UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Hierarchical Control for Hexapod Robot Navigation: Vision-Guided Motion Planning and Kinematic Gait Generation

Permalink

https://escholarship.org/uc/item/3cr5q6tb

ISBN

9798252451848

Author

Chen, Yitong

Publication Date

2026-06-24

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Hierarchical Control for Hexapod Robot Navigation: Vision-Guided Motion Planning and Kinematic Gait Generation

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Yitong Chen

Committee in charge:

Professor Nicholas Gravish, Chair
Professor Nikolay Atanasov
Professor Sylvia Herbert

2026

Copyright

Yitong Chen, 2026

All rights reserved.

The Thesis of Yitong Chen is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

DEDICATION

To my parents, for their love and support.

To Professor Nikolay Atanasov, for his guidance, insight, and patience throughout this research.

TABLE OF CONTENTS

THESIS APPROVAL PAGE iii
DEDICATION iv
TABLE OF CONTENTS v
LIST OF FIGURES vii
LIST OF TABLES x
ACKNOWLEDGEMENTS xi
ABSTRACT OF THE THESIS xii
Chapter 1 INTRODUCTION AND BACKGROUND 1
1.1 Legged Robot Locomotion 2
1.2 Planning and Control Methods for Legged Robot Navigation 3
1.3 Thesis Contributions and Organization 6
Chapter 2 PROBLEM STATEMENT 9
2.1 Physical Problem Setting 9
2.2 Discrete-Time Control and Optimization Formulation 13
2.3 Hierarchical Solution Strategy and Thesis Scope 15
Chapter 3 KINEMATIC MODELING OF THE HEXAPOD ROBOT 18
3.1 Robot Structure and Coordinate Frames 18
3.2 Screw Axis Representation 23
3.3 Forward Kinematics 24
3.4 Jacobian and Inverse Kinematics 25
3.5 Implementation and Visualization 28
Chapter 4 KINEMATICS-BASED GAIT GENERATION 31
4.1 Method Overview 32
4.2 Geometric Foot-Tip Trajectory Generation 34
4.3 Joint Command Generation via Inverse Kinematics 44
4.4 Learning-Based Approximation of the Command Library 46
4.5 Summary 49
Chapter 5 VISION-GUIDED LEARNING-BASED NAVIGATION 51
5.1 Route-Conditioned Locomotion 52
5.2 MDP Formulation for Planar Waypoint Tracking 53
5.3 Reinforcement Learning for MDP Optimization 57
5.4 Deep Reinforcement Learning Algorithms for Navigation 60
5.5 RRT* Waypoint Planning 74
5.6 Summary 77
Chapter 6 RESULTS AND ANALYSIS 78
6.1 Simulation Results 78
6.2 Real-Robot Deployment 92
6.3 Summary and Analysis 98
Chapter 7 CONCLUSION AND FUTURE WORK 100

7.1 Summary and Engineering Significance 100
7.2 Limitations 101
7.3 Future Work 102
Appendix A SUPPLEMENTARY HARDWARE DESIGN 104
A.1 Hardware Modules for Real-Robot Deployment 104
REFERENCES 106

LIST OF FIGURES

Figure 2.1. Hierarchical control loop. The RRT* planner provides waypoints $\boldsymbol{w}_{m_t}$ from the current robot position $\boldsymbol{p}_t$. The navigation policy maps the full state $\boldsymbol{s}_t$ and waypoint to a body-velocity command $\boldsymbol{a}_t$, which the locomotion layer converts to a joint-command sequence $\boldsymbol{Q}_t^{\text{cmd}}$ applied to the robot. 16

Figure 3.1. Body and leg-base coordinate frames (z-axes out of the page). The global frame $\{\boldsymbol{s}\}$ is the inertial reference, the body-centered frame $\{\boldsymbol{b}\}$ is attached to the robot trunk, and the six leg-base frames $\{\boldsymbol{b}_i\}$ define the mounting positions for the individual leg kinematic chains. 19

