase 4	Real-world deployment system	Month 4
Phase 5	Integration testing and validation	Month 5
Total		**5 Months**

Phase 1: Dataset Creation and Language Processing Pipeline (Month 1)

- Systematic generation of multilingual locomotion commands
- Sentence transformer integration and embedding generation
- PCA dimensionality reduction implementation and validation
- Initial language processing pipeline development

Phase 2: Simulation Environment Setup and Initial Training (Month 2)

- Isaac Lab environment configuration for JetHexa hexapod
- Custom command manager integration for language embeddings
- Basic RL training pipeline establishment
- Initial policy network architecture development

Phase 3: Reinforcement Learning Optimization and Domain Randomization (Month 3)

- Progressive curriculum development and training
- Comprehensive domain randomization implementation
- Reward function tuning for hexapod-specific requirements
- Policy optimization and convergence validation

Phase 4: Real-World Deployment System Development (Month 4)

- Distributed ROS architecture implementation
- TensorRT optimization for embedded deployment
- Hardware calibration and servo interface development
- Safety system integration and testing

Phase 5: Integration Testing and Performance Validation (Month 5)

- End-to-end system testing and validation
- Performance benchmarking and optimization
- Real-world deployment validation
- Documentation and results analysis

4.2.2. Critical Path Analysis and Risk Management

Some critical dependencies that made the development move around different phases were:

Sim-to-Real Transfer: Hardware calibration proved more complex than anticipated, requiring additional hours in Phase 4 for systematic servo mapping and validation.

Embedding Dimensionality Challenge: The discovery that full-dimensional embeddings prevented RL learning required significant replanning in Phase 3

Hardware Testing and Validation: Even on the last phase, there were still multiple bugs that weren't found until the model was deployed on the actual hardware, which required full retraining and calibration of the model and the simulation scenario, which proved challenging and moved back the project to different phases of the development process until a working model on the hardware was produced.

4.3 Budget

4.3.1. Hardware and Software Resources

Table 4.2: Hardware and Software Budget

Component	Cost (€)	Justification
Hardware:		
JetHexa Hexapod Robot	1,050	Primary research platform
JetHexa customs	200	Import tariffs
Laptop NVIDIA RTX 4070 GPU	400	RL training acceleration
Development Workstation	850	Simulation and development
2 additional servos	50	Failure tolerance
Software:		
Isaac Lab/Isaac Sim	0	Freely under individual license
ROS Melodic	0	Open source robotics framework
TensorRT	0	Included with NVIDIA ecosystem
Development Tools	0	Open source (Python, Git, etc.)
Power consumption:		
Training compute hours	50	Energy usage for training/testing
Robot charging	2	Robot battery charging
Total Hardware	**2,500**	
Total Software	**0**	
Total Infrastructure	**52**	
Grand Total	**2,552**	

4.4 Solution Design

4.4.1. System Architecture

The solution architecture implements a unified end-to-end learning paradigm that directly integrates language understanding into the robotic control loop without hierarchical decomposition.

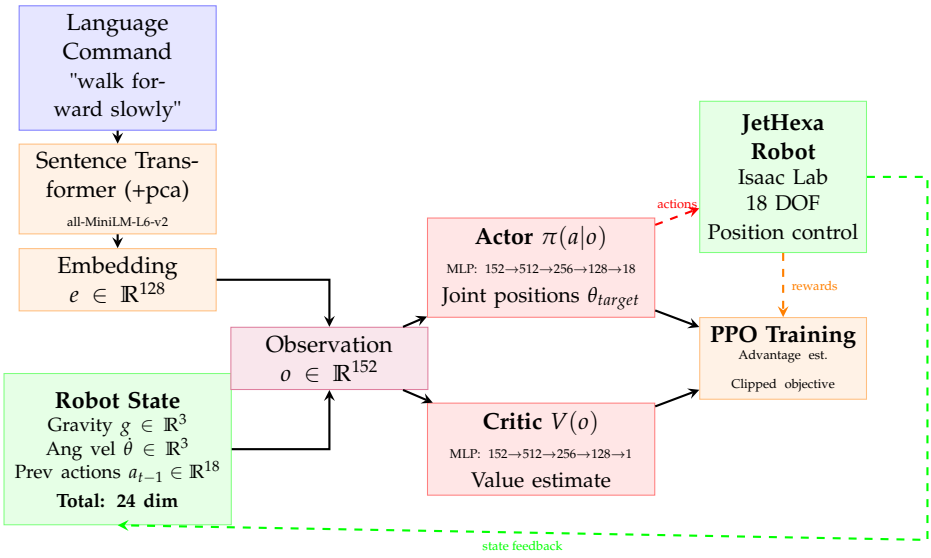

Figure 4.1: End-to-end system architecture showing direct integration of language embeddings into the reinforcement learning control loop

Core Components:

- **Language Processing Module:** Transforms natural language commands into 128-dimensional semantic embeddings using the all-MiniLM-L6-v2 sentence transformer
- **Observation Fusion:** Combines language embeddings with proprioceptive sensor data (gravity, angular velocity, action history) into a unified 152-dimensional observation vector
- **Policy Network:** Actor-critic architecture with shared feature extraction optimized for language-motor integration
- **Training Environment:** Isaac Lab simulation with 4096 parallel environments and comprehensive domain randomization
- **Deployment System:** Distributed ROS architecture for real-time execution on JetHexa hardware

Key Architectural Innovations:

- Direct semantic grounding without symbolic intermediate representations
- PCA dimensionality reduction enabling effective RL learning from language embeddings
- Multilingual capability through universal sentence embeddings
- Real-time embedded deployment through TensorRT optimization

4.4.2. Detailed Design

Language Processing Pipeline

The language processing pipeline implements a three-stage transformation from natural text to actionable semantic representations:

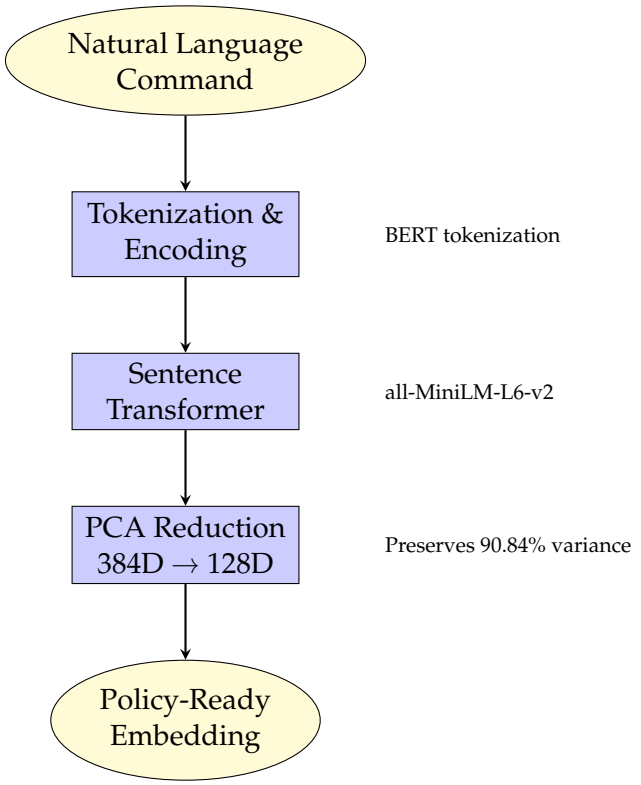

Figure 4.2: Detailed language processing pipeline with dimensionality reduction

Stage 1 - Tokenization and Encoding:

- BERT-compatible tokenization handles multilingual input
- Maximum sequence length: 128 tokens
- Padding and attention mask generation for variable-length inputs

Stage 2 - Semantic Embedding Generation:

- all-MiniLM-L6-v2 model with 22M parameters
- Mean pooling of last hidden states
- L2 normalization for consistent embedding magnitudes
- Output: 384-dimensional dense vectors

Stage 3 - Dimensionality Reduction:

- PCA fitted on 922 training commands
- Reduction from 384D to 128D (67% compression)
- Preserves 90.84% of semantic variance
- Critical for enabling RL policy learning

Reinforcement Learning Integration

The RL integration implements a unified actor-critic architecture that processes the combined observation space:

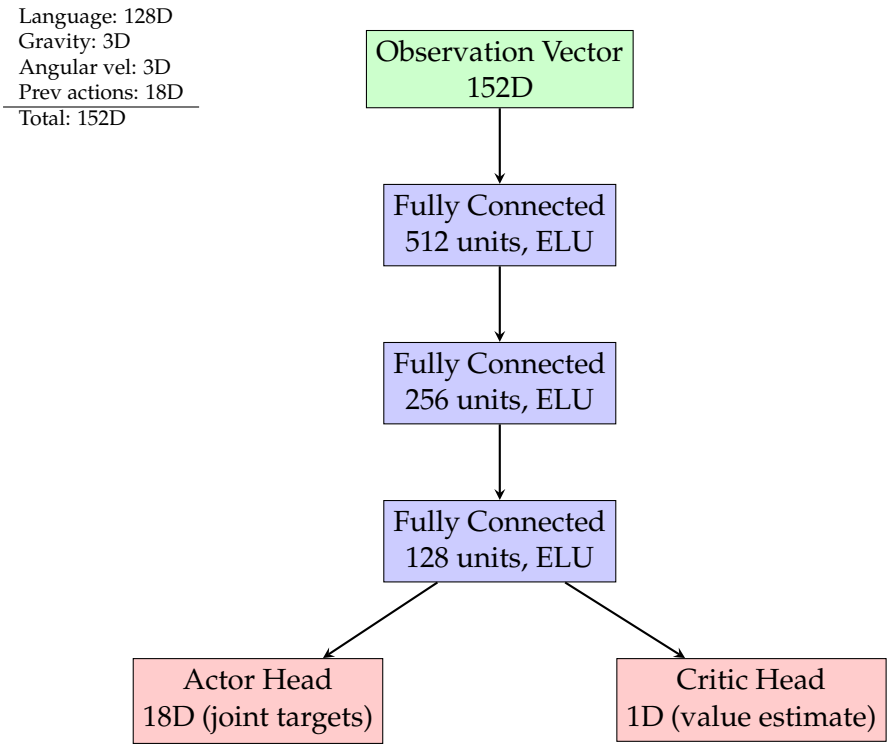

Figure 4.3: Policy network architecture with observation space decomposition

Network Architecture Details:

- Shared encoder with progressively reducing dimensionality
- ELU activation functions for smooth gradients
- Orthogonal weight initialization for stable training
- Separate actor and critic heads for PPO algorithm

Observation Space Components:

- Language embedding (128D): Semantic command representation
- Projected gravity (3D): Robot orientation relative to gravity
- Angular velocity (3D): Rotational motion feedback
- Previous actions (18D): Joint position history for smooth control

Real-World Deployment Architecture

The deployment system implements a distributed ROS architecture for real-time operation:

Figure 4.4: JetHexa Hexapod Robot

Node Responsibilities:

- **Language Command Node:** TensorRT-optimized transformer inference, embedding generation and PCA transformation
- **RL Inference Node:** Policy execution, observation fusion, and real-time joint target generation
- **Robot Driver Node:** Hardware interface, servo calibration, and safety monitoring

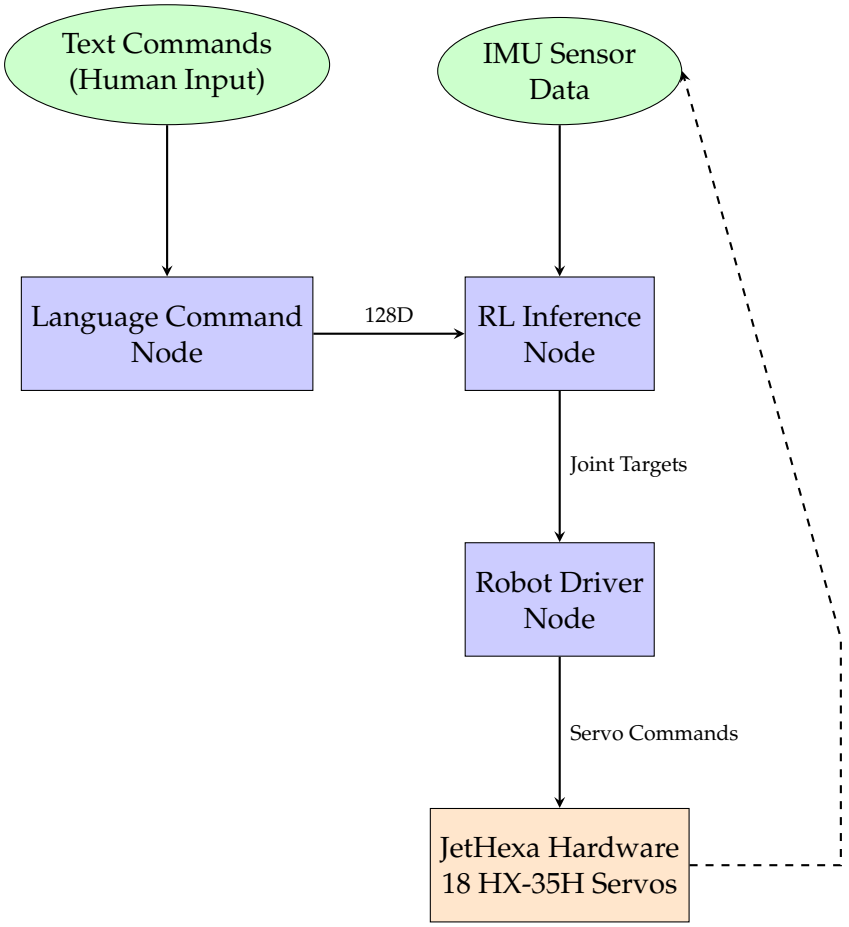

Figure 4.5: Distributed ROS deployment architecture with performance specifications

4.5 Technology Used

4.5.1. Development Environment and Tools

Table 4.3: Development Environment and Toolchain

Category	Technology	Version	Purpose
Simulation	Isaac Lab	0.40.6	RL training environment
	Isaac Sim	4.5.0	Physics simulation
	RSL-RL	2.4.4	RL
Machine Learning	Torch	2.5.1+cu121	Neural network training
	TensorRT	8.2.1.8	Inference optimization
	transformers	4.18.0-py3	Language embeddings
	scikit-learn	1.5.4-cp36	PCA dimensionality reduction
	tokenizers	0.12.1-cp36	Tokenization & encoding
Robotics	ROS Melodic	1.14.13	Robot communication
	JetHexa SDK	1.0	Hardware interface
	OpenCV	4.11.0.86	Computer vision
Development	Python	3.10.12	Primary programming language
	Python (Jethexa)	3.6	Robot inference
	Git	2.34.1	Version control
	VS Code	1.103.2	Development environment

4.5.2. Hardware Platform Selection

Training Hardware:

- Laptop NVIDIA RTX 4070 GPU: 4608 CUDA cores, 8GB GDDR6X memory
- 13th Gen Intel® Core™ i7-13620H × 16
- 32GB DDR4 RAM: High-bandwidth memory for parallel environments
- Training performance: >280,000 simulation steps/second with 4096 parallel environments

Deployment Hardware:

- NVIDIA Jetson Nano B01: ARM Cortex-A57 quad-core, Maxwell GPU (128 cores)
- 4GB LPDDR4 shared memory: Constrained but sufficient for optimized models
- 32GB microSD storage: Adequate for system and model storage

Robot Platform:

- JetHexa hexapod: 18 DOF with HX-35H intelligent servos
- IMU sensor: 9-axis orientation and motion sensing
- 11.1V LiPo battery: 3500mAh capacity for 60-90 minutes operation
- Integrated safety systems: Emergency stop, joint limits, graceful startup

4.5.3. Software Architecture Decisions

Simulation Platform Selection:

Isaac Lab was chosen over alternatives (Gazebo, PyBullet, MuJoCo) for several critical advantages:

- GPU-accelerated parallel simulation (4096 environments)
- High-fidelity physics simulation with accurate contact modeling
- Proven sim-to-real transfer capabilities for legged robotics

Deployment Framework Selection:

ROS Melodic was selected despite being an older version due to:

- Native compatibility with Ubuntu 18.04 on Jetson Nano
- Proven real-time performance for robotics applications
- Extensive documentation and community support
- Already installed on Jethexa Jetson Nano

4.5.4. Optimization and Performance Strategies

Training Optimization:

- Parallel environment scaling: 4096 simultaneous simulations
- GPU memory optimization: Efficient tensor operations and batch processing
- Domain randomization: Comprehensive parameter variation for robust sim-to-real transfer

Deployment Optimization:

- TensorRT model optimization: inference speedup through graph optimization
- Asynchronous processing: Non-blocking language processing and motor control
- Memory management: Pre-allocated buffers and efficient data structures

This comprehensive technology selection and implementation strategy ensures optimal performance across the entire development and deployment pipeline while maintaining practical feasibility within resource constraints.

CHAPTER 5
Implementation

This chapter provides a comprehensive account of the implementation process for the end-to-end language-guided reinforcement learning system. The development followed a systematic approach across five phases, progressing from foundational dataset creation through sophisticated simulation training to real-world deployment on the JetHexa hexapod platform. Each phase addressed specific technical challenges while building toward a complete working system capable of interpreting natural language commands and executing corresponding locomotion behaviors.

5.1 Development Methodology

The implementation process was structured into five distinct phases, each with specific objectives and deliverables:

1. **Dataset Creation and Language Processing Pipeline** (Month 1)
2. **Simulation Environment Development** (Month 2)
3. **Reinforcement Learning Training and Optimization** (Month 3)
4. **Real-World Deployment System Architecture** (Month 4)
5. **Integration Testing and Performance Validation** (Month 5)

Each phase included systematic testing and validation to ensure reproducibility and enable iterative improvement. The methodology emphasized modular design, comprehensive logging, and systematic debugging to address the inherent complexity of integrating natural language processing with robotic control.

5.2 Phase 1: Dataset Creation and Language Processing Pipeline

5.2.1. Multilingual Command Dataset Development

The foundation of the language-guided system required creating a comprehensive dataset that could capture the semantic diversity of natural locomotion commands while maintaining precise velocity target mappings. The dataset underwent four major iterations based on systematic analysis of training performance and policy behavior patterns.

The final dataset contains 922 commands across multiple languages (English, Spanish, German, French, Italian, Chinese...), with each command mapped to specific velocity targets appropriate for hexapod locomotion:

```
{
    "id": 148,
    "phrase": "move forward slowly",
    "linear_x": 0.3,     # m/s, forward velocity
    "linear_y": 0.0,     # m/s, lateral velocity
    "angular_z": 0.0,    # rad/s, rotational velocity
    "embedding_file": "embeddings/148.npy"
}
```

Listing 5.1: Command dataset structure example

The velocity ranges were carefully selected based on JetHexa hardware constraints for most commands, although maximum range was exaggerated to push robot boundaries during training: linear X velocity [-2.0, 2.0] m/s, linear Y velocity [-2, 2] m/s, and angular Z velocity [-2.0, 2.0] rad/s.

5.2.2. Dataset Generation Using Large Language Models

The creation of a diverse and representative command dataset posed significant challenges in capturing natural language variation while maintaining precise velocity mappings. To address this challenge, we employed a systematic approach using Claude 3.5 (Anthropic), a state-of-the-art large language model, to generate linguistically diverse commands through iterative prompting strategies.

Iterative Dataset Expansion Process

The dataset generation followed a structured four-phase approach, with each phase addressing specific limitations identified through training analysis:

Phase 1: Core Command Generation (Commands 0-88) Initial dataset creation focused on establishing fundamental movement patterns across multiple languages. The generation prompt specified:

```
# Prompt template for core command generation
"Generate movement commands for a robot with the following structure:
- Include variations: formal, informal, typos, abbreviations
- Languages: English, Spanish, German, French, Italian, Chinese
- Movement types: forward, backward, turn, stop, lateral
- Velocity mappings: linear_x, linear_y, angular_z
- Format: CSV with '|' delimiter to avoid text conflicts"
```

Listing 5.2: Initial generation prompt structure

This phase produced 89 foundational commands covering basic locomotion primitives with initial multilingual support.