Figure 3.2. Coordinate frames along a representative leg (y-axes of $\{\boldsymbol{b}_i\}$, $\{\boldsymbol{l}_{i1}\}$ and z-axes of remaining frames point into the page). From the leg-base frame $\{\boldsymbol{b}_i\}$, successive frames $\{\boldsymbol{l}_{i1}\}$, $\{\boldsymbol{l}_{i2}\}$, and $\{\boldsymbol{l}_{i3}\}$ are attached to the coxa, femur, and tibia, with $\{\boldsymbol{e}_i\}$ at the foot-tip end-effector. 22

Figure 3.3. The visualization of the hexapod model generated from the kinematic description. 30

Figure 4.1. Model-based command-generation pipeline. Motion command (ω, v) and period parameters (T, N_{c}) feed a geometric trajectory generator, whose foot-tip output $\boldsymbol{P}_{\text{tip}} \in \mathbb{R}^{18\times N_{\text{c}}}$ is converted by inverse kinematics to the joint-command trajectory $\boldsymbol{Q} \in \mathbb{R}^{18\times N_{\text{c}}}$ for actuator execution. 32

Figure 4.2. Leg-indexing convention (top view). Right-side legs are numbered $1, 2, 3$ front-to-rear; left-side legs are numbered $4, 5, 6$ front-to-rear. 35

Figure 4.3. Tripod-gait schedule over three repeated cycles. Legs $\mathcal{T}_1 = \{1, 3, 5\}$ and $\mathcal{T}_2 = \{2, 4, 6\}$ are offset by half a cycle, with the origin $g = 0$ at mid-stance of $\mathcal{T}_1$ and mid-swing of $\mathcal{T}_2$. 36

Figure 4.4. Straight-motion foot-tip trajectory in body frame $\{\boldsymbol{b}\}$. The stance tip, fixed in world frame $\{\boldsymbol{s}\}$, traces a reversed straight path centered at mid-stance; the swing phase returns the tip along the same line. 39

Figure 4.5. Turning-motion foot-tip trajectory in body frame $\{\boldsymbol{b}\}$. The stance tip, fixed in world frame $\{\boldsymbol{s}\}$, traces a circular arc about the instantaneous center $\boldsymbol{c}$ with reversed angular direction; the swing phase returns along the same arc. 40

Figure 4.6. Normalized phase plane for one leg. Coordinate $\mu \in [-1, 1]$ is planar progression and $\zeta \in [0, 1]$ is normalized lift height. The foot moves $\mu : 1 \to -1$ on $\zeta = 0$ during stance and $\mu : -1 \to 1$ with peak lift at mid-swing. 43

Figure 5.1. Route-conditioned navigation loop. A planner generates a waypoint sequence from the start, goal, and map; a waypoint manager advances or holds the active target; the learned policy closes the inner loop using vision, proprioception, and the active waypoint. 52

Figure 5.2. Policy network architecture. Branch encoders compress visual, depth, proprioceptive, and target inputs into feature vectors fused by the policy MLP, then passed through a Tanh squash to produce the body-velocity command $\boldsymbol{a}_t = [v_t, \omega_t]^\top$. 61

Figure 5.3. Value network architecture. The same input-branch organization as Fig. 5.2 with separate parameters $\boldsymbol{\phi}$; the MLP output is a scalar state-value estimate $V_{\boldsymbol{\phi}}(\boldsymbol{s}_t)$ with no squash stage. 62

Figure 5.4. Mixture-of-experts coordinator policy. Each expert produces a candidate command from the current state; the coordinator scores the experts using the state and candidates; the selector outputs the final command. 67

Figure 5.5. RECAP training flow. Online, replay, and teacher data are split between the Q-function branch (source-aware targets, twin-critic update) and the policy branch (source-aware advantages, clipped PPO update). Dashed arrows indicate quantities carried to the next cycle. 74

Figure 6.1. Training map layout. The environment uses a bounded $30\,\mathrm{m} \times 30\,\mathrm{m}$ workspace with high outer walls and an obstacle arrangement formed from rectangular and cylindrical primitives. 79

Figure 6.2. MuJoCo training environment visualization. The scene shows the hexapod robot and obstacle geometry used in the simulated rollout environment. 80

Figure 6.3. Visual observation at a single time step (64×48 pixels per channel). Rows 1–2 show the seven scalar network inputs: R, G, B channels of the left view $\boldsymbol{I}_t^{\mathrm{L}}$, right view $\boldsymbol{I}_t^{\mathrm{R}}$, and depth $\boldsymbol{D}_t$. Row 3 shows composite RGB images for visual reference only; not fed to the network. 81

Figure 6.4. Training curves of the four learned navigation policies. PPO Continuous, PPO-MoE, RECAP Continuous, and RECAP-MoE are compared using average episode return, success rate, collision step rate, and average episode length over the first 200 training iterations. 83

Figure 6.5. Representative testing scene and start–goal pairs. Triangular markers indicate initial robot poses and heading directions; circular markers indicate goal positions. Obstacle colors are varied to test policy robustness to visual appearance changes. 85

Figure 6.6. Qualitative trajectory comparison on a testing map. Each subplot overlays direct (circular markers) and waypoint-guided (triangular markers) executions for the same start–goal pair; the yellow curve shows the planned route. Red highlighting indicates contact with an obstacle. 89

Figure 6.7. Expert-selection diagnostics for PPO-MoE. Each subplot shows the expert probabilities, the actually selected expert at each trajectory step, and the percentage of steps assigned to each expert. 90

Figure 6.8. Expert-selection diagnostics for RECAP-MoE. Each subplot shows the categorical policy distribution, sampled expert decisions per step, and expert usage share over the trajectory. 91

Figure 6.9. Motion-capture-assisted real-robot deployment loop. Motion capture measures robot and obstacle poses; the scene is reconstructed in MuJoCo to generate policy inputs $\boldsymbol{\zeta}_t$ (visual/depth), $\boldsymbol{\eta}_t$ (proprioceptive), and $\boldsymbol{g}_t$ (target state); the controller returns joint-command sequences to the robot. 93

Figure 6.10. Static-obstacle deployment sequence. Each column pairs the physical robot snapshot (top) with the MuJoCo reconstruction used for policy observation (bottom). Blue boxes are obstacles, the orange cone is the goal, and red trajectory segments indicate contact. 95

Figure 6.11. Representative waypoint-guided deployment sequence. For each time step, the physical robot snapshot is paired with the corresponding reconstructed scene used for waypoint-conditioned policy execution. 96

Figure 6.12. Representative moving-obstacle deployment sequence. For each time step, the physical robot snapshot is placed above the corresponding reconstructed scene used by the policy observation and command update. 97

LIST OF TABLES

Table 3.1. Geometric parameters defining the six leg-base installation positions 21

Table 6.1. Training data settings for the four learned policies. 82

Table 6.2. Testing metrics in the direct-goal setting. 86

Table 6.3. Testing metrics in the waypoint-guided setting. 86

Table A.1. Hardware modules used in the real-robot deployment platform. 105

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Nikolay Atanasov for his mentorship and support throughout this research. He consistently encouraged me to pursue my own ideas and develop my thinking independently, while providing the computational resources, laboratory equipment, and access to advanced tools that made this work possible. Whenever I found myself stuck, his guidance helped me find a clear direction forward.

I would also like to thank Professor Nicholas Gravish and Professor Sylvia Herbert for serving on my committee. Their feedback and perspectives have broadened my thinking and strengthened the writing of this thesis.

I would like to thank Chenyang Yan, Kaifeng Chen, and Guanxiang Jing for their effort and dedication in the hardware design, electronics design, and assembly of the physical robot platform. We worked on this project together for over a year, and I deeply treasure the time and experiences we shared throughout this journey.

I would also like to thank Viswa Subramanian and Darin Djapri for their thoughtful ideas and suggestions they shared throughout this project.

ABSTRACT OF THE THESIS

Hierarchical Control for Hexapod Robot Navigation: Vision-Guided Motion Planning and Kinematic Gait Generation

by

Yitong Chen

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2026

Professor Nicholas Gravish, Chair

This thesis develops a hierarchical control approach for hexapod robot navigation, consisting for a high-level vision-guided motion planning and a low-level kinematic gait generation. The low-level controller converts body-velocity commands into full eighteen-degree-of-freedom joint trajectories using inverse kinematics and a kinematics-based tripod gait generator. The high-level motion planner selects body-velocity commands from visual, depth, proprioceptive, and target-state inputs using reinforcement learning. A proximal policy optimization (PPO) baseline for body-velocity planning is first established. Two extensions are then proposed to

improve the learning performance: a mixture-of-experts (MoE) architecture for multi-modal body-velocity command selection, and Replay-Enhanced Clipped Actor-Critic Policy Learning (RECAP) for improved sample efficiency. Route-level planning via a rapidly-exploring random tree with rewiring (RRT*) planner is also integrated to improve the safety and quality of the generated motion plans. The integrated approach, including RRT* planning, body-velocity generation via MoE and RECAP, and low-level gait control is demonstrated in simulation and on a physical hexapod robot across cluttered and moving-obstacle environments.