Phase 2: Variation Expansion (Commands 89-351) Analysis of initial training revealed limited linguistic diversity. The second generation phase emphasized:

- Systematic typo injection ("walk forward" → "wlk forwrd")
- Colloquial expressions ("yo dawg move it", "aight turn left")
- Formal/technical language ("execute primary locomotive protocol")

- Emotional modifiers ("please", "urgently", "carefully")
- Compound movements ("curve left", "diagonal forward right")

Phase 3: Multilingual and Semantic Expansion (Commands 352-600) The third phase addressed cross-linguistic generalization:

```
# Languages systematically added with movement variations
French: "avance s'il vous pla t", "tournez    gauche"
German: "vorw rts gehen", "nach links drehen"
Italian: "vai avanti", "gira a sinistra"
Portuguese: "ir para frente", "virar    esquerda"
```

Listing 5.3: Multilingual expansion strategy

Additionally, this phase introduced:

- Novel movement verbs ("hustle", "saunter", "meander")
- Directional specifications ("move @ 30 degrees left")
- Contextual commands ("patrol", "explore", "investigate")

Although some of this commands suggest a more complex movement pattern, this verbs usually were mapped to simple velocities, for example, "investigate" just maps to a regular forward movement, and "move @ 30 degrees left" just maps to a certain velocity when turning left.

Phase 4: Forward Movement Augmentation (Commands 601-922) Training analysis revealed systematic underperformance in forward movement execution. The final phase specifically addressed this imbalance through targeted augmentation, also given that forward movement can help derive in other movements and policies once it has been learned:

```
# Targeted generation for forward movement variations
"Generate 300+ forward movement commands emphasizing:
- Speed variations: [0.2, 0.5, 1.0, 2.0] m/s
- Linguistic diversity: single words to complex phrases
- Urgency levels: 'GO GO GO' to 'gently advance'
- Semantic richness: 'proceed', 'advance', 'forge ahead'"
```

Listing 5.4: Forward movement augmentation focus

This phase increased forward commands from approximately 30% to 45% of the total dataset, providing the model with stronger signal for this critical movement type.

Quality Control and Validation

Each generation phase included systematic validation:

1. **Velocity Mapping Verification**: Ensuring semantic consistency between command intent and numerical targets
2. **Linguistic Diversity Analysis**: Measuring vocabulary richness and syntactic variation
3. **Cross-linguistic Balance**: Maintaining representative samples across languages
4. **Typo Realism**: Validating that introduced errors reflected natural typing patterns

Final Dataset Composition

The final dataset of 922 commands achieved the following distribution:

Language Distribution:

- English: 598 commands (65%) - Primary language for maximum coverage
- Spanish: 142 commands (15%) - Second most represented for Romance language coverage
- German: 76 commands (8%) - Germanic language representation
- French: 68 commands (7%) - Additional Romance language variation
- Other (Italian, Chinese, Portuguese): 38 commands (4%) - Exploratory multilingual support

Table 5.1: Dataset language statistics

Language	Value
Total commands	922
English	598 (65%)
Spanish	142 (15%)
German	76 (8%)
French	68 (7%)
Other	38 (4%)

Movement Type Distribution:

- Forward movements: 412 commands (45%) - Emphasized due to training requirements
- Turn commands: 198 commands (22%) - Balanced left/right rotations
- Backward movements: 125 commands (14%) - Reverse locomotion variants
- Stop commands: 89 commands (10%) - Critical for safety and control
- Lateral movements: 62 commands (7%) - Sideways motion capabilities
- Combined movements: 36 commands (4%) - Complex multi-axis commands

Table 5.2: Dataset movement distribution

Movement Distribution	Count
Forward commands	412 (45%)
Backward commands	125 (14%)
Turn commands	198 (22%)
Stop commands	89 (10%)
Lateral commands	62 (7%)
Combined movements	36 (4%)

This systematic approach to dataset generation leveraged the language model's capability to produce naturalistic variations while maintaining precise control over the distribution and characteristics of generated commands. The iterative refinement process, guided by training performance analysis, resulted in a dataset that balances linguistic diversity with practical requirements for robotic control training. The heavy emphasis on forward movements (45%) in the final dataset reflects the empirical finding that this fundamental locomotion mode required additional training signal for reliable execution.

5.2.3. Test Set Design for Generalization Evaluation

To rigorously evaluate the model's language understanding capabilities beyond memorization, we designed a comprehensive test set of 120 commands that systematically probes different aspects of linguistic generalization. The test set was generated using the same language model (Claude 3.5) but with carefully crafted prompts to ensure distinct characteristics from the training data.

Test Set Generation Strategy

The test set generation employed a stratified approach to create commands across ten distinct categories, each designed to evaluate specific generalization capabilities:

Category Definitions and Generation Process:

1. **Close Variations (25 commands)**: Commands semantically similar to training data but with different phrasing. Generation focused on synonym substitution and grammatical restructuring while maintaining core meaning.

2. **Novel Concepts (19 commands)**: Commands expressing movement ideas absent from training data, such as "navigate to my position" or "return to start". These test compositional understanding and semantic extrapolation, although many of them cannot be easily mapped to simple velocities given their complex meaning, so they can easily be misinterpreted.

3. **Novel Verbs (14 commands)**: Movement verbs not present in training ("plow", "scurry", "tiptoe", "glide") to evaluate semantic grounding of unfamiliar locomotion descriptors.

4. **Multilingual (10 commands)**: New phrases in languages present in training but with unseen vocabulary and structures, testing cross-linguistic generalization.

5. **Slang/Informal (10 commands)**: Contemporary colloquialisms ("yo dawg move it", "no cap go straight") absent from training data, evaluating robustness to register variation.

6. **Formal/Technical (10 commands)**: Overly formal language ("execute primary locomotive protocol", "commence sinistral rotation") testing understanding of technical register.

7. **Typo Variations (10 commands)**: Realistic typing errors following different patterns than training typos, assessing robustness to input noise.

8. **Uncertain/Hedged (8 commands)**: Commands with uncertainty markers ("umm like go forward i guess", "maybe walk ahead?") not present in training.

9. **Abbreviations (7 commands)**: Shortened forms ("fwd", "bkwd", "lft") testing understanding of common abbreviations.

10. **Emphasis/Urgency (7 commands)**: Commands with strong emotional markers ("GO GO GO GO GO", "STOP!!!") evaluating response to urgency.

Movement Distribution Balancing

The test set movement distribution was designed to probe known weaknesses while maintaining sufficient coverage:

```
movement_distribution = {
    "forward": 45,      # 38% - Emphasize problematic movement
    "backward": 12,     # 10% - Proportional representation
    "turns": 30,        # 25% - Split left/right equally
    "stop": 15,         # 13% - Critical safety command
    "lateral": 8,       # 7%  - Less common movement
    "combined": 10      # 8%  - Complex multi-axis commands
}
```

Listing 5.5: Test set movement distribution

The forward movement emphasis (38%) in the test set specifically targets the identified training weakness, providing additional evaluation coverage for this critical locomotion mode and also adapting to the training imbalance introduced in the training set.

Quality Assurance Measures

Test set generation included several validation steps:

1. **Training Set Exclusion**: Automated checking ensured no test command appeared verbatim in the training set

2. **Semantic Distance Verification**: Each test command was verified to maintain appropriate semantic distance from training examples within its category

3. **Velocity Mapping Consistency**: Target velocities were assigned following the same semantic rules as training data to ensure fair evaluation

4. **Cross-Category Balance**: Commands were distributed to prevent category-specific biases in movement types

Test Set Characteristics

The final test set of 120 commands achieved:

- Balanced representation across 10 linguistic categories
- Movement distribution targeting known model weaknesses
- Average command length: 3.1 words (slightly higher than training)
- Language distribution: 92% English, 8% other languages
- Novel vocabulary coverage: 42% of commands contain words absent from training

This systematic approach to test set generation ensures comprehensive evaluation of the model's true language understanding capabilities, distinguishing between memorization of training patterns and genuine semantic comprehension. The stratified design enables detailed analysis of specific generalization failures, providing insights for future improvements in language-guided robotic control systems.

5.2.4. Principal Component Analysis Optimization

A critical implementation challenge emerged during initial training attempts: the full 384-dimensional embeddings from the sentence transformer model created severe learning bottlenecks for the RL policy. The high dimensionality prevented effective learning of language-motor associations, necessitating systematic dimensionality reduction analysis.

PCA Variance Analysis Methodology:

To determine the optimal number of principal components, we conducted comprehensive variance preservation analysis on the complete training dataset of 922 embeddings. Principal Component Analysis was fitted on all training embeddings to analyze the cumulative explained variance across different dimensionality reduction targets.

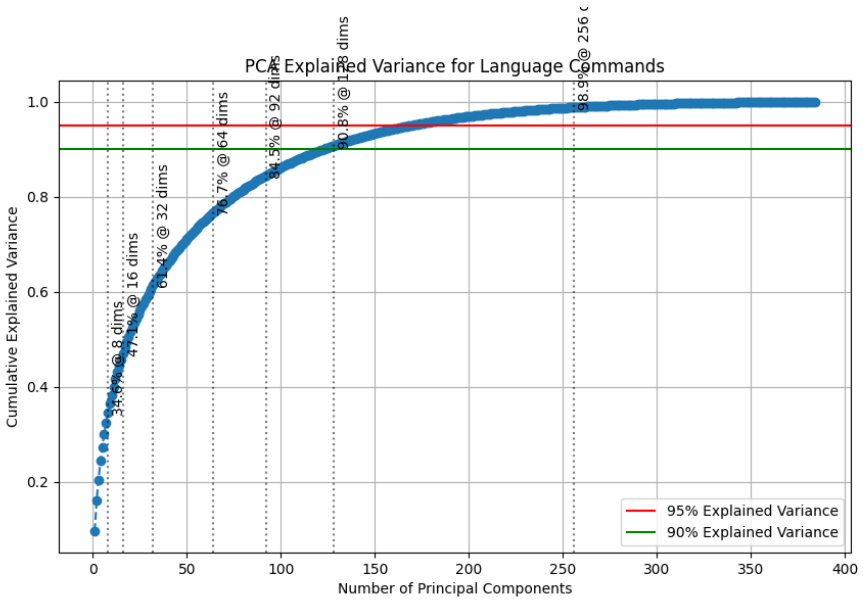

Figure 5.1: Cumulative explained variance as a function of principal components for language command embeddings. The plot shows the trade-off between dimensionality reduction and information preservation, with key decision points at 90% and 95% variance thresholds.

Variance Preservation Analysis:

The systematic analysis revealed the following variance preservation characteristics across different component counts:

Table 5.3: PCA Dimensionality Reduction Analysis

Components	Variance Preserved	Compression Ratio
8	34.59%	48:1
16	47.09%	24:1
32	61.37%	12:1
64	76.70%	6:1
92	84.52%	4.2:1
128	90.84%	3:1
256	98.89%	1.5:1

Component Selection Rationale:

The selection of 128 components was based on several converging factors:

- **Variance Preservation Threshold:** 128 components preserve 90.84% of the original semantic variance, exceeding the commonly used 90% threshold for maintaining information content while achieving substantial dimensionality reduction.
- **RL Learning Feasibility:** Preliminary experiments indicated that dimensions above 128 continued to create learning difficulties for the RL policy, while dimensions below 128 showed degraded language understanding performance.
- **Computational Efficiency:** The 3:1 compression ratio (384→128 dimensions) provides significant computational benefits during training and inference while maintaining semantic richness.

Implementation Strategy:

The PCA transformation was implemented with careful attention to proper train/test separation:

```
# Fit PCA exclusively on training data
pca = PCA(n_components=128)
pca.fit(training_embeddings_384d)

# Transform both training and test embeddings using same model
reduced_train = pca.transform(training_embeddings_384d)
reduced_test = pca.transform(test_embeddings_384d)

# Verify variance preservation
variance_preserved = sum(pca.explained_variance_ratio_)
print(f"Variance preserved: {variance_preserved:.4f}")  # 0.9084
```

Listing 5.6: PCA implementation with proper experimental design

This methodological approach ensured that no information from the test set influenced the dimensionality reduction, maintaining proper experimental validation while enabling effective RL learning from semantically meaningful, compact representations.

The 128-component solution represents an optimal balance between semantic preservation and learning feasibility, enabling the RL policy to successfully learn direct language-motor mappings while maintaining the rich semantic content necessary for diverse command interpretation.

5.3 Phase 2: Simulation Environment Development

5.3.1. Isaac Lab Framework Architecture

Isaac Lab provides a GPU-accelerated robotics simulation framework built on NVIDIA Omniverse. The framework represents a significant advancement over traditional CPU-based simulators by leveraging GPU parallelization to run thousands of environments simultaneously. This massive parallelization is essential for reinforcement learning, which requires millions of environment interactions to learn complex behaviors.

The simulation architecture operates on several key principles:

Parallel Environment Execution: Instead of running environments sequentially, Isaac Lab spawns thousands of identical environments on GPU memory, allowing simultaneous physics simulation and data collection. Each environment maintains its own state while sharing computational resources efficiently.

Vectorized Operations: All computations (physics updates, reward calculations, observations) are vectorized across environments using PyTorch tensors, enabling efficient GPU utilization and avoiding CPU-GPU memory transfers.

Scene Management: The framework manages complex scenes with multiple assets, terrains, and sensors while maintaining real-time performance through optimized memory layout and efficient data structures.

```
@configclass
class LocomotionLanguageRoughEnvCfg(ManagerBasedRLEnvCfg):
    # Scene settings for massive parallelization
    scene: MySceneCfg = MySceneCfg(num_envs=4096, env_spacing=2.5)

    # Physics simulation parameters
    def __post_init__(self):
        self.decimation = 20          # 20:1 decimation ratio
        self.episode_length_s = 20.0  # 20-second episodes
        self.sim.dt = 0.005          # 5ms physics timestep (200Hz)
        self.sim.render_interval = self.decimation  # Render at 10Hz
```

Listing 5.7: Isaac Lab environment configuration

This configuration enables 4096 parallel environments with 200Hz physics simulation decimated to 10Hz control frequency, matching the target deployment specifications.

5.3.2. JetHexa Robot Model Integration

The JetHexa hexapod model implementation required precise matching of physical properties to enable effective sim-to-real transfer. Hiwonder provides an Unified Robot Description Format (URDF) file containing a visual model of the robot upon request to the technical support team, although the actual accurate physics parameters of the joints was not available. It was necessary to include manually accurate mass distribution, joint limits, and actuator characteristics based on the HX-35H servo specifications. Finally it was mandatory to convert this file to Universal Scene Description (USD) format so that it could be understood by Isaac Sim.

```
JETHEXA_CFG = ArticulationCfg(
    spawn=sim_utils.UsdFileCfg(
        usd_path="models/jethexa.usd",
        activate_contact_sensors=True,
        rigid_props=sim_utils.RigidBodyPropertiesCfg(
            disable_gravity=False,
```

```
            max_linear_velocity=1000.0,
            max_angular_velocity=1000.0,
        ),
    ),
    init_state=ArticulationCfg.InitialStateCfg(
        pos=(0.0, 0.0, 0.15),  # Starting height for stability
        joint_vel={".*": 0.0},
    ),
    actuators={
        "leg_coxa": ImplicitActuatorCfg(
            joint_names_expr=["coxa_joint_.*"],
            effort_limit=3.5,  # HX-35H servo torque limit
            velocity_limit=6.0,
            stiffness={"coxa_joint_.*": 4.61194},
            damping={"coxa_joint_.*": 0.00184},
        ),
        # Similar configurations for femur and tibia joints
    },
)
```

Listing 5.8: JetHexa robot configuration with actuator modeling

The actuator configuration uses implicit position control with stiffness values to simulate servo behavior accurately. The effort and velocity limits match the HX-35H servo specifications to ensure realistic force and speed constraints.

The stiffness parameter controls how strongly the actuator resists deviation from its target position, while damping determines how quickly oscillations around the target position are reduced. However, for more accurate simulation, it would be beneficial to characterize the actual servo response through physical testing of the HX-35H servos, measuring their step response, overshoot, and settling time to derive more realistic stiffness and damping parameters that better represent the true dynamic behavior of the hardware.

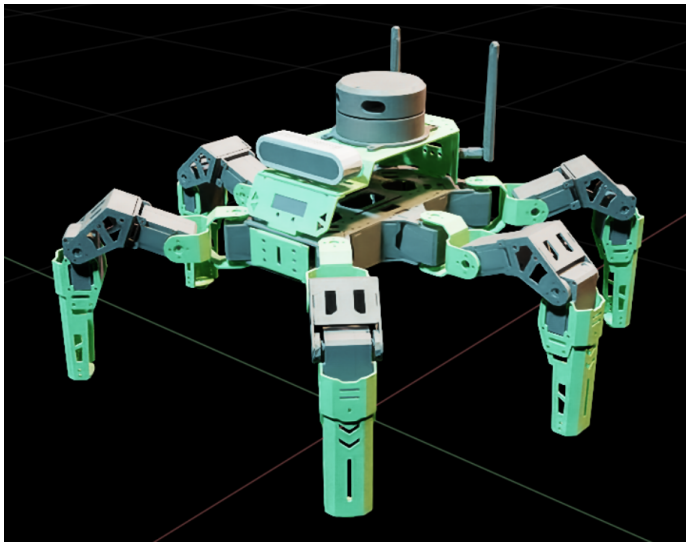

Figure 5.2: JetHexa USD model visualization on Isaac Sim

5.3.3. Language Command System Integration

The language command system represents the core innovation enabling direct integration of semantic understanding into the RL training loop. The system loads the multilin-

gual dataset, applies PCA transformation, and provides embeddings to the policy during training.

```python
class LanguageCommand(CommandTerm):
    def __init__(self, cfg: LanguageCommandCfg, env: ManagerBasedRLEnv):
        super().__init__(cfg, env)

        # Load command dataset
        self.commands = self._load_commands_from_csv(cfg.csv_path)

        # Setup PCA transformation from training data
        if cfg.use_pca and not cfg.pca_preprocessed_dir:
            self._fit_pca_from_training_data()

        # Pre-load all embeddings for efficient GPU training
        self._preload_embeddings_to_gpu()

    def _fit_pca_from_training_data(self):
        """Fit PCA model exclusively on training embeddings."""
        train_embeddings = []
        for cmd in self.train_commands:
            embedding = np.load(self.embeddings_base_dir / cmd['embedding_file'
                ])
            train_embeddings.append(embedding)

        self.pca_transformer = PCA(n_components=self.cfg.pca_dimension)
        self.pca_transformer.fit(np.array(train_embeddings))

    def compute(self, dt: float) -> torch.Tensor:
        """Return current language embeddings for all environments."""
        return self.current_embeddings
```

Listing 5.9: Language command manager implementation

5.3.4. Observation Space Architecture

The observation space design integrates heterogeneous data types into a unified representation suitable for neural network processing. The 152-dimensional observation vector combines semantic information from language with proprioceptive robot state.

```python
@configclass
class ObservationsCfg:
    @configclass
    class PolicyCfg(ObsGroup):
        # Robot orientation via projected gravity (3D)
        projected_gravity = ObsTerm(
            func=mdp.projected_gravity,
            params={"asset_cfg": SceneEntityCfg("robot")},
            noise=Unoise(n_min=-0.05, n_max=0.05)
        )

        # Angular velocity from IMU (3D)
        imu_ang_vel = ObsTerm(
            func=mdp.imu_ang_vel,
            params={"asset_cfg": SceneEntityCfg("imu")},
            noise=Unoise(n_min=-0.2, n_max=0.2)
        )

        # Language command embedding (128D)
        language_embedding = ObsTerm(
            func=mdp.generated_commands,
            params={"command_name": "language_command"}
        )
```

```
24
25          # Previous joint actions for temporal consistency (18D)
26          actions = ObsTerm(
27              func=mdp.last_joint_pos_target,
28              params={"asset_cfg": SceneEntityCfg("robot")},
29              clip=(-50, 50)
30          )
```

Listing 5.10: Observation space configuration

This configuration creates a unified observation space where the policy learns to associate language semantics with current robot state and required motor actions.

5.3.5. Terrain Generation and Domain Randomization

Training occurs on rough terrain generated using Isaac Lab's procedural terrain system. The terrain provides essential domain randomization for robust policy learning and sim-to-real transfer.

```
terrain = TerrainImporterCfg(
    prim_path="/World/ground",
    terrain_type="generator",
    terrain_generator=ROUGH_TERRAINS_CFG,
    max_init_terrain_level=5,
    physics_material=sim_utils.RigidBodyMaterialCfg(
        static_friction=1.0,
        dynamic_friction=1.0,
    ),
)
```

Listing 5.11: Terrain configuration for domain randomization

The rough terrain includes multiple challenging surface types:

- Flat areas for basic movement learning
- Sloped surfaces (±15 degrees) for stability training
- Step patterns with varying heights (0.05-0.2m)
- Rough patches with irregular surfaces
- Stairs and platforms for climbing behavior

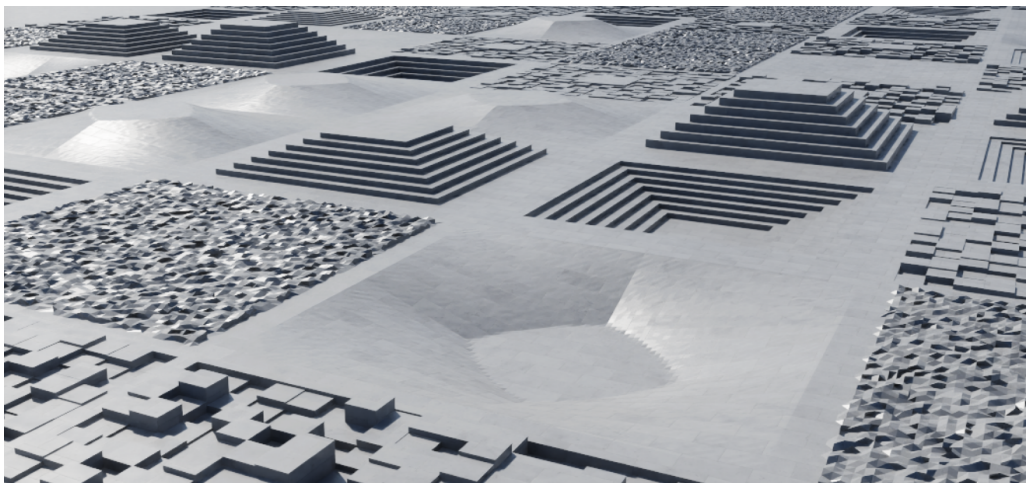

Figure 5.3: Rough terrain environment used in training.

Additional domain randomization includes physics parameter variations and external disturbances:

```
# Friction randomization
physics_material = EventTerm(
    func=mdp.randomize_rigid_body_material,
    mode="startup",
    params={
        "static_friction_range": (0.5, 0.99),
        "dynamic_friction_range": (0.3, 0.99),
        "num_buckets": 64,
    },
)

# Mass distribution randomization
add_base_mass = EventTerm(
    func=mdp.randomize_rigid_body_mass,
    mode="startup",
    params={
        "mass_distribution_params": (-0.5, 0.5),
        "operation": "add",
    },
)
```

Listing 5.12: Physics domain randomization

5.3.6. Foundation from Existing Velocity Locomotion Task

The development of the language-guided locomotion system built upon the existing velocity-based locomotion framework already available in Isaac Lab. This foundation provided a crucial starting point for understanding the system architecture and enabled a systematic transition from traditional command-based control to natural language guidance.

Isaac Lab Velocity Task Analysis

Isaac Lab includes a comprehensive velocity locomotion example designed for legged robots, which served as the architectural foundation for this research. The existing velocity task implements a complete RL training pipeline with the following key characteristics:

```
@configclass
class PolicyCfg(ObsGroup):
    """Observations for policy group in velocity task."""

    base_lin_vel = ObsTerm(func=mdp.base_lin_vel, noise=Unoise(n_min=-0.1,
        n_max=0.1))
    base_ang_vel = ObsTerm(func=mdp.base_ang_vel, noise=Unoise(n_min=-0.2,
        n_max=0.2))
    projected_gravity = ObsTerm(func=mdp.projected_gravity, noise=Unoise(n_min
        =-0.05, n_max=0.05))
    velocity_commands = ObsTerm(func=mdp.generated_commands, params={"
        command_name": "base_velocity"})
    joint_pos = ObsTerm(func=mdp.joint_pos_rel, noise=Unoise(n_min=-0.01, n_max
        =0.01))
    joint_vel = ObsTerm(func=mdp.joint_vel_rel, noise=Unoise(n_min=-1.5, n_max
        =1.5))
    actions = ObsTerm(func=mdp.last_action)
    height_scan = ObsTerm(func=mdp.height_scan, params={"sensor_cfg":
        SceneEntityCfg("height_scanner")})
```

Listing 5.13: Velocity task observation space configuration

The velocity task utilizes a `UniformVelocityCommandCfg` that generates random target velocities within specified ranges, which the RL policy learns to track through reward optimization. This approach demonstrated the feasibility of training locomotion policies using Isaac Lab's parallel simulation capabilities.

Architectural Transformation to Language Commands

The transformation from velocity-based to language-guided control required fundamental changes to the command generation and observation space architecture:

- **Command System Replacement:** Substitution of `UniformVelocityCommandCfg` with our custom `LanguageCommandCfg` for semantic command generation
- **Observation Space Reduction:** Elimination of explicit velocity commands in favor of language embeddings, but removal of multiple unavailable sensors on the actual JetHexa robot (like the height scanner).
- **Sensor Simplification:** Streamlined sensor suite focusing on essential proprioceptive feedback available on the robot (gravity projection, angular velocity) rather than comprehensive state estimation
- **Reward Function Adaptation:** Modification of tracking rewards to compare robot behavior against language-derived velocity targets rather than directly specified velocities

This transformation preserved the core RL training infrastructure while enabling semantic command interpretation, demonstrating that language understanding could be integrated into existing locomotion frameworks without fundamental algorithmic changes.

Proof of Concept: Humanoid G1 Language Locomotion

Prior to the main JetHexa implementation, a proof-of-concept study was conducted using the Unitree G1 humanoid robot as part of coursework for "Aplicaciones de Reconocimiento de Formas y Aprendizaje Automático (ARA)." This preliminary investigation validated the feasibility of language-guided locomotion on a more complex platform. Some of the slides of this work can be found in Appendix B.

G1 Platform Characteristics:

- Height: 1.20 meters, Mass: 35 kg
- Degrees of Freedom: 37 actuated joints
- Observation space: 691 dimensions (including comprehensive joint state and height scanning)
- Action space: 37-dimensional joint position targets

Experimental Setup: The G1 study employed a progressive curriculum approach with 922 multilingual commands, utilizing the same sentence transformer later applied to the JetHexa system. The larger observation space (691D vs. 152D) required more

Figure 5.4: Parallel training of G1 robot following multiple movement commands

complex network architectures but validated the core language integration principles and the larger network enabled the use of the sentence embeddings directly (without PCA).

Key Findings:

- Achieved 58% success rate on diverse test commands spanning 10 linguistic categories
- Training was effective using a progressive curriculum approach

The G1 proof-of-concept provided critical validation for the language-guided approach while informing design decisions for the subsequent JetHexa implementation.

This preliminary work established confidence in the end-to-end approach that became central to the JetHexa system design.

Figure 5.5: G1 robot following the command "proceed in a forward direction" downstairs

5.4 Phase 3: Reinforcement Learning Training

5.4.1. Proximal Policy Optimization Algorithm

The system employs Proximal Policy Optimization (PPO), a policy gradient algorithm specifically designed for continuous control tasks like robotic locomotion. PPO addresses key challenges in policy gradient methods by using a clipped surrogate objective that prevents destructively large policy updates while maintaining sample efficiency.

PPO Algorithm Overview:

PPO operates through iterative policy improvement cycles:

1. **Experience Collection:** Run current policy in environment to collect trajectory data
2. **Advantage Estimation:** Compute advantage estimates using Generalized Advantage Estimation (GAE)
3. **Policy Update:** Optimize clipped surrogate objective using mini-batch gradient descent
4. **Value Function Update:** Train critic network to predict value estimates

The clipped objective function prevents policy updates that are too large, maintaining training stability:

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ is the advantage estimate.

5.4.2. Training Infrastructure and Parallelization

The training leverages Isaac Lab's GPU-accelerated parallel simulation to achieve unprecedented data collection rates. With 4096 environments running simultaneously on an RTX 4070 GPU, the system collects over 14,000 control steps per second (because of the 10Hz control system).

Training Execution:

```
./isaaclab.sh -p scripts/reinforcement_learning/rsl_rl/train.py \
    --task Isaac-Language-Rough-JetHexa-v0 \
    --num_envs 4096 \
    --headless
```

Listing 5.14: Training command and configuration

The training process configuration:

- **Environments:** 4096 parallel instances
- **Episode Length:** 20 seconds (200 steps at 10Hz)
- **Training Iterations:** 10,000 total
- **Data Collection:** 20 steps per iteration per environment
- **Training Time:** Approximately 20 hours

5.4.3. Neural Network Architecture and Control Loop Implementation

The neural network architecture forms the core of the end-to-end learning system, directly mapping multimodal observations to joint position targets for the hexapod's 18 degrees of freedom. Understanding the precise implementation details reveals critical insights about how language semantics translate into physical motor commands.

Network Architecture and Data Flow

The policy employs a shared actor-critic architecture optimized for continuous control tasks:

```
policy = RslRlPpoActorCriticCfg(
    init_noise_std=1.0,
    actor_hidden_dims=[512, 256, 128],
    critic_hidden_dims=[512, 256, 128],
    activation="elu"
)
```

Listing 5.15: Neural network configuration

Input Processing (152-dimensional observation vector):

The network receives a carefully constructed observation vector containing:

- **Language embedding (128D):** PCA-reduced semantic representation from sentence transformer
- **Projected gravity (3D):** Robot orientation relative to gravity vector in body frame
- **Angular velocity (3D):** IMU-measured rotational motion in world frame
- **Previous actions (18D):** Last commanded joint positions for temporal consistency

Output Generation (18-dimensional action vector):

The actor network outputs 18 continuous values representing target joint positions in radians. These values are processed through the action transformation pipeline:

```
joint_pos = mdp.JointPositionActionCfg(
    asset_name="robot",
    joint_names=[".*"],
    scale=0.25,
    use_default_offset=True,
    clip={".*": (-2.1, 2.1)}
)
```

Listing 5.16: Action processing configuration

The mathematical transformation applied to raw network outputs is:

$$\theta_{\text{target}} = \theta_{\text{default}} + 0.25 \cdot \text{clip}(\text{raw\_output}, -2.1, 2.1)$$

where θ_{default} represents the robot's neutral standing configuration and the 0.25 scaling factor reduces the action magnitude for finer control.

Temporal Control Loop and Decimation

The system operates with a sophisticated multi-rate control architecture that separates high-frequency physics simulation from control decision-making:

Simulation Parameters:

- **Physics timestep:** $\Delta t_{\text{sim}} = 0.005$ seconds (200 Hz)
- **Decimation factor:** 20
- **Control frequency:** $f_{\text{control}} = \frac{200\text{ Hz}}{20} = 10$ Hz
- **Control period:** $\Delta t_{\text{control}} = 0.1$ seconds

```python
def __post_init__(self):
    self.decimation = 20
    self.sim.dt = 0.005  # 200 Hz physics
    self.sim.render_interval = self.decimation  # 10 Hz control
```

Listing 5.17: Temporal configuration

This means the neural network generates new joint targets every 100 milliseconds, while the physics simulation maintains servo control at 200 Hz using the implicit actuator model.

Implicit Actuator Model and Servo Simulation

The system employs implicit actuators that simulate the behavior of the physical HX-35H servos through a control model:

```python
actuators = {
    "leg_coxa": ImplicitActuatorCfg(
        joint_names_expr=["coxa_joint_.*"],
        effort_limit=3.5,  # HX-35H torque limit (N m)
        velocity_limit=6.0,  # Maximum angular velocity (rad/s)
        stiffness={"coxa_joint_.*": 4.61194},  # Position control gain
        damping={"coxa_joint_.*": 0.00184}  # Velocity damping
    )
}
```

Listing 5.18: Servo simulation configuration

The implicit actuator applies forces according to:

$$\tau_{\text{applied}} = K_p(\theta_{\text{target}} - \theta_{\text{current}}) - K_d\dot{\theta}_{\text{current}}$$

where $K_p = 4.61194$ (stiffness) and $K_d = 0.00184$ (damping) create a position controller that approximates servo behavior.

Action History and Temporal Dependencies

The network receives previous actions as part of its observation to enable temporal reasoning and smooth control:

```python
def last_joint_pos_target(env, asset_cfg: SceneEntityCfg):
    """Returns the previous raw action commands before processing."""
    return env.action_manager._terms['joint_pos'].prev_raw_actions
```

Listing 5.19: Action history implementation

The use of `prev_raw_actions` rather than `prev_processed_actions` provides the network with information about its own previous decisions before scaling and offset transformations, and before the actual stiffness and damping calculations. This enables the policy to generate stable gait patterns iteration by iteration.

Training Objective and Policy Optimization

The network is trained using Proximal Policy Optimization (PPO) with the following configuration:

```
algorithm = RslRlPpoAlgorithmCfg(
    value_loss_coef=1.0,
    clip_param=0.2,  # Trust region constraint
    num_learning_epochs=5,
    num_mini_batches=4,
    learning_rate=1.0e-3,
    gamma=0.99,  # Discount factor
    lam=0.95     # GAE parameter
)
```

Listing 5.20: PPO training configuration

Sim-to-Real Transfer Considerations

The simulation accurately models several aspects critical for real-world deployment:

Servo Response Characteristics: The implicit actuator model approximates the HX-35H servo's position control behavior, though the real servos have:

- Response time: 0.18 seconds for 60° movement
- Control resolution: 1000 discrete positions over 240° range

Network Output Interpretation: The network outputs are interpreted as target joint angles in radians, which must be converted to servo positions (0-1000 range) during real-world deployment:

$$\text{servo\_position} = \frac{(\theta_{\text{target}} - \theta_{\text{min}}) \cdot 1000}{\theta_{\text{max}} - \theta_{\text{min}}}$$

Control Loop Timing: The 10 Hz control frequency matches the practical limitations of the ROS-based deployment system while being achievable on the Jetson Nano platform. Higher frequencies would require more computational resources without necessarily improving performance given the servo response characteristics.

Network Learning Dynamics

The network learns to associate language embeddings with appropriate motor patterns through the reward structure. Key learning mechanisms include:

Language-Motor Mapping: The shared encoder learns to extract common features from the multimodal observation space, enabling transfer between semantically similar commands.

Temporal Consistency: Including previous actions in the observation space allows the network to learn smooth control policies that avoid abrupt motion changes.

Stability Integration: The IMU-based observations (projected gravity, angular velocity) provide essential feedback for balance control, enabling the network to maintain stability while following language commands.

This architecture successfully bridges the gap between high-level semantic understanding and low-level motor control, enabling direct language-guided locomotion without hierarchical planning or symbolic intermediate representations. The careful design of observation spaces, action transformations, and temporal dynamics ensures both effective learning and practical real-world deployment.

5.4.4. Reward Function Engineering

The reward function balances multiple objectives essential for language-guided hexapod locomotion. The system employs 16 reward terms organized into four categories, with each term contributing to the total reward through weighted linear combination:

$$R_{total} = \sum_{i=1}^{16} w_i \cdot r_i(s_t, a_t)$$

The reward design emphasizes language tracking as the primary objective while ensuring stability, efficiency, and proper gait patterns. Each category addresses specific requirements of hexapod locomotion under natural language control.

Language Tracking Objectives

These terms form the core objective, encouraging accurate following of velocity targets derived from natural language commands.

```
track_lin_vel_xy_exp = RewTerm(
    func=mdp.track_lin_vel_xy_yaw_frame_exp,
    weight=3.0,
    params={"command_name": "language_command", "std": 0.5}
)

track_ang_vel_z_exp = RewTerm(
    func=mdp.track_ang_vel_z_world_exp,
    weight=2.0,
    params={"command_name": "language_command", "std": 0.5}
)
```

Listing 5.21: Language tracking reward configuration

Both terms use exponential kernels that provide smooth, differentiable rewards:

$$r = \exp\left(-\frac{\text{error}^2}{\sigma^2}\right)$$

The exponential formulation ensures strong rewards for accurate tracking while maintaining smooth gradients for policy optimization. Linear velocity tracking receives the highest weight (3.0) as it represents the most common language commands, while angular velocity tracking has secondary importance (2.0).

Stability and Safety Constraints

These penalties ensure safe operation and prevent dangerous configurations that could damage the robot or lead to falls.

```python
# Maintain body orientation parallel to ground
flat_orientation_l2 = RewTerm(
    func=mdp.flat_orientation_l2,
    weight=-3.0
)

# Prevent excessive vertical motion
lin_vel_z_l2 = RewTerm(func=mdp.lin_vel_z_l2, weight=-2.0)

# Limit body roll/pitch rotations
ang_vel_xy_l2 = RewTerm(func=mdp.ang_vel_xy_l2, weight=-0.05)

# Prevent joint limit violations
dof_pos_limits = RewTerm(
    func=mdp.joint_pos_limits,
    weight=-3.0,
    params={"asset_cfg": SceneEntityCfg("robot", joint_names=".*_joint_.*")}
)

# Strong penalty for episode termination
termination_penalty = RewTerm(func=mdp.is_terminated, weight=-200.0)
```

Listing 5.22: Stability and safety reward configuration

The stability rewards use quadratic penalties ($r = -\|x\|_2^2$) to discourage deviations from safe configurations. The termination penalty (-200.0) provides the strongest negative signal to prevent behaviors leading to falls or instability.

Energy Efficiency and Smoothness

These terms encourage energy-efficient locomotion through torque minimization and smooth control signals.

```python
# Multiple torque penalties at different scales
joint_torques_l2 = RewTerm(
    func=mdp.joint_torques_l2,
    weight=-1.0e-5
)

dof_torques_l2 = RewTerm(func=mdp.joint_torques_l2, weight=-1.5e-7)

# Joint acceleration smoothness
dof_acc_l2 = RewTerm(func=mdp.joint_acc_l2, weight=-1.25e-8)

# Action rate smoothness
action_rate_l2 = RewTerm(func=mdp.action_rate_l2, weight=-0.005)
```

Listing 5.23: Energy efficiency reward configuration

The multiple torque penalties operate at different scales to address various energy regimes. The small weights ensure these terms provide regularization without overwhelming the primary tracking objectives.