Chapter 1

Introduction and Background

Legged robots have long been studied as mobile systems for environments where continuous wheel contact is difficult to maintain. Unlike wheeled platforms, legged machines move through discrete ground contacts and must coordinate multiple limbs to generate body motion. This gives them the ability to step over local obstacles and redistribute support across several legs, but it also makes locomotion execution more difficult: even a simple desired body motion must be realized through gait timing, swing and stance phases, foot trajectories, and coordinated joint commands.

Motivated by this difficulty, this thesis studies a modular framework that connects learning-based navigation with model-based locomotion. The navigation policy is responsible for selecting compact body-level commands, while the locomotion layer expands these commands into executable gait and joint motion. This separation avoids learning one large policy from sensory input directly to all joint commands, and instead uses the compact body-command interface as a bridge between decision making and locomotion execution.

1.1 Legged Robot Locomotion

This section reviews the legged-robot research background needed for the thesis. Legged robots include bipedal robots, humanoid robots, quadrupedal robots, hexapod robots, and other multi-legged systems. Across these morphologies, locomotion is organized around intermittent ground contacts rather than continuous wheel contact. The main locomotion problems therefore include balance, gait cycles, contact timing, swing and stance coordination, foot placement, terrain interaction, and joint-level execution. Classical work on legged locomotion showed that body motion and contact timing must be considered together in legged control [1]. Studies of biological and robotic gaits also show that legged movement is often organized through repeatable timing patterns rather than arbitrary limb motion [2].

Different legged morphologies emphasize different technical challenges, and the hexapod form has strengths that cannot be directly replaced by bipedal or quadrupedal designs. The main advantage is support redundancy: several legs can remain in stance while other legs swing. This property makes structured gait patterns, such as tripod and wave gaits, natural choices for stable and interpretable locomotion. Early multilegged robot research studied adaptive locomotion over rough terrain [3], behavior-based six-legged robots showed how layered local behaviors can generate walking in complex environments [4], and the RHex platform demonstrated that mechanically informed hexapod locomotion can produce robust mobility on obstacle-rich terrain [5]. These studies motivate the use of hexapod morphology as a meaningful platform for studying stable, redundant, and obstacle-aware legged motion.

Structured gait generation remains important for hexapod robots because it provides a

clear relation between desired body motion and coordinated leg motion. Biological studies of leg coordination have influenced robotic gait design [6], and central-pattern-generator approaches provide another model for rhythmic legged motion [7]. Mechanical studies of legged locomotion also emphasize the relation among morphology, contact, and control [8]. For the present thesis, structured gait generation is useful not because tripod gait or inverse kinematics is new, but because it provides an interpretable way to transform compact body commands into feasible multi-leg motion.

Stable locomotion capability, however, is not the same as autonomous navigation. A gait can generate motion, but it does not decide how local sensing and obstacles should affect the next command. Recent visual-proprioceptive legged navigation work shows that perception can be integrated with locomotion and local planning for goal-directed movement [9]. This motivates the connection studied here between learning-based navigation and model-based locomotion.

1.2 Planning and Control Methods for Legged Robot Navigation

This section reviews the main method families used for legged robot navigation: route planning, model-based locomotion, and learning-based control. The goal is to clarify what each method provides and why their interfaces matter for executable robot behavior.