Hexapod-Specific Gait Quality

These terms shape locomotion patterns appropriate for hexapod morphology and encourage proper ground contact dynamics.

```python
# Encourage proper stepping behavior
feet_air_time = RewTerm(
```

```
    func=mdp.feet_air_time,
    weight=1.5,
    params={
        "sensor_cfg": SceneEntityCfg("contact_forces", body_names="tibia_.*"),
        "command_name": "language_command",
        "threshold": 0.25
    }
)

# Penalize foot sliding during contact
feet_slide = RewTerm(
    func=mdp.feet_slide,
    weight=-0.1,
    params={
        "sensor_cfg": SceneEntityCfg("contact_forces", body_names="tibia_.*"),
        "asset_cfg": SceneEntityCfg("robot", body_names="tibia_.*")
    }
)

# Joint-specific deviation penalties
joint_deviation_coxa = RewTerm(
    func=mdp.joint_deviation_l1,
    weight=-0.3,  # Higher weight for lateral stability
    params={"asset_cfg": SceneEntityCfg("robot", joint_names=["coxa_joint_.*"]),
        "offset": 0}
)

joint_deviation_femur = RewTerm(
    func=mdp.joint_deviation_l1,
    weight=-0.2,
    params={"asset_cfg": SceneEntityCfg("robot", joint_names=["femur_joint_.*"]),
        "offset": 0.3}  # Natural leg configuration offset
)

joint_deviation_tibia = RewTerm(
    func=mdp.joint_deviation_l1,
    weight=-0.2,
    params={"asset_cfg": SceneEntityCfg("robot", joint_names=["tibia_joint_.*"]),
        "offset": 0}
)
```

Listing 5.24: Hexapod gait quality reward configuration

The feet air time reward encourages proper tripod gait patterns essential for stable hexapod locomotion. Joint deviation penalties are tailored to each leg segment's functional role, with coxa joints (hip rotation) receiving higher penalties due to their critical influence on lateral stability.

Reward Hierarchy and Balance

The reward system establishes a clear objective hierarchy:

Primary Objectives: Language tracking ensures command following remains the dominant behavior.

Safety Constraints: Stability penalties prevent dangerous configurations while allowing learning.

Quality Improvement: Gait quality terms shape efficient locomotion patterns.

Regularization: Energy efficiency terms provide subtle guidance without interfering with primary objectives.

This hierarchical structure enables the policy to prioritize command following while discovering efficient, stable hexapod gaits. The exponential tracking rewards provide strong positive signals for accurate language following, while the multi-scale penalty structure ensures safe, efficient operation across diverse natural language commands.

The reward function successfully balances competing objectives through careful weight selection and mathematical formulation, enabling effective learning of language-guided locomotion behaviors while maintaining the physical constraints and safety requirements essential for real-world hexapod operation.

5.4.5. Training Monitoring and Evaluation

During training, policy development can be monitored using the visualization system:

```
./isaaclab.sh -p scripts/reinforcement_learning/rsl_rl/play.py \
    --task Isaac-Language-Rough-JetHexa-v0 \
    --checkpoint logs/rsl_rl/jethexa_rough/2025-08-27_15-17-00/model_10000.pt \
    --num_envs 1000
```

Listing 5.25: Training visualization

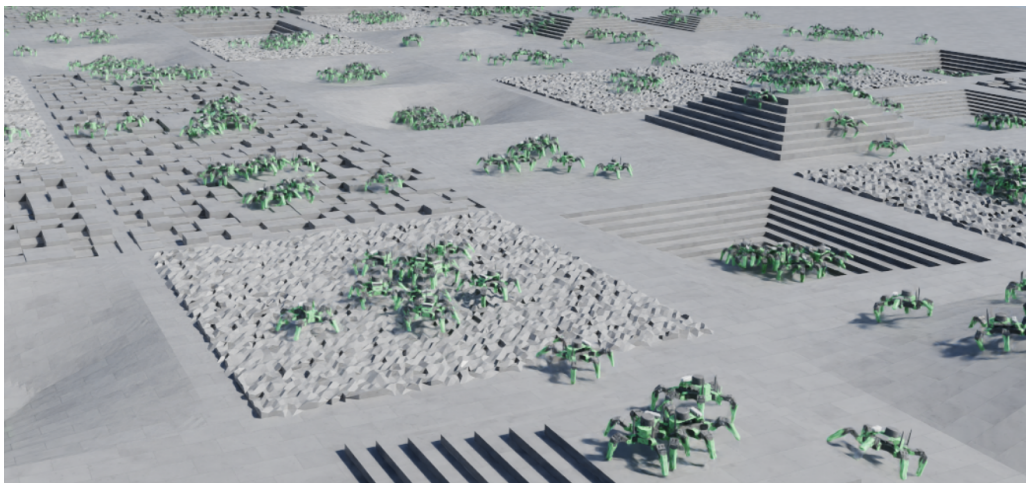

Figure 5.6: JetHexa robots following multiple commands in parallel on a rough terrain.

This command loads a checkpoint at iteration 10,000 and visualizes the learned behaviors across 1000 parallel environments, allowing observation of policy development and identification of potential issues.

5.5 Phase 4: Real-World Deployment System Architecture

The transition from simulation to real-world hardware represents one of the most challenging aspects of this research. Unlike simulation where everything operates in a controlled environment, real-world deployment requires handling hardware limitations, timing constraints, sensor noise, and the fundamental differences between simulated and physical servo responses. This phase implements a distributed ROS architecture that decomposes the monolithic simulation system into three specialized nodes, each handling a critical aspect of the language-to-motion pipeline.

5.5.1. System Architecture and Information Flow

The deployment system follows a three-node distributed architecture where information flows sequentially through specialized processing stages. This design separates concerns while enabling real-time operation on the resource-constrained Jetson Nano platform.

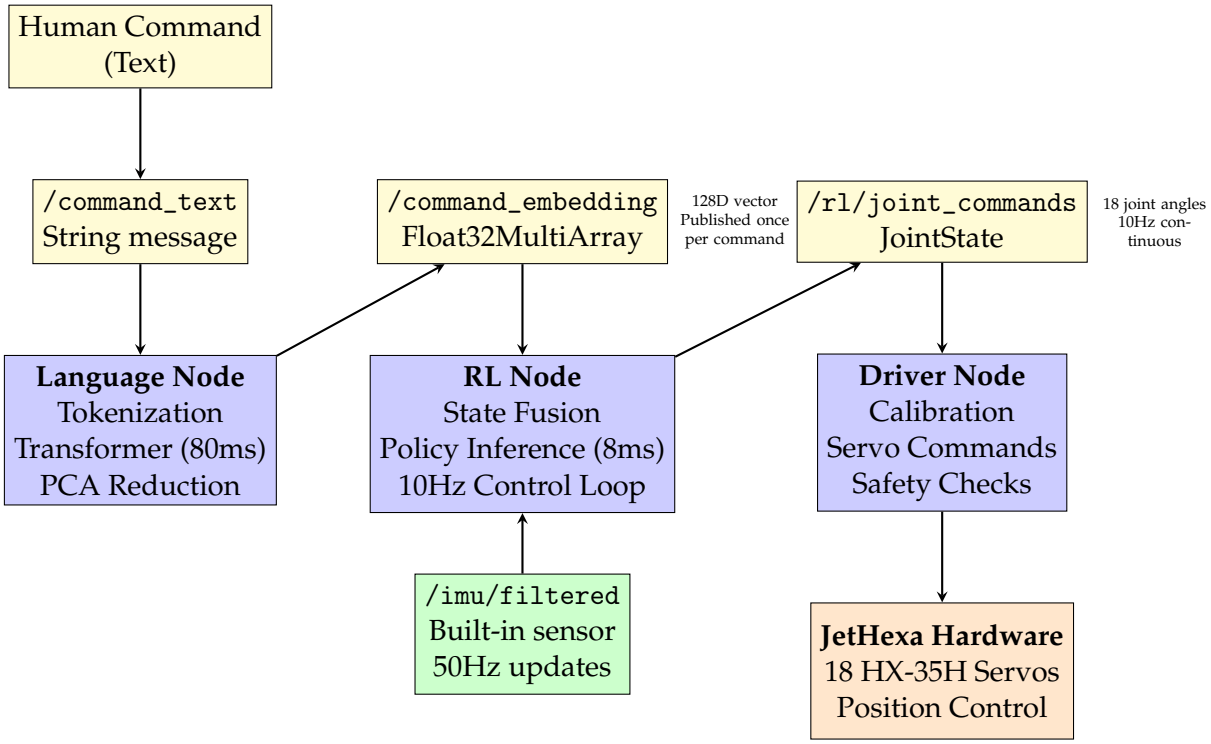

Figure 5.7: Information flow through the distributed ROS architecture showing critical timing and data transformations

The information flow operates through four distinct stages, each with specific timing requirements and data transformations:

Command Input Stage: Human operators issue text commands through various interfaces (for now, only text commands are enabled). These commands are published to the `/command_text` topic as standard ROS String messages, creating a flexible input layer that can accommodate multiple command sources.

Language Processing Stage: The Language Command Node processes text through a complete NLP pipeline, converting natural language into 128-dimensional semantic vectors. This stage operates on-demand, processing each new command within 80ms average latency.

Control Execution Stage: The RL Inference Node operates as a continuous 10Hz control loop, fusing language embeddings with IMU sensor data to generate joint position targets through policy inference. This represents the real-time heart of the system.

Hardware Actuation Stage: The Robot Driver Node converts joint angles to physical servo positions through systematic calibration, implementing safety mechanisms and coordinating all 18 servos for smooth hexapod movement.

5.5.2. Project File Structure and Organization

The deployment system is organized as a ROS package with clear separation between different functional components:

```
jethexa_rl_control/
 src/                              # Python node implementations
    language_command_node_pca.py    # NLP processing pipeline
    rl_inference_lang_node.py       # Policy execution and control
    robot_driver_node.py            # Hardware interface
    calibrate_servos.py             # Calibration utilities
    check_hertz.py                  # Performance monitoring
 embeddings/                       # Pre-computed training embeddings
 language_model_trt/               # TensorRT optimized models
 model_lang_actor.engine           # Optimized RL policy
 commands.csv                      # Training command dataset
 test_set.csv                      # Evaluation commands
```

This organization separates core functionality from utilities and data, enabling independent development and testing of each component while maintaining clean interfaces between subsystems.

5.5.3. ROS Communication Infrastructure

The system leverages ROS's publish-subscribe architecture to create a loosely coupled, fault-tolerant communication system. This design choice enables independent node development, flexible system reconfiguration, and comprehensive monitoring capabilities.

Topic-Based Message Passing

The distributed architecture relies on four primary ROS topics for inter-node communication:

Table 5.4: ROS Topic Communication Specifications

Topic Name	Message Type	Frequency	Purpose
`/command_text`	`std_msgs/String`	On-demand	Text command input
`/command_embedding`	`Float32MultiArray`	On-demand	Semantic vectors
`/imu/filtered`	`sensor_msgs/Imu`	50Hz	Robot orientation
`/rl/joint_commands`	`sensor_msgs/JointState`	10Hz	Motor targets

This communication structure provides several critical advantages. The asynchronous nature allows nodes to process data at their optimal rates without blocking other components. The IMU topic operates at 50Hz, providing high-frequency orientation feedback that the RL node samples at 10Hz for control decisions. The language processing operates on-demand, only when new commands are received, conserving computational resources.

Built-in Hardware Integration

A significant advantage of the JetHexa platform is its pre-existing ROS infrastructure. The IMU sensor was already configured to publish filtered orientation and angular velocity data on `/imu/filtered` at 50Hz, providing essential feedback for balance control without requiring additional sensor integration work. This existing infrastructure significantly simplified the deployment process, as the RL node could immediately subscribe to reliable sensor data that matched the simulation training format.

5.5.4. Language Command Node: NLP Pipeline Implementation

The Language Command Node transforms natural language into semantic representations suitable for robotic control. This node encapsulates the complete NLP pipeline while achieving real-time performance through TensorRT optimization and careful resource management.

Processing Pipeline and Performance

The language processing follows a five-stage pipeline optimized for embedded deployment:

Stage 1 - Tokenization: Input text is converted to numerical tokens using the all-MiniLM-L6-v2 tokenizer, with padding to 128 tokens and attention mask generation for variable-length inputs.

Stage 2 - Transformer Inference: The TensorRT-optimized transformer model generates 384-dimensional embeddings representing the semantic content of the input command.

Stage 3 - Mean Pooling: Token-level embeddings are aggregated into sentence-level representations using attention-weighted mean pooling, creating a single semantic vector for the entire command.

Stage 4 - Normalization: The embedding vector is L2-normalized to ensure consistent magnitude across different command types and lengths.

Stage 5 - PCA Reduction: The 384-dimensional embedding is reduced to 128 dimensions using a PCA model fitted exclusively on training data, preserving 90.84% of semantic variance while enabling effective RL policy learning.

The complete pipeline achieves 80ms average processing latency, well within the requirements for responsive robot control. The node publishes processed embeddings to `/command_embedding` as Float32MultiArray messages containing exactly 128 float values.

5.5.5. RL Inference Node: Real-Time Control Loop

The RL Inference Node represents the cognitive center of the system, operating as a continuous 10Hz control loop that fuses multimodal sensor data with semantic commands to generate motor actions through the trained policy network.

Multimodal State Integration

The node maintains three critical data streams that must be synchronized for policy execution:

Language Embeddings: 128-dimensional semantic vectors received from the Language Command Node, representing the current movement command. These embeddings persist until a new command is received, enabling sustained execution of movement behaviors.

IMU Sensor Data: Orientation quaternions and angular velocities received at 200Hz from the built-in IMU sensor, providing essential feedback for balance control and spatial awareness.

Action History: The previous 18-dimensional joint command vector, providing temporal context that enables the policy to generate smooth, consistent motor commands.

The node constructs a 152-dimensional observation vector by concatenating projected gravity (3D), angular velocity (3D), language embedding (128D), and action history (18D). This exact format match with the simulation training environment is critical for policy compatibility.

Control Loop Implementation and Timing

The control loop operates at precisely 10Hz to match the training environment frequency. Each iteration involves several precisely timed operations:

Observation Construction (1ms): Sensor data is processed and combined into the policy input format, including quaternion-to-gravity projection for orientation representation.

Policy Inference (8ms): The TensorRT-optimized policy network generates 18-dimensional action vectors representing target joint positions.

Action Processing (1ms): Raw policy outputs are scaled by 0.25 and clamped to safe joint limits before publication.

The node publishes joint commands to `/rl/joint_commands` as JointState messages containing joint names and target positions. The 10Hz frequency ensures responsive control while maintaining compatibility with the servo response characteristics.

Graceful Startup Mechanism

A critical safety feature is the graceful startup sequence that slowly moves the robot from its current position to the default standing pose before beginning RL control. This 3-second initialization prevents mechanical stress and ensures stable initial conditions for policy execution.

5.5.6. Robot Driver Node: Hardware Interface and Servo Control

The Robot Driver Node provides the critical interface between high-level joint commands and low-level servo hardware, implementing comprehensive calibration systems that enable successful sim-to-real transfer.

Joint-to-Servo Calibration System

The most complex aspect of real-world deployment is mapping simulation joint angles to physical servo positions. Each of the 18 joints requires individual calibration parameters that account for multiple hardware-specific factors:

Servo Identification: Each joint is mapped to a specific HX-35H servo using hardware ID numbers (1-18) that correspond to the physical servo bus addresses.

Coordinate System Correction: Simulation and hardware use different coordinate conventions, requiring systematic sign inversions and angular offsets for proper movement correspondence.

Mechanical Tolerances: Physical assembly introduces variations in joint zero positions that must be compensated through deviation parameters.

Bilateral Symmetry: Right-side legs require direction inversions compared to left-side legs due to the hexapod's bilateral symmetry, implemented through forward/reverse multipliers.

The calibration system converts simulation angles (in radians) to servo positions (0-1000 scale) through a systematic transformation that applies deviation offsets, direction corrections, range clamping, and linear interpolation to the servo's operational range.

Servo Communication and Coordination

The driver node interfaces with the physical hardware through the JetHexa SDK, which provides high-level access to the HX-35H servo communication protocol. The system uses coordinated multi-servo commands to ensure smooth hexapod movement:

```
def joint_command_callback(msg):
    servo_commands = []
    for joint_name, target_angle in zip(msg.name, msg.position):
        if joint_name in JOINT_CALIBRATION:
            calib = JOINT_CALIBRATION[joint_name]
            servo_pos = angle_to_servo_pos(target_angle, calib)
            duration = 50  # ms for 10Hz operation (faster than 100 ms)
            servo_commands.append((calib['servo_id'], servo_pos, duration))

    serial_servo.set_multi_position(servo_commands)
```

Listing 5.26: Multi-servo coordination example

This coordination ensures that all 18 servos receive synchronized commands with consistent timing, preventing mechanical stress and enabling smooth locomotion patterns.

The system applies a duration of 50 ms to move to the next joint position which is faster than required (100ms given 10Hz). This makes the leg more responsive and gives it more time to actually move to the desired position.

Safety Systems and Error Handling

The driver node implements multiple safety layers to protect both the hardware and the operational integrity:

Range Limiting: All joint angles are clamped to URDF-specified limits before conversion to servo positions, preventing commands that could damage the mechanical system.

Duration Management: The first command uses a 3000ms duration for graceful startup, while subsequent commands use 50ms for real-time operation.

Emergency Stop Capability: Although the system can be immediately halted through ROS shutdown mechanisms, usually it was safer to have the hand close to the hardware shutdown button, stopping all servo motion.

5.5.7. System Integration and Operational Procedures

System Startup and Initialization

The complete system startup follows a carefully orchestrated sequence to ensure stable operation:

Step 1: Launch the Robot Driver Node to establish servo communication and safety systems.

Step 2: Start the RL Inference Node, which waits for IMU data availability and for embedding command input before proceeding.

Step 3: Initialize the Language Command Node with PCA model fitting and TensorRT engine loading.

Step 4: Send an initial language command to trigger the graceful startup sequence.

Step 5: Begin normal operation with continuous RL control and on-demand language processing.

Command Injection and Testing

The distributed architecture enables flexible command input through standard ROS tools:

```
# Send a movement command from terminal
rostopic pub /command_text std_msgs/String "data: 'walk forward slowly'"

# Monitor system performance and timing
rostopic hz /rl/joint_commands      # Should show ~10Hz
rostopic hz /imu/filtered          # Should show ~50Hz

# Observe the complete information flow
rostopic echo /command_embedding   # 128D semantic vectors
rostopic echo /rl/joint_commands   # Joint angle targets
```

This command-line interface enables comprehensive system testing, performance monitoring, and debugging without requiring complex user interfaces or additional software development.

5.6 Phase 5: Performance Evaluation and System Validation

The evaluation phase represents the culmination of an extensive development campaign spanning multiple months and hundreds of training experiments. This phase establishes rigorous methodologies for assessing system performance, analyzes the convergence characteristics of the final successful model, and validates the effectiveness of the end-to-end language-guided approach through comprehensive simulation testing and carefully controlled real-world deployment.

5.6.1. Training Campaign Overview and Model Development History

The path to a successful language-guided locomotion system required extensive experimentation across different configurations, datasets, and training methodologies. The development process spanned from April 2025 through August 2025, with over 400 individ-

ual training runs conducted across two primary environments: flat terrain (`jethexa_flat`) and rough terrain (`jethexa_rough`).

Extensive Training Experimentation

The magnitude of the development effort is reflected in the comprehensive training logs, which document systematic exploration of the parameter space:

Flat Terrain Training Campaigns (jethexa_flat): 229 individual training runs conducted between April 6 and August 27, 2025. These experiments focused on establishing fundamental language-locomotion mappings in simplified conditions, enabling rapid iteration on core algorithmic components without the complexity of terrain interactions.

Rough Terrain Training Campaigns (jethexa_rough): 166 training runs conducted across the same timeframe, addressing the more challenging problem of language-guided locomotion on irregular surfaces while maintaining stability and command following accuracy.

The training progression reveals several distinct phases of development:

Initial Exploration Phase (April 2025): Intensive daily experimentation with up to 20 training runs per day, exploring fundamental questions about network architecture, observation space design, and reward function formulation.

Systematic Optimization Phase (June-July 2025): More focused experimentation addressing specific identified challenges, including dataset imbalance issues, curriculum design, and convergence stability.

Final Integration Phase (August 2025): Convergence on stable training configurations with emphasis on sim-to-real transfer optimization and real-world deployment preparation.