Route-level planning addresses the question of where the robot should go. Classical graph-search methods plan over a discretized environment; A* is the canonical example, using an admissible heuristic to guarantee both optimality and completeness on the discretization [10]. Sampling-based planning instead avoids explicit discretization, providing a general way to construct geometric routes through obstacle spaces [11]. Rapidly-exploring random trees

(RRTs) are a representative family of sampling-based planners [12]. RRT* extends this idea with asymptotic optimality under suitable assumptions [13]. In a legged robot navigation task, these methods can provide route guidance or waypoint sequences, but a geometric route alone is not a closed-loop legged execution method. Recent legged-planning work narrows this gap by coupling route-level search with reduced-order dynamics: kinodynamic sampling-based planners select footsteps and body dynamics jointly so that plans remain dynamically feasible and can react to moving obstacles [14], and stability-aware traversability estimation lets route decisions account for where a legged robot can maintain balance rather than only where it can avoid geometric collision [15].

Model-based locomotion and control focus on the execution problem: given a desired body motion or foot trajectory, the robot must express that motion in consistent body and leg frames and convert it into joint commands. For position-level execution, inverse kinematics converts desired foot positions into joint commands, providing the geometric bridge from body-level motion descriptions to executable leg motion [16, 17]. These kinematic and dynamic models can also be embedded in model predictive control (MPC), where future robot motion is predicted and a constrained finite-horizon optimization problem is solved online. MPC has been applied to whole-body control through contacts [18] and to dynamic legged locomotion [19], including recent formulations that embed high-level task and safety specifications via signal temporal logic [20]. This gives model-based control strong physical structure, but optimizing constraints, contacts, and dynamics online can be computationally demanding.

Learning-based control rests on two foundational paradigms. Supervised learning fits a model to labeled input-output pairs by minimizing a regression or classification loss [21]; it

is effective when high-quality labeled data are available but the learned behavior is bounded by the coverage of the training set. Reinforcement learning (RL) instead formulates control as sequential decision making: an agent interacts with an environment, receives scalar reward feedback, and updates its policy to maximize cumulative return [22]. RL does not require labeled demonstrations and can discover behaviors that exceed the quality of any provided data, but it requires carefully designed reward signals and is often sample-intensive.

In robotics, supervised learning frequently appears as learning from demonstration, where expert trajectories provide regression targets for the policy [23]. A key limitation is that a regression objective averages over demonstrated behaviors, which degrades performance when multiple valid actions exist for a given state. Generative policy learning addresses this by modeling the full conditional distribution over actions rather than its mean. Diffusion-based policies are a prominent recent instance: the policy is a denoising diffusion model conditioned on the current state, capturing multimodal action distributions and enabling diverse skill generation. DiffuseLoco applies this approach to legged locomotion, training a diffusion policy from offline demonstration datasets to produce real-time, diverse locomotion skills across quadrupedal and bipedal platforms [24].

Reinforcement learning for robot control has been advanced by several algorithmic families. Value-based methods learn action-value estimates to guide decision making [25]. Policy-gradient methods directly optimize a parameterized policy using gradient estimates [26]. Actor-critic methods combine both, using a learned value baseline to reduce gradient variance and stabilize updates [27]. Proximal policy optimization (PPO) builds on this with a clipped surrogate objective that prevents excessively large policy changes and has become a standard algorithm in

practice [28]. A natural extension combines RL with demonstrations: rather than treating demos as strict supervision targets, they serve as a motion prior while task-specific rewards drive further optimization. DeepMimic realizes this by combining a reference-motion tracking reward with a task reward in a physics-based RL loop, enabling physically natural and goal-directed motion [29]. Adversarial Motion Priors (AMP) refines the idea by replacing per-frame pose matching with an adversarial style discriminator, providing flexible demo guidance that can be combined with arbitrary task objectives such as balance, energy efficiency, and obstacle avoidance [30]. In this thesis, RL is used at the command-selection level rather than as direct full-joint control, so the policy does not need to learn navigation, gait coordination, and joint-level execution simultaneously.

Recent integrated legged-navigation systems show the importance of combining perception, planning, and locomotion. Visual-proprioceptive navigation work demonstrates that local sensing can guide legged robots toward goals while accounting for locomotion behavior [9]. Perceptive locomotion research also shows that terrain-aware feedback is important for real-world legged motion [31, 32, 33]. The remaining issue addressed here is the interface among route guidance, local sensing, learned command selection, and model-based hexapod gait execution. This thesis studies a compact body-command interface that connects these components without requiring the learned policy to output all joint commands directly.