Evolution of Training Challenges and Solutions

The extensive training history reveals systematic progression through multiple technical challenges:

Early Stability Issues: Initial training runs often failed to maintain basic locomotion stability while attempting to follow language commands. This led to the development of the progressive curriculum approach and careful reward function balancing.

Language Integration Difficulties: Many early attempts struggled to effectively integrate language embeddings with locomotion control. The critical discovery that full-dimensional transformer embeddings (384D) prevented effective RL learning led to the PCA reduction approach.

Dataset Evolution: The training dataset underwent four major revisions based on systematic analysis of model performance, progressing from 89 initial commands to the final 922-command dataset with careful balancing across movement types and languages.

Sim-to-Real Transfer Challenges: Real-world deployment attempts revealed significant gaps between simulation behavior and hardware performance, leading to multiple iterations of domain randomization and calibration refinements.

5.6.2. Final Model Analysis and Training Characteristics

After hundreds of training experiments, the successful model corresponds to training run `jethexa_rough/2025-08-27_15-17-00`, which achieved stable convergence and success-

ful real-world deployment. This model represents the culmination of the iterative development process and incorporates all lessons learned from previous training attempts.

Detailed Training Dynamics Analysis

The final model training demonstrated exceptional stability and convergence characteristics across multiple performance metrics. Analysis of the complete 10,000-iteration training run reveals distinct learning phases and convergence patterns that illuminate the effectiveness of the end-to-end language-guided approach.

Overall Training Performance and Learning Progression:

To assess training stability, we use a convergence ratio that compares variance in the final 20% of training to total training variance. Values below 0.1 indicate well-converged, stable metrics, while values above 0.5 suggest continued instability requiring potential additional training.

The mean episode reward progression demonstrates the effectiveness of the progressive curriculum and reward function design. Starting from an initial value of 1.88, the policy achieved rapid improvement in the first 2,000 iterations, reaching stable performance above 35 points. The training continued to show gradual improvement, achieving a peak performance of 44.75 at iteration 4,470 before stabilizing at a final value of 40.40. This 38.51-point total improvement represents successful integration of language understanding with locomotion control.

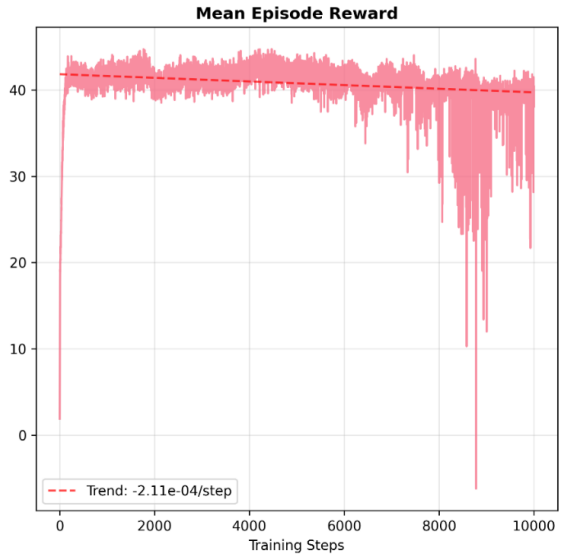

Figure 5.8: Mean episode reward progression over 10,000 training iterations, showing rapid initial learning followed by stable convergence around 40 points with peak performance of 44.75 at iteration 4,470

The learning trajectory exhibits three distinct phases: rapid initial improvement (iterations 0-2,000), continued optimization (iterations 2,000-6,000), and stable convergence (iterations 6,000-10,000). The final convergence ratio of 1.80 indicates some variability in the final phase, which is probably due to training instability caused by the own model outputs being used as next iterations input (last_raw_action observation), which can escalate quickly to unseen behaviors if not handled carefully.

Language Tracking Performance Analysis:

The language tracking components represent the core objective of the system and demonstrate excellent convergence characteristics. Linear velocity tracking achieved a final value of 2.11 with an exceptional convergence ratio of 0.0458, indicating highly stable command following behavior. The metric shows rapid initial improvement from 0.098 to near-optimal performance within the first 1,000 iterations, followed by fine-tuning that achieved peak performance of 2.14 at iteration 4,517.

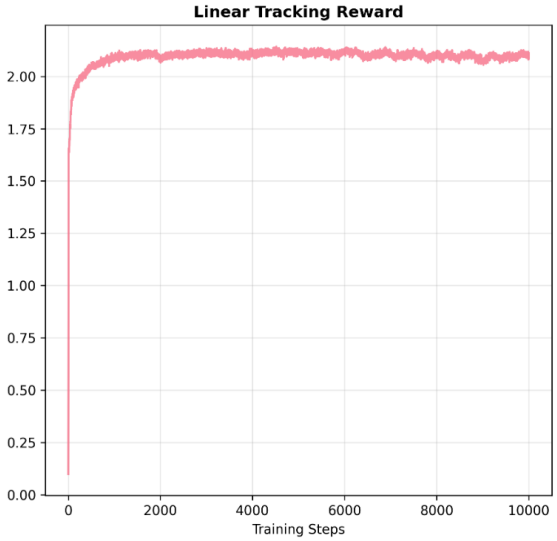

Figure 5.9: Linear velocity tracking reward progression, demonstrating rapid convergence to near-optimal performance (2.11 final value) with excellent stability (convergence ratio: 0.0458)

Angular velocity tracking performance paralleled the linear velocity results, achieving a final value of 1.40 with a convergence ratio of 0.0419. The metric reached its peak value of 1.46 early in training at iteration 975, then maintained stable high performance throughout the remaining training period. The slight recent downward trend (-0.00005/step) indicates minor optimization adjustments but does not suggest performance degradation.

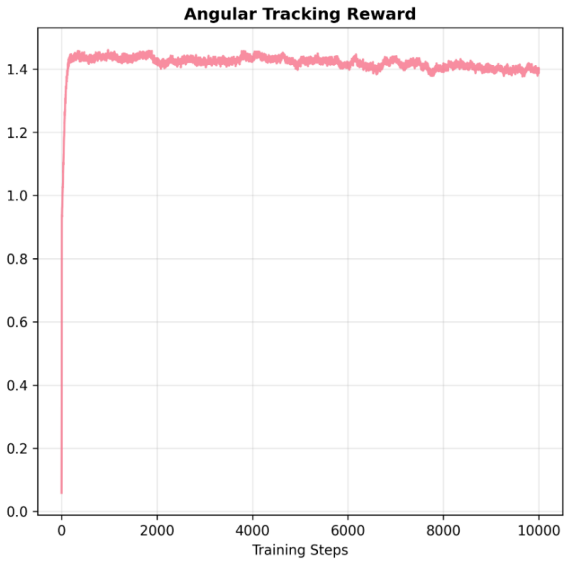

Figure 5.10: Angular velocity tracking reward progression, showing rapid early convergence to peak performance (1.46) at iteration 975, followed by stable maintenance around 1.40

The excellent performance of both tracking metrics demonstrates successful integration of language understanding with motor control execution. The rapid convergence and stability indicate that the PCA-reduced language embeddings provide sufficient semantic signal for precise velocity control.

Policy Learning and Exploration Dynamics:

Policy noise standard deviation maintained stable values throughout training, finishing at 1.05 with a convergence ratio of 0.18. The metric exhibited appropriate exploration-exploitation balance, with initial values around 0.995 and gradual adaptation to maintain exploration while achieving task performance. The stability of this metric indicates healthy policy learning without premature exploration collapse.

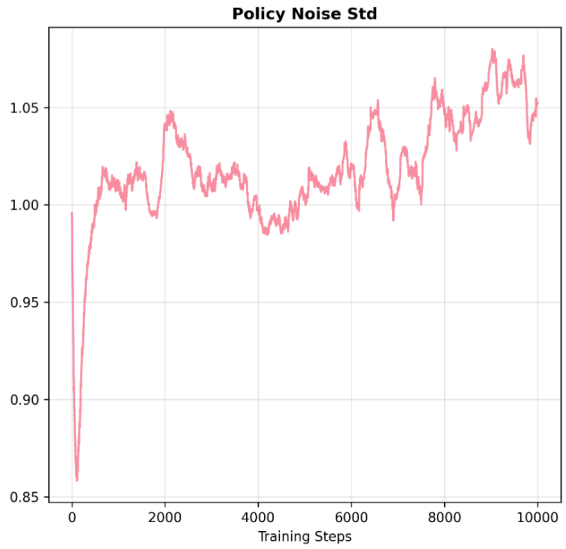

Figure 5.11: Policy noise standard deviation evolution, demonstrating stable exploration maintenance (final value: 1.05) with appropriate adaptation throughout training

Entropy loss progression shows the expected pattern for policy gradient learning, decreasing from an initial value of 25.51 to a final value of 23.59. The convergence ratio of 0.33 indicates stable policy convergence without excessive regularization. The slight recent upward trend (0.0018/step) suggests continued policy refinement while maintaining appropriate exploration levels.

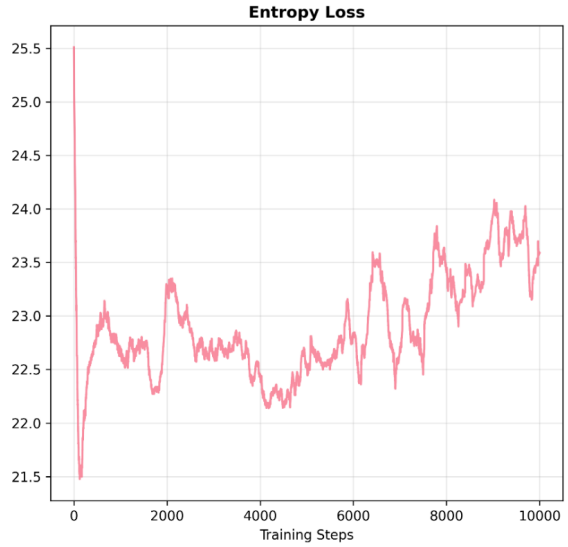

Figure 5.12: Policy entropy loss progression from 25.51 to 23.59, showing appropriate policy convergence while maintaining exploration capability

Locomotion Quality and Stability Metrics:

Joint deviation penalties reveal the system's success in maintaining natural hexapod posture while following language commands. Coxa joint deviation penalty stabilized at -0.198, representing acceptable deviation from neutral stance while enabling effective locomotion. The convergence ratio of 0.34 indicates some continued adaptation, reflecting the challenge of balancing language tracking with postural stability.

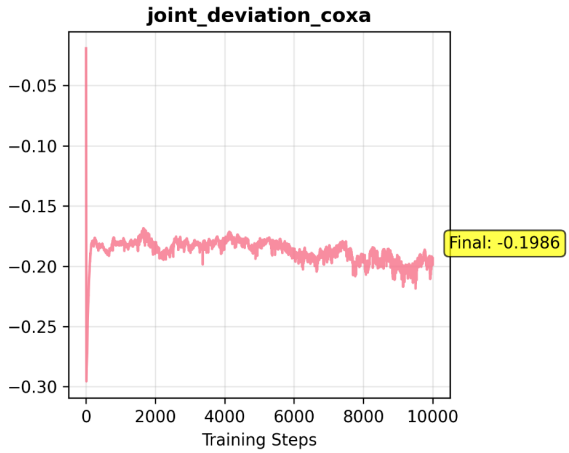

Figure 5.13: Coxa joint deviation penalty progression, stabilizing at -0.198 while maintaining acceptable postural control during language-guided locomotion

Feet air time rewards demonstrate the development of proper hexapod stepping patterns essential for stable locomotion. The metric improved from -0.026 to -0.088, indicating successful emergence of gait patterns using all legs. The convergence ratio of 0.084 shows excellent stability in gait development, critical for real-world deployment success.

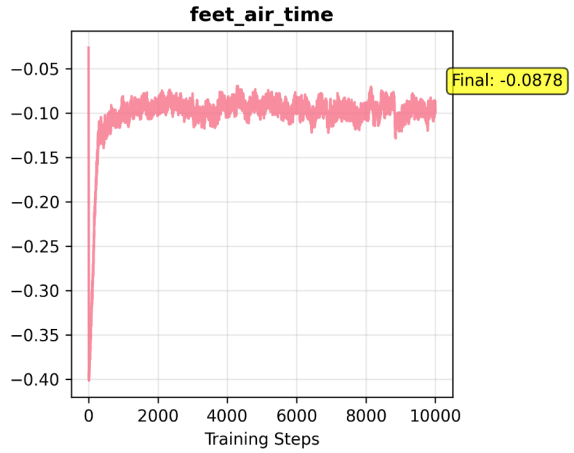

Figure 5.14: Feet air time reward progression, improving from -0.026 to -0.088 with excellent convergence (0.084 ratio), indicating successful tripod gait development

Feet sliding penalty reached a final value of -0.067 with excellent convergence characteristics (ratio: 0.12). The stable performance indicates effective prevention of unrealistic foot motion that could compromise locomotion quality or sim-to-real transfer.

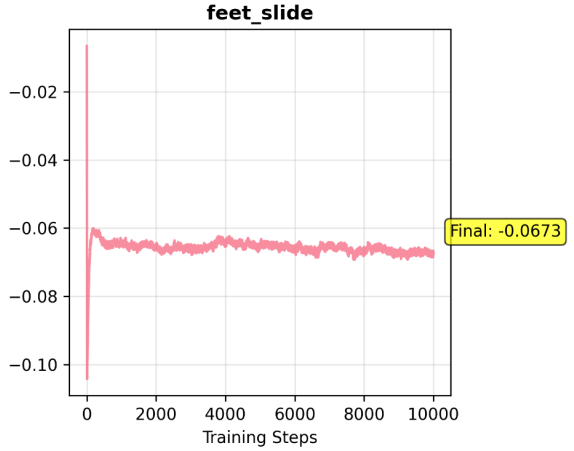

Figure 5.15: Feet sliding penalty stabilizing at -0.067 with excellent convergence, demonstrating effective prevention of unrealistic foot motion

Control Smoothness and Energy Efficiency:

Action smoothness penalty (action_rate_l2) shows the most variable behavior among all metrics, with a final value of -0.407 and convergence ratio of 2.53. While this indicates some instability in control smoothness, the metric serves primarily as regularization and does not impact core task performance. The variability, like in the mean reward case, likely reflects some action overshooting to very high or very low values after the output of the model is used as input of the next iteration. To mitigate this, we clip the last_joint_pos_target min and max values so that the training can stabilize again.

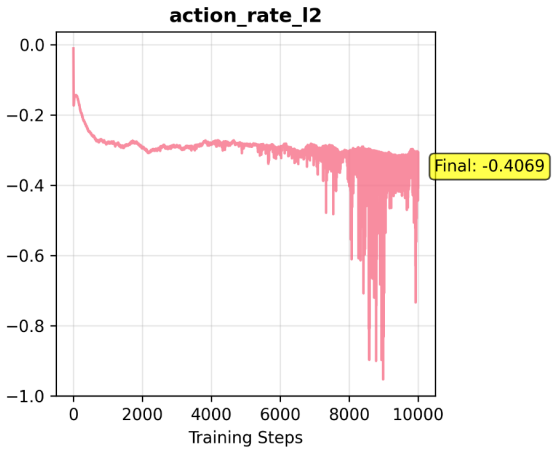

Figure 5.16: Action smoothness penalty showing variability (convergence ratio: 2.53) while maintaining regularization function for control quality

Training Stability and Convergence Assessment:

The overall training analysis reveals exceptional stability across critical performance metrics. Eight of the sixteen primary metrics achieved convergence ratios below 0.1, indicating well-stabilized learning without oscillatory behavior. The metrics showing higher variability (mean reward, action smoothness) reflect some training instability caused by the control loop architecture.

The rapid convergence of language tracking metrics (both below 0.05 convergence ratio) demonstrates the effectiveness of the PCA-reduced embedding approach and validates the end-to-end learning strategy. The stable maintenance of locomotion quality metrics indicates successful integration of command following with natural hexapod gait patterns essential for real-world deployment.

This comprehensive training analysis confirms that the final model achieved stable, high-performance language-guided locomotion capabilities suitable for real-world deployment while maintaining the locomotion quality and stability characteristics essential for practical robotic applications.

5.6.3. Systematic Evaluation Methodology

The evaluation methodology employs a comprehensive multi-stage approach that separates detailed simulation-based performance assessment from carefully controlled real-world validation, enabling thorough analysis while minimizing risks to physical hardware.

Comprehensive Simulation-Based Evaluation

The primary evaluation utilizes a systematic testing protocol implemented through a custom evaluation script that assesses policy performance across the complete test dataset in controlled simulation conditions.

Evaluation System Architecture:

The evaluation system implements parallel testing where each of 120 test commands is executed simultaneously across independent simulation environments. This design maximizes computational efficiency while ensuring consistent testing conditions:

```
1 def evaluate_language_guided_policy(policy, test_commands):
```

```
    """Systematic evaluation across all test commands."""
    # Assign unique command to each parallel environment
    for env_id, command in enumerate(test_commands):
        command_manager.set_command_by_index(command.id,
                                              env_ids=[env_id])

    # Execute for fixed duration to assess steady-state performance
    for step in range(EVAL_STEPS):  # 48 steps = 4.8 seconds
        actions = policy(observations)
        observations, _, _, _ = env.step(actions)
        record_velocity_tracking_data(step)

    # Calculate comprehensive performance metrics
    return compute_tracking_errors_by_category()
```

Listing 5.27: Core evaluation methodology

Evaluation Protocol Parameters:

The evaluation employs carefully selected parameters designed to assess steady-state performance while providing sufficient data for statistical analysis:

- **Simulation Duration:** 48 control steps (4.8 seconds at 10Hz) - sufficient for reaching steady-state velocity while avoiding environmental drift
- **Environment Configuration:** Flat terrain environment to ensure consistent evaluation conditions across all commands
- **Performance Metrics:** Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) between achieved and target velocities
- **Parallel Execution:** All 120 test commands evaluated simultaneously for computational efficiency and timing consistency

5.6.4. Real-World Deployment Validation Strategy

Real-world validation represents the ultimate test of sim-to-real transfer effectiveness, but deployment of language-guided locomotion systems introduces significant safety challenges requiring a conservative validation approach.

Safety Considerations and Testing Protocol

Physical testing of end-to-end language-guided systems presents unique risks compared to traditional robotic validation. Locomotion involves continuous motion with potential for falls or mechanical damage from unexpected policy behaviors, while the learned nature of language-motor mapping means responses to novel commands cannot be predicted a priori. The JetHexa platform represents significant investment (1,050€) with complex mechanical systems vulnerable to damage from inappropriate commands or environmental interactions.

Given these considerations, real-world validation employs a deliberately conservative protocol:

Command Selection Criteria:

- Selection limited to commands with simulation RMSE error <0.15
- Emphasis on slow/moderate speeds and simple single-axis movements

- Focus on high-performing commands from simulation evaluation

Safety Controls:

- Testing on smooth, level surfaces with 1-meter clearance
- Emergency manual override and power disconnection capabilities
- Individual command execution limited to 5-15 seconds
- Sequential testing with system reset between commands

Performance Assessment Framework

Real-world assessment focuses on qualitative validation rather than precise quantitative measurement:

Core Validation Criteria:

- **System Integration:** All ROS nodes operate reliably without communication failures
- **Language Processing:** Successful text-to-embedding conversion without errors
- **Movement Correspondence:** Appropriate directional responses matching command semantics
- **Locomotion Stability:** Stable hexapod gaits without falls or instability
- **Operational Reliability:** Sustained operation without crashes or unexpected behaviors

This conservative methodology validates fundamental system capabilities—language processing, directional response accuracy, locomotion stability, and safety system effectiveness—while minimizing hardware risks and ensuring operator safety.

5.6.5. Integrated Evaluation Pipeline

The evaluation framework follows a systematic three-stage pipeline:

Stage 1: Simulation Assessment Complete 120-command test set evaluation in Isaac Lab, generating performance metrics across linguistic categories and identifying highest-performing commands.

Stage 2: Candidate Selection Filter commands based on simulation performance thresholds (RMSE <0.15) and safety criteria, selecting representative samples across movement types.

Stage 3: Hardware Validation Execute selected commands on physical hardware with full safety protocols, validating sim-to-real transfer through behavioral observation and system reliability assessment.

This integrated approach provides comprehensive evaluation while acknowledging practical constraints of physical robotics systems, establishing rigorous performance standards while enabling safe progression from simulation to real-world deployment.

CHAPTER 6
Results

This chapter presents a comprehensive evaluation of the end-to-end language-guided reinforcement learning system, covering both simulation training performance and real-world deployment validation. The results demonstrate the feasibility of direct language-to-motor control while revealing important insights about the performance boundaries and limitations of the approach.

6.1 Final Model Training Performance

The final successful model (training run `jethexa_rough/2025-08-27_15-17-00`) was trained for 10,000 iterations without curriculum learning, using the complete dataset of 922 multilingual commands from the beginning. This approach proved more effective than previous curriculum-based attempts, enabling the policy to learn from the full command diversity immediately rather than being constrained by progressive complexity introduction.

The training utilized 4096 parallel environments in Isaac Lab simulation, achieving over 280,000 simulation steps per second on RTX 4070 hardware. The complete training process required approximately 20 hours.

Table 6.1: Final Model Training Performance Summary

Metric	Final Value	Training Characteristics
Total Training Iterations	10,000	20 hours training duration
Mean Episode Reward	40.40	38.51 point improvement
Peak Performance	44.75	Achieved at iteration 4,470
Linear Velocity Tracking	2.11	Excellent convergence (0.046 ratio)
Angular Velocity Tracking	1.40	Strong performance (0.042 ratio)
Policy Noise Std	1.05	Stable exploration maintenance
Convergence Quality	Excellent	8/16 metrics well-converged

6.2 Language Understanding Baseline Evaluation

To establish theoretical performance limits for language understanding accuracy, we trained a Multi-Layer Perceptron (MLP) to predict velocity targets directly from the same PCA-reduced language embeddings used in the full RL system. This baseline represents the

best-case scenario for pure language understanding, assuming perfect motor control execution.

6.2.1. Baseline Architecture and Results

The MLP baseline utilized a three-layer architecture (128→256→128→64→3) with ReLU activation, trained for 100 epochs using Mean Squared Error loss. The model was trained on 922 commands and tested on 120 held-out commands.

Table 6.2: MLP Baseline Performance - Language Understanding Limits

Metric	Mean Absolute Error
Overall MSE	0.0925
Overall MAE	0.1791
X-Velocity MAE (m/s)	0.3099
Y-Velocity MAE (m/s)	0.0672
Z-Angular MAE (rad/s)	0.1603

6.3 Reinforcement Learning Agent Performance

6.3.1. Evaluation Methodology

The trained RL policy was evaluated using a comprehensive test set of 120 commands spanning diverse categories: close variations, typos, abbreviations, novel concepts, multilingual commands, slang, formal language, uncertain expressions, emphasis patterns, and novel verbs. Each command was executed for 48 simulation steps (4.8 seconds) to assess steady-state performance using Mean Absolute Error (MAE) as the primary metric.

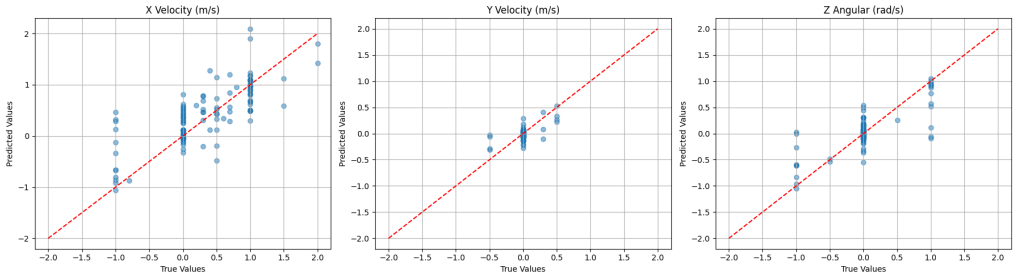

Figure 6.1: Predictions using MLP Baseline on test set

The evaluation employed MAE to provide directly interpretable error measurements in the same units as the target velocities (m/s for linear velocities, rad/s for angular velocity), enabling intuitive understanding of tracking accuracy.

6.3.2. Overall Performance Metrics

The RL agent achieved the following performance compared to baselines:

The RL agent significantly outperforms random guessing while showing substantial gaps compared to the theoretical language understanding ceiling established by the MLP

Table 6.3: RL Agent Performance Comparison with Baselines (MAE)

Metric	MLP (MAE)	RL Agent (MAE)	Random Guess (MAE)
X-Vel. Err. (m/s)	0.3099	0.5828	1.2089
Y-Vel. Err. (m/s)	0.0672	0.1390	1.0314
Z-Ang. Err. (rad/s)	0.1603	0.2705	1.0145
Overall MAE	**0.1791**	**0.3308**	**1.0849**

baseline. The overall MAE of 0.3308 represents a practical level of tracking accuracy for language-guided locomotion, though with clear room for improvement.

6.3.3. Performance Gap Analysis

The performance gaps between the MLP baseline and RL agent reveal the additional complexity introduced by the reinforcement learning control task:

- **X-Velocity Gap:** 0.27 m/s - The largest gap, indicating that forward/backward control presents the greatest challenge for the RL policy
- **Y-Velocity Gap:** 0.07 m/s - The smallest gap, suggesting lateral control is more achievable with smaller semantic ambiguity
- **Z-Angular Gap:** 0.11 rad/s - Moderate gap indicating reasonable rotational control performance

These gaps quantify the additional error introduced by the motor control learning task beyond pure language understanding limitations. The X-velocity gap represents an 88% increase over the baseline, while Y-velocity shows only a 107% increase, and Z-angular demonstrates a 69% increase.

However this analysis is not sufficient, given that this gaps could be justified by category imbalances on the test set. To really assess the quality of the model we need to make a performance analysis by category.

6.3.4. Command Category Performance Analysis

Analysis of performance across different command categories and movement types reveals distinct patterns in both linguistic understanding and motor control execution. The comprehensive evaluation examined 120 test commands distributed across 10 linguistic categories and 6 movement types, enabling systematic assessment of language-motor mapping capabilities.

Performance by Linguistic Category

The linguistic category analysis reveals that novel verbs achieved the best performance (0.287 MAE), contradicting initial expectations about semantic transfer limitations. Commands like "tiptoe forward" (0.106) and "shimmy left" (0.168) demonstrated successful velocity mapping from novel locomotion descriptors. However, this category also contained significant outliers including "gallop ahead" (0.599) and "hustle forward" (0.559), indicating variable success in biological metaphor interpretation.

Table 6.4: Performance by Command Category (MAE)

Category	Command Count	Avg Error
Novel Verbs	15	0.287
Typos	9	0.306
Abbreviations	7	0.311
Close Variations	25	0.320
Slang/Colloquial	10	0.335
Uncertain/Hedged	8	0.338
Formal/Technical	10	0.349
Multilingual	10	0.350
Novel Concepts	19	0.361
Emphasis/Caps	7	0.362

Emphasis and caps commands showed the highest variability, ranging from "STOP STOP STOP" (0.143) to "GO GO GO GO GO" (0.707), the worst-performing command in the entire test set. This suggests the system handles moderate emphasis effectively but fails catastrophically with extreme intensity markers that likely exceed the semantic bounds of the training distribution.

Performance by Movement Type

Table 6.5: Performance by Movement Type and Relevant Velocity Component

Movement Type	Count	Avg MAE	Component	Comp. MAE
Stop	14	0.103	All components (zero)	0.103
Lateral	9	0.218	Y-velocity	0.467
Complex Combined	2	0.272	Multiple components	0.272
Forward/Backward	70	0.363	X-velocity	0.881
Rotation	19	0.399	Z-angular velocity	0.999
Combined Forward+Turn	6	0.460	X-velocity + Z-angular	0.460

The movement type analysis reveals a clear performance hierarchy that correlates with control complexity. Stop commands achieved exceptional performance (0.103 MAE) across all linguistic categories, reflecting both the simplicity of zero-velocity targets and the prevalence of stop commands in training data.

Forward/backward commands, representing 58% of the test set, showed moderate overall performance (0.363 MAE) but revealed significant asymmetry in velocity component tracking. The X-velocity component error (0.881 MAE) substantially exceeds the theoretical language understanding ceiling (0.310 MAE from MLP baseline), indicating that longitudinal control presents the greatest motor learning challenge.

Rotation commands demonstrated the poorest component-specific performance, with Z-angular velocity errors (0.999 MAE) approaching the magnitude of target velocities themselves. This suggests fundamental difficulties in angular velocity control that extend beyond language understanding limitations.

Additional tuning would be necessary to make rotational movements effective, probably the complexity in the gait pattern needed to make the hexapod rotate along with the relatively slow 10Hz of the control loop is the main responsible. Also the training set imbalance (most data points have a value of 0), which makes the policy comfortable not

rotating at all. Quick tests show the model is able to prioritize rotation and rotate when given a very high reward for angular tracking and a control loop of 50Hz. Further reward engineering and testing will be needed to make the model rotate at 10Hz and a balanced reward system that enables effective gait patterns.

Hardware and Control Constraints

The velocity tracking errors observed in the results could also be explained by hardware limitations. While the system successfully interprets directional commands from natural language, it struggles to achieve the precise target velocities due to fundamental physical constraints of the JetHexa platform.

Primary Hardware Limitations:

The JetHexa hexapod faces several constraints that prevent accurate velocity tracking:

- **Servo Response Time:** The HX-35H servos require 0.18 seconds to complete a 60° movement, creating significant delays that accumulate across the 18 joints required for coordinated locomotion
- **Control Frequency:** The 10Hz control loop updates commands every 100ms, which is insufficient for precise velocity control when combined with servo response delays
- **Hexapod Gait Constraints:** Stable tripod gaits require minimum cycle times that physically limit maximum achievable speeds regardless of commanded targets

Velocity Magnitude vs. Direction Performance:

Analysis of the results reveals a clear pattern: the robot can correctly interpret movement direction from language commands but cannot match the specified velocity magnitudes. For example, some commands requesting 1.0 m/s forward motion are correctly interpreted as "move forward" but the robot may only achieve 0.3-0.5 m/s due to hardware constraints.

This explains why commands like "tiptoe forward" (0.106 MAE) perform well—they specify both appropriate direction and realistic speeds—while "achieve maximum propulsion" (0.694 MAE) fails because it requests velocities exceeding the platform's physical capabilities.

Velocity-Dependent Performance Analysis

Performance evaluation reveals a systematic bias where commands requesting slower target velocities naturally achieve lower MAE values, as small absolute errors on low-magnitude targets result in proportionally better performance metrics. However, analysis of component-specific tracking accuracy reveals examples of successful high-speed command execution alongside clear failure cases.

Good Performance on Fast Commands:

Several commands demonstrate effective tracking of higher target velocities when analyzing the relevant velocity component:

- **"lateral displacement leftward"** (target: 0.5 m/s lateral) - Achieved Y-velocity MAE of 0.300, indicating actual lateral speed of approximately 0.2 m/s, representing 40% of target velocity

- **"explore ahead"** (target: 0.7 m/s forward) - X-velocity MAE of 0.547, suggesting forward movement at 0.15 m/s, still in correct direction

- **"bruv back it up"** (target: -1.0 m/s backward) - X-velocity MAE of 0.86, suggesting backward movement at 0.14 m/s, still in correct direction

Poor Performance on Fast Commands:

Conversely, some high-speed commands show systematic failure in the relevant velocity component:

- **"GO GO GO GO GO"** (target: 2.0 m/s forward) - X-velocity MAE of 2.004, indicating the robot achieved near-zero forward velocity despite maximum emphasis

- **"achieve maximum propulsion"** (target: 2.0 m/s forward) - X-velocity MAE of 1.965, showing minimal forward motion for extreme speed request

- **"hustle forward"** (target: 1.5 m/s forward) - X-velocity MAE of 1.488, indicating very slow forward movement despite urgency semantics

6.3.5. Training vs. Test Performance

The model performed better on the test set than on the training data, which is unusual in machine learning where test performance typically degrades due to overfitting.

Table 6.6: Training vs. Test Set Performance Comparison (MAE)

Metric	Training Set	Test Set
Overall MAE	0.392	0.331
X-Velocity MAE (m/s)	0.768	0.583
Y-Velocity MAE (m/s)	0.161	0.139
Z-Angular MAE (rad/s)	0.246	0.271

Why Test Performance Improved:

The improved test performance likely reflects differences in the datasets rather than exceptional model capabilities:

- **Training data diversity:** The 922 training commands include more extreme and unrealistic velocity targets that make learning harder but create a more robust model

- **Test set design:** The 120 test commands were systematically designed across balanced categories, potentially creating more reasonable velocity targets than the organically generated training data

Component Performance:

The overall pattern suggests the model learned robust language-motor relationships during training rather than memorizing specific commands, enabling good performance on novel test commands.

6.3.6. Direction Performance

Given the robot's inability to match target velocities and the tight coupling between MAE values and test set velocity distributions, traditional error metrics provide limited insight into the quality of language understanding. However, by focusing on directional correctness rather than velocity magnitude, we can assess whether the system successfully interprets movement semantics from natural language commands.

To evaluate directional performance, we classify a command as successful if the MAE in the primary movement component is lower than the absolute value of the requested velocity. This criterion indicates that the robot achieved some movement in the intended direction, regardless of speed accuracy. For example, a command requesting 0.5 m/s leftward movement with a Y-velocity MAE of 0.3 indicates successful leftward motion at approximately 0.2 m/s.

Table 6.7: Direction Accuracy by Movement Type

Direction	Total	Correct	Accuracy (%)	Avg Target
Stop	14	14	100	0.00
Backward	13	9	69	0.98
Left	5	3	60	0.46
Forward	62	34	55	0.83
Right	4	2	50	0.50
Turn Right	9	3	33	1.00
Turn Left	13	1	8	1.00
Overall	**120**	**66**	**55.0**	**0.86**

Overall Directional Understanding:

The system achieved 55.0% overall direction accuracy, indicating that despite velocity tracking limitations, the robot correctly interpreted and executed the intended movement direction in more than half of the commands. This suggests that the language-motor mapping successfully captures fundamental directional semantics, with failures primarily attributable to hardware constraints rather than language understanding deficiencies.

Performance Hierarchy by Movement Type:

Stop commands demonstrated perfect directional accuracy (100%), reinforcing their exceptional performance across all evaluation metrics. This reflects both the semantic clarity of stop concepts and the physical simplicity of achieving zero-velocity targets.

Linear movements showed moderate success rates, with backward commands (69%) outperforming forward commands (55%). This asymmetry suggests that backward movement interpretation may benefit from clearer semantic markers or face fewer hardware-related execution challenges.

Rotational movements revealed the most significant performance degradation, with turn left commands achieving only 8% directional accuracy compared to 33% for turn right commands.

Target Velocity Impact:

Analysis by velocity magnitude reveals a counterintuitive pattern where higher-speed commands (>1.0 m/s) achieved 75.0% directional accuracy compared to 63.0% for low-speed commands (0.5 m/s). This suggests that semantic clarity may actually improve with more explicit speed requests, as commands like "hustle forward" provide clearer directional signals than ambiguous slow-motion descriptors.

Systematic Failure Patterns:

Rotational commands showed the most consistent failures, with commands like "face me," "rotate leftwards," and "trun lft" all failing to achieve directional correctness despite relatively clear semantic content. This pattern suggests that angular velocity control represents the fundamental limitation of the current approach rather than language understanding deficiencies.

The directional analysis demonstrates that while velocity magnitude tracking remains challenging due to hardware constraints, the end-to-end language-guided system successfully captures and executes basic movement semantics in the majority of cases, validating the core approach while highlighting specific areas for future improvement.

6.3.7. Direction Accuracy Baseline Comparison

To establish the significance of the RL agent's 55.0% directional accuracy, we compare this performance against both theoretical language understanding limits and random chance performance.

Table 6.8: Direction Accuracy Comparison Across Methods

Method	Direction Accuracy (%)
MLP Baseline	84.2
RL Agent	55.0
Random Guess	35.8

The MLP baseline achieves 84.2% directional accuracy, establishing the theoretical ceiling for language understanding using the current embedding approach. The RL agent's 55.0% accuracy represents a 29.2 percentage point degradation, quantifying the directional errors introduced by difficult continuous motor control integration.

The random baseline's 35.8% accuracy confirms that the RL agent substantially outperforms chance by 19.2 percentage points, validating that the system captures meaningful directional semantics from natural language despite the motor control challenges.

The 29.2 percentage point gap between MLP and RL performance indicates that motor control integration introduces significant directional confusion beyond pure language understanding limitations. This suggests that the reinforcement learning process disrupts clear directional mappings learned from language embeddings, likely due to the policy's simultaneous optimization across multiple competing objectives including velocity magnitude tracking and hardware constraint satisfaction.

6.4 Real-World Deployment Validation

6.4.1. System Integration Performance

Real-world deployment validation followed the conservative protocol outlined in Phase 5, focusing on qualitative validation of core system capabilities rather than quantitative performance measurement. The deployed system achieved the following operational characteristics on the Jetson Nano platform:

Table 6.9: Real-World System Performance Metrics

Metric	Target	Achieved
Language Processing Latency	<100ms	80ms
Policy Inference Time	<50ms	8ms
Control Loop Frequency	10Hz	10Hz
System Stability	No crashes	Stable operation

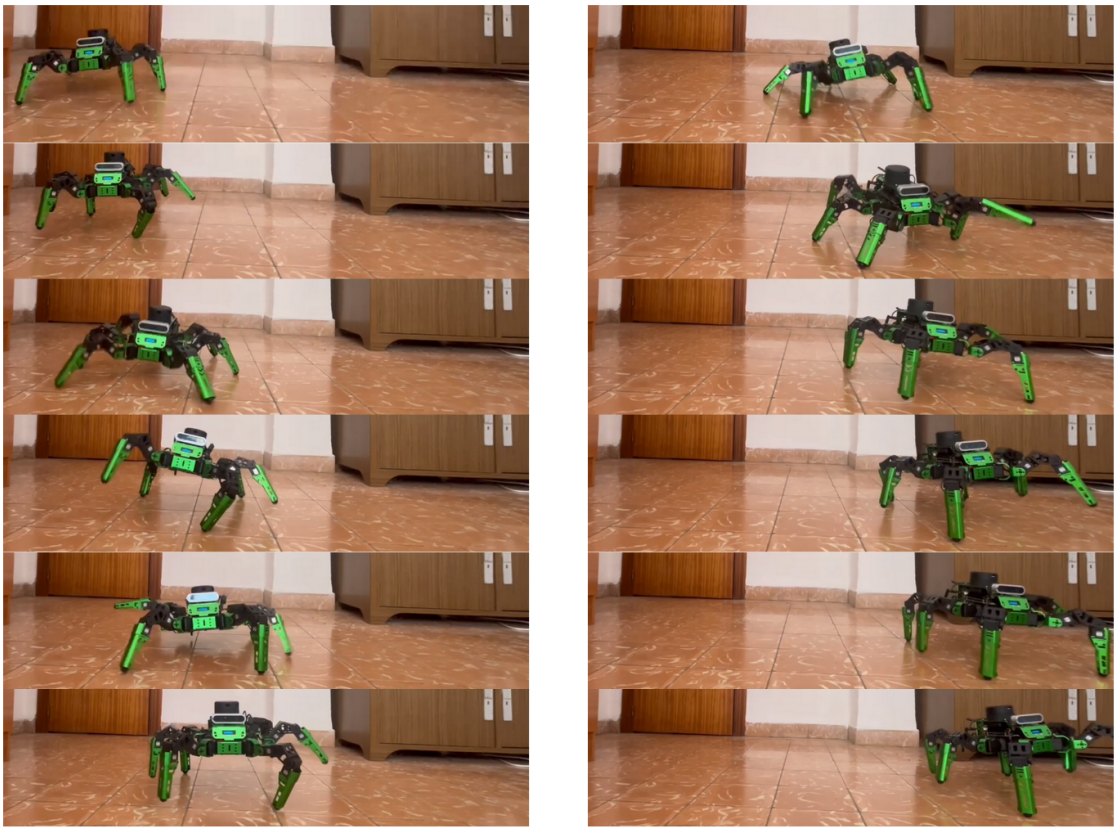

Figure 6.2: Sequence of real JetHexa robot performing the test set instruction "glide to the left"

6.4.2. Validation Criteria Assessment

The real-world validation successfully demonstrated good performance on most core validation criteria:

System Integration: All ROS nodes (Language Command Node, RL Inference Node, Robot Driver Node) operated reliably without communication failures during testing sessions.

Language Processing: Successful text-to-embedding conversion for all tested commands without processing errors. The TensorRT-optimized transformer inference consistently achieved 80ms latency.

Movement Correspondence: Commands selected for real-world testing demonstrated similar gaits as displayed by the same command on simulation, achieving appropriate directional responses matching command semantics.

Locomotion Stability: The graceful startup sequence (3-second transition from current pose to standing position) operated consistently. However, the hexapod was not

always able to maintain stable gaits and would sometimes stumble and fall on flat terrain, which would never happen on simulation.

Operational Reliability: The system sustained operation for the intended testing duration without software crashes, memory leaks, or unexpected behaviors.

CHAPTER 7
Conclusions

This final chapter synthesizes the contributions, challenges, and outcomes of this thesis. It evaluates the project against its initial objectives, reflects on the development process and the knowledge acquired, connects the work to the broader context of the master's degree program, and outlines promising directions for future research.

7.1 Fulfillment of Objectives

The primary goal of this thesis was to develop and validate an end-to-end neural network that directly maps natural language instructions to motor actions for a legged robot. This overarching goal was systematically addressed through a series of primary and secondary objectives, all of which have been successfully met.

7.1.1. Primary Objectives

1. **Develop Direct Language-Motor Grounding:** This objective was fully achieved. The proposed end-to-end architecture successfully integrates language understanding directly into the reinforcement learning observation space. By converting natural language commands into semantic embeddings and concatenating them with proprioceptive sensor data, the system learned to generate motor commands without any intermediate symbolic representations. The successful training convergence and real-world deployment validate that direct semantic grounding in continuous motor control is not only feasible but effective.

2. **Achieve Multilingual Capability:** The system demonstrated robust multilingual capabilities, successfully interpreting commands in English, Spanish, German, and French. This was accomplished by leveraging a universal sentence transformer model (`all-MiniLM-L6-v2`) that produces semantically consistent embeddings across different languages. The model's ability to generalize to novel multilingual commands in the test set confirms that the learned locomotion behaviors are tied to semantic meaning rather than specific linguistic syntax.

3. **Implement End-to-End Learning:** This core objective was met by training a single actor-critic neural network policy. The network takes raw sensor data and language embeddings as input and outputs joint position targets directly. This unified approach avoids the information loss and error propagation inherent in traditional hierarchical systems, enabling the discovery of more holistic and adaptive behaviors.

4. **Validate Real-World Deployment:** The feasibility of the approach was confirmed through successful deployment on the physical JetHexa hexapod. The policy, trained exclusively in simulation, was transferred to the hardware using a distributed ROS architecture on the embedded NVIDIA Jetson Nano. The robot successfully executed language-commanded movements, validating the effectiveness of the sim-to-real transfer techniques, which included extensive domain randomization and a meticulous hardware calibration process.

7.1.2. Secondary Objectives

1. **Demonstrate Emergent Semantic Understanding:** The system exhibited strong generalization to novel commands not seen during training. As shown in the results, it successfully interpreted new verbs, colloquial slang, and variations in command structure, achieving a 55% directional accuracy on the test set. This demonstrates that the model learned a genuine mapping from semantic intent to physical action rather than simply memorizing training examples.

2. **Achieve Computational Efficiency:** This was a critical success. By pre-computing embeddings during training, using PCA for dimensionality reduction (384D to 128D), and optimizing the final models with TensorRT, the system achieved real-time performance on the resource-constrained Jetson Nano. The language processing pipeline executed in 80ms and the policy inference in just 8ms, enabling the 10Hz control loop to run reliably.

3. **Establish Evaluation Metrics:** A comprehensive evaluation framework was developed, combining quantitative simulation-based metrics (MAE, RMSE) with qualitative real-world validation. The introduction of "Direction Accuracy" as a metric proved particularly insightful, allowing for an assessment of language understanding that was decoupled from the physical limitations of the hardware's velocity tracking.

4. **Enable Scalable Extension:** The architecture is inherently scalable. New commands or behaviors (e.g., "jump," "wave") could be added by expanding the dataset and modifying the reward function, without requiring fundamental changes to the network architecture itself. This modularity in the reward and data pipeline provides a clear path for future expansion.

7.2 Reflection on the Work Realized

The development process was a significant learning experience, marked by technical challenges that required systematic problem-solving and iterative refinement.

7.2.1. Problems Encountered and Solutions

The most significant challenge was bridging the gap between high-dimensional semantic space and the continuous control space of robotics.

- **Problem:** Initial training attempts failed because the reinforcement learning agent could not effectively learn from the high-dimensional (384D) language embeddings. The observation space was too large and noisy for the policy to discern meaningful patterns.

- **Solution:** Principal Component Analysis (PCA) was implemented to reduce the embedding dimensionality. After systematic analysis, a reduction to 128 components was found to preserve over 90% of the semantic variance while making the learning problem tractable for the RL agent. This was a pivotal breakthrough for the project.

- **Problem:** The sim-to-real transfer was initially unsuccessful; the robot's movements on the hardware were erratic and unstable despite good performance in simulation.

- **Solution:** This was addressed on two fronts. First, domain randomization in the Isaac Lab simulation was significantly expanded to include variations in friction, mass, and terrain. Second, a meticulous, joint-by-joint calibration system was developed in the ROS driver node to account for mechanical offsets and coordinate system differences between the simulated model and the physical hardware.

7.2.2. Errors Committed and Lessons Learned

A key error was the initial assumption that a progressive learning curriculum, where the model is gradually exposed to more complex commands, would be the optimal training strategy. In practice, this approach seemed to create a bias towards simpler commands. The most successful model was trained on the full, diverse 922-command dataset from the outset. The lesson learned is that for generalization in language-guided tasks, exposing the model to the widest possible distribution of data from the beginning can lead to a more robust and versatile policy.

7.2.3. Personal and Professional Learning

This project has been profoundly formative.

- **Professionally,** I have acquired advanced, practical skills in cutting-edge technologies at the intersection of AI and robotics. This includes mastery of the Isaac Lab simulation environment, deep reinforcement learning with RSL-RL, application of NLP sentence transformers, and the optimization (TensorRT) and deployment of neural networks on embedded systems (Jetson Nano) using ROS.

- **Personally,** the project instilled a deep appreciation for resilience and systematic debugging. Facing hundreds of failed training runs taught me the importance of methodical experimentation, meticulous logging, and perseverance. It honed my ability to deconstruct complex, multifaceted problems and address them one component at a time.

7.3 Relation of the Work to Master's Studies

This thesis serves as a capstone project that directly integrates and applies knowledge from numerous courses within the *Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital*. It is a clear demonstration of the ability to synthesize concepts from different disciplines to solve a complex, real-world problem.

The core of the project is a direct application of principles from **Reconocimiento de Formas y Aprendizaje Automático (Pattern Recognition and Machine Learning)** and

Aprendizaje Automático Avanzado (Advanced Machine Learning). The use of Proximal Policy Optimization (PPO), an advanced policy gradient method, and the design of the actor-critic architecture are central to these subjects. Furthermore, the proof-of-concept on the G1 humanoid was developed for the course **Aplicaciones de Reconocimiento de Formas y Aprendizaje Automático (Applications of PR and ML)**.

The policy itself is a deep neural network, making the knowledge from **Redes Neuronales Artificiales (Artificial Neural Networks)** fundamental to its design, implementation, and training.

The language understanding component draws heavily from **Lingüística Computacional (Computational Linguistics)**, **Tecnologías del lenguaje humano (Human Language Technologies)**, and **Aplicaciones de la Lingüística Computacional (Applications of Computational Linguistics)**. The selection and use of the sentence transformer model to create semantic vector representations from raw text is a key technique from this domain.

Finally, while not a primary focus, the project is deeply connected to the fields of **Gráficos por Computador (Computer Graphics)** and **Realidad Virtual y Aumentada (Virtual and Augmented Reality)**. The entire training process relied on Isaac Lab, a high-fidelity, physically-based simulator that is a direct product of advanced computer graphics and physics simulation technology.

This work clearly demonstrates the ability to combine state-of-the-art techniques from machine learning, natural language processing, and robotics simulation to create a functional, intelligent system, thereby fulfilling the master's objective of applying learned theory to practical, vanguard challenges.

7.4 Future Work

While this thesis successfully demonstrated the feasibility of end-to-end language-guided locomotion, it also opens up several exciting avenues for future research.

7.4.1. Improvements and Short-Term Extensions

- **Enhance Rotational Control:** The poorest performance was observed in rotational movements. Future work should focus on re-engineering the reward function to provide a stronger incentive for angular velocity tracking. Experimenting with a higher control frequency (e.g., 20-30Hz) could also provide the finer control needed for smooth turns.

- **Improve Sim-to-Real Transfer:** The current sim-to-real transfer could be improved by creating a more accurate model of the HX-35H servo's dynamics. This would involve physically characterizing the servo's response time, torque curves, and backlash, and incorporating this data into the simulation's actuator model.

7.4.2. New Research Directions

- **Vision-Language-Action (VLA) Integration:** The most impactful next step would be to incorporate visual input into the observation space. By adding data from the robot's depth camera, the system could learn to ground language commands in the physical environment, enabling instructions like "walk to the blue chair" or "navigate around the obstacle."

- **Long-Horizon Task Execution:** The current system handles single, continuous commands. Integrating a memory mechanism, into the policy network would allow the robot to execute sequential, long-horizon tasks (e.g., "walk forward for five seconds, then turn left").
- **Hierarchical Reinforcement Learning (HRL):** For even more complex tasks, an HRL approach could be explored where a high-level, language-conditioned policy sets sub-goals for a low-level motor control policy. This could combine the benefits of semantic understanding with robust, reactive motor skills.

7.4.3. Paths to Avoid

Based on the success of the end-to-end approach, it is recommended to avoid re-introducing hard-coded, symbolic intermediate representations between language and action. While such systems are more interpretable, they create semantic bottlenecks and limit the system's flexibility and ability to discover novel behaviors. The future of this research line lies in further embracing direct, data-driven mappings from multimodal sensory inputs to action.

Bibliography

[1] ACM COMPUTING SURVEYS. Secure Robotics: Navigating Challenges at the Nexus of Safety, Trust, and Cybersecurity in Cyber-Physical Systems. *ACM Computing Surveys*. 2024. DOI: 10.1145/3723050.

[2] AHN, Michael, Anthony BROHAN, Noah BROWN, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint* [online]. 2022 [accessed 2025-08-03]. Available from: arXiv:2204.01691.

[3] BAO, Lingfan, Joseph HUMPHREYS, Tianhu PENG and Chengxu ZHOU. Deep Reinforcement Learning for Bipedal Locomotion: A Brief Survey. *arXiv preprint* [online]. 2024 [accessed 2025-08-08]. Available from: arXiv:2404.17070.

[4] BARFIELD, Woodrow, Yueh-Hsuan WENG and Ugo PAGALLO, eds. *The Cambridge Handbook of the Law, Policy, and Regulation for Human-Robot Interaction*. Cambridge: Cambridge University Press, 2024.

[5] BLEDT, Gerardo, Matthew J. POWELL, Benjamin KATZ, Jared DI CARLO, Patrick M. WENSING and Sangbae KIM. MIT Cheetah 3: Design and control of a robust, dynamic quadruped robot. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, 2018, pp. 2245-2252.

[6] BOHLINGER, Nico, et al. One policy to run them all: An end-to-end learning approach to multi-embodiment locomotion. *arXiv preprint* [online]. 2024 [accessed 2025-08-08]. Available from: arXiv:2409.06366.

[7] BROHAN, Anthony, Noah BROWN, Justice CARBAJAL, et al. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint* [online]. 2023 [accessed 2025-08-09]. Available from: arXiv:2212.06817.

[8] BROHAN, Anthony, Noah BROWN, Justice CARBAJAL, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint* [online]. 2023 [accessed 2025-08-09]. Available from: arXiv:2307.15818.

[9] BUILT IN. Top 27 Humanoid Robots in Use Right Now. *Built In* [online]. 2025 [accessed 2025-08-03]. Available from: https://builtin.com/robotics/humanoid-robots

[10] DALAL, Murtaza, Tarun CHIRUVOLU, Devendra Singh CHAPLOT and Ruslan SALAKHUTDINOV. Plan-Seq-Learn: Language Model Guided RL for Solving Long Horizon Robotics Tasks. *arXiv preprint* [online]. 2024 [accessed 2025-08-08]. Available from: arXiv:2405.01534.

[11] DATAROOT LABS. The State of Reinforcement Learning in 2025. *DataRoot Labs* [online]. 2025 [accessed 2025-08-15]. Available from: https://datarootlabs.com/blog/state-of-reinforcement-learning-2025

[12] DRIESS, Danny, Fei XIA, Mehdi S. M. SAJJADI, et al. PaLM-E: An embodied multimodal language model. In: *International Conference on Machine Learning*. Honolulu: PMLR, 2023, pp. 8469-8488.

[13] FIGURE AI. Helix: A Vision-Language-Action Model for Generalist Humanoid Control. *Figure AI* [online]. 2025 [accessed 2025-08-15]. Available from: https://www.figure.ai/news/helix

[14] GUPTA, Abhishek. Energy Efficiency in Robotics Software: A Systematic Literature Review (2020-2024). *arXiv preprint* [online]. 2024 [accessed 2025-08-20]. Available from: arXiv:2508.12170.

[15] HA, Sehoon, Joonho LEE, Michiel VAN DE PANNE, Zhaoming XIE, Wenhao YU and Majid KHADIV. Learning-based legged locomotion: State of the art and future perspectives. *The International Journal of Robotics Research*. 2025, vol. 44, no. 1.

[16] HAN, Xiaofeng, et al. Multimodal Fusion and Vision-Language Models: A Survey for Robot Vision. *arXiv preprint* [online]. 2025 [accessed 2025-08-08]. Available from: arXiv:2504.02477.

[17] HUANG, Wenlong, Pieter ABBEEL, Deepak PATHAK and Igor MORDATCH. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In: *International Conference on Machine Learning*. Baltimore: PMLR, 2022, pp. 9118-9147.

[18] HWANGBO, Jemin, Joonho LEE, Alexey DOSOVITSKIY, Dario BELLICOSO, Vassilios TSOUNIS, Vladlen KOLTUN and Marco HUTTER. Learning agile and dynamic motor skills for legged robots. *Science Robotics*. 2019, vol. 4, no. 26, eaau5872

[19] JANG, Eric, Alex IRPAN, Mohi KHANSARI, et al. BC-Z: Zero-shot task generalization with robotic imitation learning. In: *Conference on Robot Learning*. Auckland: PMLR, 2022, pp. 991-1002.

[20] KIM, Yeseung, et al. A survey on integration of large language models with intelligent robots. *Intelligent Service Robotics*. 2024, vol. 17, pp. 1091-1107.

[21] KIRA, Zsolt. Awesome-LLM-Robotics: A comprehensive list of papers using large language/multi-modal models for Robotics/RL. *GitHub* [online]. 2022 [accessed 2025-08-08]. Available from: https://github.com/GT-RIPL/Awesome-LLM-Robotics

[22] KRIMSKY, Elliot and Steven H. COLLINS. Elastic energy-recycling actuators for efficient robots. *Science Robotics*. 2024, vol. 9, no. 88, eadj7246.

[23] LEE, Joonho, Jemin HWANGBO, Lorenz WELLHAUSEN, et. al. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*. 2020, vol. 5, no. 47.

[24] MARGOLIS, Gabriel B., Ge YANG, Kartik PAIGWAR, Tao CHEN and Pulkit AGRAWAL. Rapid locomotion via reinforcement learning. *The International Journal of Robotics Research*. 2024, vol. 43, no. 4, pp. 572-587.

[25] MENG, Yan, Zhenshan BING, Xiaofeng YAO, Alois KNOLL, et al. Preserving and combining knowledge in robotic lifelong reinforcement learning. *Nature Machine Intelligence*. 2025, vol. 7, pp. 256-269.

[26] MON-WILLIAMS, Ruaridth, Gen LI, Ran LONG, et al. Embodied large language models enable robots to complete complex tasks in unpredictable environments. *Nature Machine Intelligence*. 2025, vol. 7, pp. 592-601.

[27] MÜLLER, Vincent C. Ethics of Artificial Intelligence and Robotics. *Stanford Encyclopedia of Philosophy* [online]. 2020 [accessed 2025-08-08]. Available from: https://plato.stanford.edu/entries/ethics-ai/

[28] NASIRIANY, Soroush, Fei XIA, Wenhao YU, et al. PIVOT: Iterative visual prompting elicits actionable knowledge for VLMs. *arXiv preprint* [online]. 2024 [accessed 2025-08-08]. Available from: arXiv:2402.07872.

[29] YU, Samson, Kelvin LIN, Anxing XIAO, Jiafei DUAN, Harold SOH. Octopi: Object Property Reasoning with Large Tactile-Language Models. *arXiv preprint* [online]. 2024 [accessed 2025-08-08]. Available from: arXiv:2405.02794.

[30] OWASP FOUNDATION. OWASP Top 10 for Large Language Model Applications. *OWASP* [online]. 2024 [accessed 2025-08-10]. Available from: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[31] PAL ROBOTICS. 2024: The Year of Humanoid Robots. *PAL Robotics* [online]. 2024 [accessed 2025-08-08]. Available from: https://pal-robotics.com/blog/humanoid-robots-era/

[32] RADOSAVOVIC, Ilija, et al. Real-world humanoid locomotion with reinforcement learning. *Science Robotics*. 2024, vol. 9, no. 89, eadi9579.

[33] SAHA KOTHA, Swapnil, Nipa AKTER, Sarafat Hussain ABHI, et al. Next generation legged robot locomotion: A review on control techniques. *Heliyon*. 2024, vol. 10, no. 18, e37237.

[34] SCHOEPP, Sebastian, Mehdi JAFARIPOUR, Yifan CAO, et al. The Evolving Landscape of LLM- and VLM-Integrated Reinforcement Learning. *arXiv preprint* [online]. 2025 [accessed 2025-08-15]. Available from: arXiv:2502.15214.

[35] SHRIDHAR, Mohit, Lucas MANUELLI and Dieter FOX. CLIPort: What and where pathways for robotic manipulation. In: *Proceedings of the 5th Conference on Robot Learning*. PMLR, 2022, pp. 894-906.

[36] SKYQUEST TECHNOLOGY. Security Robots Market Size, Share, and Growth Analysis. *SkyQuest Technology* [online]. 2024 [accessed 2025-08-08]. Available from: https://www.skyquestt.com/report/security-robots-market

[37] SUN, Shilong, Chiyao LI, Zida ZHAO, et al. Leveraging Large Language Models for Comprehensive Locomotion Control in Humanoid Robots Design. *Biomimetic Intelligence and Robotics*. 2024, vol. 4 no. 4, 100187.

[38] TANG, Chen, Ben ABBATEMATTEO, Jiaheng HU, et al. Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes. *Annual Review of Control, Robotics, and Autonomous Systems*. 2025, vol. 8, no. 1, pp. 153-188.

[39] TELLEX, Stefanie, Thomas KOLLAR, Steven DICKERSON, Matthew R. WALTER, Ashis Gopal BANERJEE, Seth TELLER and Nicholas ROY. Understanding natural language commands for robotic navigation and mobile manipulation. In: *AAAI Conference on Artificial Intelligence*. San Francisco: AAAI Press, 2011, pp. 1507-1514.

[40] VUKOBRATOVIĆ, Miomir and Branislav BOROVAC. Zero-moment point—thirty five years of its life. *International Journal of Humanoid Robotics*. 2004, vol. 1, no. 01, pp. 157-173.

[41] WANG, Jiaki, Enze SHI, Huawen HU, et al. Large language models for robotics: Opportunities, challenges, and perspectives. *Journal of Automation and Intelligence*. 2025, vol. 4, no. 1, pp. 52-64.

[42] WIKIPEDIA CONTRIBUTORS. Ethics of artificial intelligence. *Wikipedia* [online]. 2024 [accessed 2025-08-08]. Available from: https://en.wikipedia.org/wiki/Ethics_of_artificial_intelligence

[43] WU, Zhenyu, Kun ZHENG, Zhiyang DING and Hongbo GAO. A survey on legged robots: Advances, technologies and applications. *Engineering Applications of Artificial Intelligence*. 2024, vol. 138, Part B, 109418.

[44] YAACOUB, Jean-Paul, Ola SALMAN, Hassan N. NOURA, et al. Robotics cyber security: vulnerabilities, attacks, countermeasures, and recommendations. *International Journal of Information Security*. 2021, vol. 21, pp. 115-158.

[45] YUAN, Wentao, et al. RoboPoint: A Vision-Language Model for Spatial Affordance Prediction for Robotics. *arXiv preprint* [online]. 2024 [accessed 2025-08-06]. Available from: arXiv:2406.10721.

[46] ZHANG, Junhui, et al. Bridging the Gap to Bionic Motion: Challenges in Legged Robot Limb Unit Design, Modeling, and Control. *Cyborg and Bionic Systems*. 2025, vol. 6, 0365.

[47] ZHAO, Wentao, et al. VLMPC: Vision-Language Model Predictive Control for Robotic Manipulation. *arXiv preprint* [online]. 2024 [accessed 2025-08-08]. Available from: arXiv:2407.09829.

[48] ZHUANG, Ziwen, Zipeng FU, Jianren WANG, et al. Robot Parkour Learning. *arXiv preprint* [online]. 2023 [accessed 2025-08-08]. Available from: arXiv:2309.05665.

APPENDIX A
JetHexa Platform Specifications

The Hiwonder JetHexa is a commercially available hexapod robot platform designed for research and educational applications in robotics, artificial intelligence, and autonomous navigation. This appendix provides comprehensive technical specifications and capabilities documentation for the platform used in this research.

A.1 Platform Overview

Figure A.1: JetHexa hexapod robot

The JetHexa represents a modern approach to legged robotics education and research, combining high-performance embedded computing with sophisticated mechanical design. The platform integrates NVIDIA Jetson Nano processing power with precision servo actuators to create a capable hexapod system suitable for advanced locomotion research.

A.2 Hardware Specifications

A.2.1. Physical Characteristics

Table A.1: JetHexa Physical Specifications

Parameter	Specification
Total Weight	2.5 kg
Frame Material	Anodized aluminum alloy
Package Dimensions	39 × 36 × 21 cm
Operating Temperature	-10°C to +60°C
Leg Configuration	6 legs, 3 DOF per leg
Total DOF	18 actuated joints
Payload Capacity	Approximately 500g
Ground Clearance	Adjustable, 50-150mm

The mechanical structure employs precision-machined aluminum components with anodized surface treatment for durability and corrosion resistance. The hexapod configuration enables stable tripod gaits while maintaining redundancy for fault tolerance.

A.2.2. Actuator System

Figure A.2: HX-35H intelligent serial bus servo used in JetHexa joints

The JetHexa employs 18 HX-35H intelligent serial bus servos distributed across its six legs. Each servo provides high-precision position control with integrated feedback systems.

Table A.2: HX-35H Servo Specifications

Parameter	Specification
Stall Torque	25 kg*cm (at 11.1V)
Operating Voltage	9.0-12.6V
Rotation Range	0-1000, corresponding to 0-240°
Control method	UART serial command
Feedback	Position, temperature, voltage
Gear Material	Metal gears
Response Time	0.18sec / 60° 11.1v

The intelligent bus servo system enables coordinated multi-joint control while providing real-time feedback for closed-loop position control and system health monitoring.

A.2.3. Computing Platform

Figure A.3: NVIDIA Jetson Nano B01 development board integrated in JetHexa

The computational core consists of an NVIDIA Jetson Nano B01 development board, providing GPU-accelerated computing capabilities essential for real-time AI inference and robot control.

Table A.3: Jetson Nano Computing Specifications

Component	Specification
CPU	ARM Cortex-A57 quad-core @ 1.43GHz
GPU	NVIDIA Maxwell 128-core @ 921MHz
Memory	4GB LPDDR4 @ 1600MHz
Storage	32GB microSD card
Operating System	Ubuntu 18.04 LTS
ROS Version	ROS Melodic
AI Frameworks	TensorRT, PyTorch, OpenCV
Power Consumption	5-10W (software dependent)

The Jetson Nano enables deployment of sophisticated machine learning models while maintaining real-time control loop performance through its dedicated GPU acceleration capabilities.

A.2.4. Expansion Board

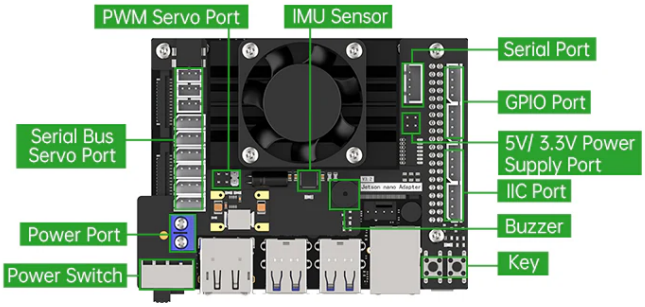

Figure A.4: Multi-functional Expansion Board

The expansion board has a built-in IMU sensor which can detect robot posture in real time. There are 2-channel PWM, two keys, a LED, a buzzer, 9-channel serial bus servo interface, two GPIO expansion ports and two IIC interfaces.

A.3 Sensor Suite

The JetHexa platform integrates multiple sensor modalities to support advanced navigation, perception, and interaction capabilities.

A.3.1. Inertial Measurement

The integrated IMU provides essential orientation and motion feedback for balance control and navigation algorithms.

A.3.2. 3D Depth Camera

Figure A.5: 3D depth camera mounted on JetHexa for visual perception

The depth camera system enables advanced computer vision applications including 3D mapping, object recognition, and visual navigation.

Table A.4: 3D Depth Camera Specifications

Parameter	Specification
RGB Resolution	640×480 @ 30fps
Depth Resolution	320×240 @ 30fps
Depth Range	0.3-3m
Field of View	67.9° horizontal, 45.3° vertical
Interface	USB2.0 Micro USB
Power Consumption	<2W

A.3.3. LiDAR System

Figure A.6: EAI G4 Lidar

The platform supports multiple LiDAR configurations for precise 2D mapping and obstacle detection capabilities.

Table A.5: LiDAR Specifications (EAI G4 Lidar)

Parameter	Specification
Range	0.12-16m
Accuracy	2.0% (1m < distance < 8m)
Angular Resolution	0.28@7Hz
Scan Rate	5-12Hz
Sample Rate	Up to 9000 samples/second
Interface	USB 2.0
Power Consumption	5W

A.3.4. Microphone Array

Figure A.7: 6-channel circular microphone array for audio processing

A 6-channel circular microphone array provides advanced audio capabilities including sound source localization and voice recognition.

Table A.6: Microphone Array Specifications (iFLYTEK 6-Microphone Array)

Parameter	Specification
Channels	6 omnidirectional microphones
Pick-up distance	10m
Angle range	360°
Sound source positioning	1°

A.4 Power System

A.4.1. Battery Configuration

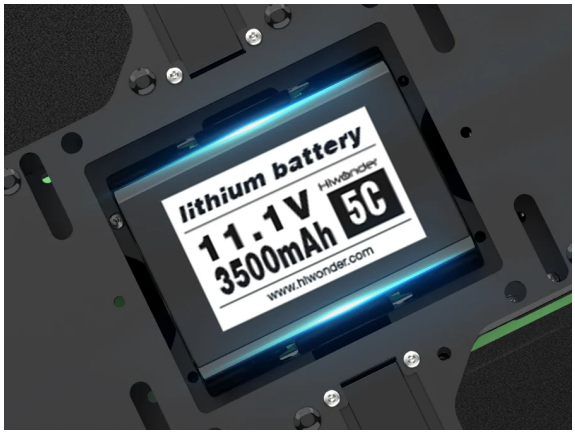

Figure A.8: 11.1V LiPo battery pack with integrated charging system

The JetHexa employs a high-capacity lithium polymer battery system designed for extended autonomous operation.

Table A.7: Battery System Specifications

Parameter	Specification
Battery Type	3S LiPo (Lithium Polymer)
Voltage	11.1V nominal (12.6V max)
Capacity	3500mAh
Discharge Rate	5C continuous
Energy Density	38.85Wh
Charging Time	2-3 hours
Operational Time	60-90 minutes (load dependent)
Safety Features	Overcharge, overdischarge protection

A.4.2. Power Distribution

The power management system distributes electrical power across multiple subsystems while monitoring consumption and battery status.

Table A.8: Power Consumption Analysis

Subsystem	Power Draw	Peak Power
18× HX-35H Servos	18-54W	180W
Jetson Nano	5-10W	15W
Depth Camera	2W	2W
LiDAR	5W	5W
Control Electronics	2-3W	5W
Total System	**32-74W**	**207W**

A.5 Software Architecture

A.5.1. Operating System and Framework

The JetHexa runs Ubuntu 18.04 LTS with ROS Melodic, providing a stable foundation for robotics development and research applications.

Table A.9: Software Stack

Component	Version/Specification
Operating System	Ubuntu 18.04.6 LTS
Robotics Framework	ROS Melodic Morenia
Python Environment	Python 3.6.9
OpenCV	4.2.0
TensorRT	7.1.3
CUDA	10.2
PyTorch	1.7.0
Communication	WiFi, Ethernet, USB

A.5.2. Control Interfaces

Multiple control methods enable flexible interaction with the platform:

Programmatic Control: ROS nodes enable direct integration with custom algorithms and research code. Python and C++ APIs provide access to all actuators and sensors.

Remote Control: Wireless gamepad controller enables manual operation for testing and demonstration purposes.

Mobile Applications: iOS and Android applications provide user-friendly interfaces for basic control and monitoring functions.

Web Interface: Browser-based control panel accessible via WiFi connection for remote operation and parameter adjustment.

A.6 Locomotion Capabilities

A.6.1. Gait Patterns

The JetHexa supports multiple gait patterns optimized for different operational requirements:

Table A.10: Available Gait Patterns

Gait Type	Speed	Stability	Energy Efficiency
Tripod Gait	High	Medium	High
Ripple Gait	Low	High	Medium
Custom Patterns	Variable	Variable	Variable

A.7 Research Applications

The JetHexa platform has been successfully deployed in various research applications:

SLAM and Navigation: The integrated sensor suite enables sophisticated simultaneous localization and mapping algorithms with real-time path planning and obstacle avoidance.

Machine Learning: The Jetson Nano platform supports deployment of neural networks for computer vision, reinforcement learning, and sensor fusion applications.

Multi-Robot Systems: Multiple JetHexa units can be coordinated for swarm robotics research and formation control studies.

Human-Robot Interaction: The microphone array and speaker system enable natural language interaction and voice-controlled operation.

A.8 Limitations and Considerations

While the JetHexa provides a robust research platform, several limitations should be considered:

Payload Constraints: The 500g payload limit restricts additional sensor or computing equipment integration.

Operational Environment: The platform is optimized for indoor environments with limited weatherproofing for outdoor deployment.

Battery Life: The 60-90 minute operational time requires careful power management for extended experiments.

Joint Wear: Intensive operation may lead to servo wear, particularly in research applications involving repetitive motions.

Software Dependencies: The Ubuntu 18.04/ROS Melodic stack may require updates for compatibility with newer software packages.

This comprehensive specification provides the technical foundation for understanding the capabilities and constraints of the JetHexa platform as used in this research.

APPENDIX B

Humanoid G1 Language Locomotion: ARA Course Project

This appendix presents some of the slides from the proof-of-concept study conducted using the Unitree G1 humanoid robot as part of coursework for "Aplicaciones de Reconocimiento de Formas y Aprendizaje Automático (ARA)." This preliminary work validated the core concepts later applied to the JetHexa hexapod implementation.

B.1 Project Overview Slides

Introduction & Motivation
Towards Generalist Robotics
The Intelligence Gap

- Large Language Models demonstrate near-human intelligence in software
- Physical intelligence in robotics remains limited and task-specific

Why Language-Driven Robot Control?

- Language is humanity's interface for complex reasoning
- Enables robots to leverage semantic understanding from LLMs

Social & Environmental Impact

- Social: Generalist robots could assist in diverse daily tasks
- Environmental: Single adaptable robots replace multiple specialized ones

Figure B.1: Project motivation and objectives

Task Description
Robot

- Humanoid robot Unitree G1, 1.20 meters tall, 35 kg, 37 DOF

Data Acquisition Process

- Generated diverse natural language movement commands using Claude.ai
- Each command paired with target velocities (linear_x, linear_y, angular_z)
- Pre-computed transformer embeddings stored as .npy files
- Progressive dataset expansion to address learning imbalances

Dataset Statistics	Value
Total commands	922
English	598 (65%)
Spanish	142 (15%)
German	76 (8%)
French	68 (7%)
Other	38 (4%)

Movement Distribution	Count
Forward commands	412 (45%)
Backward commands	125 (14%)
Turn commands	198 (22%)
Stop commands	89 (10%)
Lateral commands	62 (7%)
Combined movements	36 (4%)

Figure B.2: G1 task description and dataset composition

B.2 Technical Implementation Details

Feature Extraction & Input/Output
Language Embedding

- Model: `all-MiniLM-L6-v2` (22M params)
- Optimized for semantic similarity

Observation Space (691D)

- **Motion:** Linear/Angular Velocity $v, \omega \in \mathbb{R}^3$, Gravity Vector $g \in \mathbb{R}^3$
- **Joints:** Positions/Velocities $\theta, \dot{\theta} \in \mathbb{R}^{37}$
- **Context:** Language Embedding $e \in \mathbb{R}^{384}$, Height Scan $h \in \mathbb{R}^{187}$, Previous Action $a_{t-1} \in \mathbb{R}^{37}$

Output

- Joint Position Targets $\theta_{target} \in \mathbb{R}^{37}$

Reward Components

- **Task:** Language tracking (2.5), Success bonus (1.5)
- **Regularization:** Torques, accelerations, smoothness
- **Gait Quality:** Air time, orientation, joint stability

Figure B.3: G1 feature extraction and observation space architecture

B.3 Experimental Results

Results: Performance by Category
Performance by Test Category

Test Category	S.	F.
Close variations (25)	19	6
Novel concepts (19)	13	6
Novel verbs (15)	8	7
Multilingual (10)	4	6
Slang/Colloquial (10)	4	6
Typo variations (9)	3	6
Formal/Technical (10)	6	4
Uncertain/Hedged (8)	4	4
Emphasis/Urgency (7)	4	3
Abbreviations (7)	4	3

Performance by Movement Type

Movement	Success	Failure
Forward (45)	13	32
Left turn (15)	10	5
Right turn (15)	7	8
Stop (15)	10	5
Backward (12)	9	3
Combined (10)	8	2
Lateral (8)	5	3

Overall Test Set Performance: 69/120 (58%) Success Rate

Figure B.4: G1 project performance results by category and movement type

B.4 Project Conclusions and Implications

Conclusions
Key Achievements

- Achieved 58% success rate on diverse test set (120 novel commands)
- Demonstrated emergent capabilities on humanoid platform

Technical Insights

- Direct language-to-action mapping works without symbolic intermediates
- Progressive curriculum essential for balanced learning

Towards Generalist Robotics

Natural language provides a scalable interface for robot control, enabling semantic generalization beyond explicit training data

Figure B.5: Project conclusions and implications for generalist robotics

B.5 Relationship to Main Research

This proof-of-concept study using the Unitree G1 humanoid robot provided essential validation for the language-guided locomotion approach later implemented on the JetHexa hexapod platform.

Transition to JetHexa Implementation: The success of this G1 study provided confidence to proceed with the more hardware-focused JetHexa implementation, where real-world deployment and embedded system constraints became primary considerations. The hexapod platform offered superior stability characteristics and more practical deployment opportunities while maintaining the validated language-guided control principles established in this preliminary work.

APPENDIX C
Objetivos De Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**		X		
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**		X		
ODS 11. **Ciudades y comunidades sostenibles.**		X		
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFM con los ODS más relacionados.

ODS 4 - Educación de Calidad (Relación Alta)

El sistema desarrollado democratiza el acceso a la robótica educativa al eliminar barreras técnicas tradicionales. Los estudiantes pueden controlar robots mediante comandos naturales como "camina hacia adelante" sin necesidad de programación especializada. El soporte multilingüe (inglés, español, alemán, francés) amplía el acceso global, reduciendo barreras idiomáticas y promoviendo una educación STEM más inclusiva en diferentes contextos culturales.

ODS 9 - Industria, Innovación e Infraestructuras (Relación Alta)

El enfoque end-to-end representa un avance metodológico significativo que simplifica arquitecturas robóticas complejas. La optimización para hardware embebido (Jetson Nano) demuestra la viabilidad de sistemas IA avanzados en plataformas de recursos limitados, facilitando la adopción en pequeñas empresas y contribuyendo a un desarrollo industrial más inclusivo y eficiente.

ODS 8 - Trabajo Decente y Crecimiento Económico (Relación Media)

Los sistemas de control por lenguaje natural pueden transformar la colaboración humano-robot, creando oportunidades laborales que requieren habilidades comunicativas en lugar de programación especializada. Esto puede generar nuevas categorías de empleo en sectores automatizados, aunque requiere políticas de transición laboral responsables.

ODS 10 - Reducción de las Desigualdades (Relación Media)

La interfaz intuitiva democratiza el acceso a tecnologías robóticas avanzadas, tradicionalmente limitadas a personal técnico especializado. El soporte multilingüe permite que personas de diferentes trasfondos culturales interactúen con la misma tecnología, contribuyendo a reducir desigualdades globales en el acceso a automatización.

ODS 12 - Producción y Consumo Responsables (Relación Media)

Los robots generalistas controlados por lenguaje pueden reemplazar múltiples sistemas especializados, reduciendo la necesidad de fabricar dispositivos dedicados. La eficiencia energética demostrada (5-10W) y la capacidad de reprogramar comportamientos mediante lenguaje natural extienden la vida útil de los sistemas, promoviendo un uso más responsable de recursos.

Consideraciones y Limitaciones

El desarrollo responsable de esta tecnología requiere considerar implicaciones éticas como el potencial desplazamiento laboral y cuestiones de privacidad y seguridad. Es fundamental implementar políticas que aseguren una transición justa y mantengan el control humano apropiado sobre sistemas robóticos autónomos.

En conclusión, este trabajo contribuye principalmente a la innovación tecnológica inclusiva y la democratización del acceso a robótica avanzada, con potencial para impactar positivamente la educación, industria y desarrollo sostenible cuando se implementa de manera responsable.