1.3 Thesis Contributions and Organization

This thesis focuses on flat, obstacle-rich waypoint navigation for a hexapod robot. The goal is not to solve full simultaneous localization and mapping, uneven-terrain autonomy, highly

dynamic whole-body locomotion, or product-level autonomy. Instead, the work studies how learning-based navigation and model-based locomotion can be connected for a specific hexapod waypoint-navigation system.

The system organization follows the layered structure introduced above. Route-level guidance may provide a sequence of waypoints. A learned command policy uses local visual, depth, proprioceptive, and target information to select compact body commands, such as forward velocity and yaw rate. A model-based gait generator maps those commands to coordinated leg motion, and inverse kinematics converts the resulting foot trajectories into joint commands. In this way, the learning problem is focused on navigation-level command selection, while locomotion-level execution remains structured and interpretable.

The main contributions are summarized as follows:

1. A hierarchical formulation for vision-guided hexapod waypoint navigation that connects learning-based navigation with model-based locomotion through a compact body-command interface.

2. A command-to-joint locomotion layer that combines model-based gait generation with supervised approximation. A tabular command mapping is fast but limited to discrete velocity and yaw-rate commands, while full online computation is more flexible but computationally expensive. The approach used here generates model-based command data and learns a continuous command-to-joint mapping function, so compact body commands can be converted to joint trajectories efficiently.

3. Learning-based navigation algorithms are developed for compact command selection. The

study begins with a PPO baseline, introduces mixture-of-experts (MoE) command selection [34], and develops Replay-Enhanced Clipped Actor-Critic Policy Learning (RECAP) to improve sample efficiency through replay.

4. A physical hexapod platform is designed, assembled, and used for deployment-oriented validation. The real-robot tests verify executable locomotion behavior and provide a basis for extending the learned navigation framework beyond simulation.

The rest of the thesis is organized as follows. Chapter 2 formulates the navigation and locomotion problem formally and identifies the main challenges that the thesis addresses. Chapter 3 develops the hexapod kinematic model and the basic geometric concepts used in later chapters. Chapter 4 presents model-based locomotion, joint-command generation, and function approximation for continuous command-to-joint mapping. Chapter 5 introduces the integration of waypoint planning and learning-based navigation. Chapter 6 presents simulation and deployment-oriented evaluation. Chapter 7 summarizes the project, discusses limitations, and proposes future work to address them.

Chapter 2

Problem Statement

Before introducing the specific modeling, gait-generation, and learning methods, this chapter defines the underlying control problem addressed by the thesis. The chapter first describes the physical navigation setting, then formulates the corresponding discrete-time control objective, and finally explains why a hierarchical solution is adopted.

2.1 Physical Problem Setting

The task considered in this thesis is target-directed locomotion for a hexapod robot in a flat, obstacle-rich environment. The robot must reach a specified goal position while satisfying several concurrent requirements: avoiding collisions with external obstacles; preventing self-collision among its own legs and body links; maintaining dynamic balance throughout locomotion; and operating efficiently with respect to energy consumption and path length. Safety, stability, and efficiency are therefore integral objectives alongside goal-reaching.

Environment map.

The workspace is modeled as a bounded planar region containing static obstacles. Formally, the map is the pair

$$\mathcal{M}_{\text{map}} = (\mathcal{X}_{\text{work}},\, \mathcal{X}_{\text{obs}})\,, \tag{2.1}$$

where $\mathcal{X}_{\text{work}} \subset \mathbb{R}^2$ is the bounded planar workspace and $\mathcal{X}_{\text{obs}} \subset \mathcal{X}_{\text{work}}$ is the static obstacle region. The collision-free navigable area follows as

$$\mathcal{X}_{\text{free}} = \mathcal{X}_{\text{work}} \setminus \mathcal{X}_{\text{obs}}. \tag{2.2}$$

Robot model.

The robot $\mathcal{R}$ is a hexapod with a rigid trunk and six articulated legs. Each leg is a serial chain of three rigid links — coxa, femur, and tibia — connected in sequence from the body trunk to the foot tip by three revolute joints, giving 18 actuated degrees of freedom in total. The joint configuration is

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_1^\top & \cdots & \boldsymbol{q}_6^\top \end{bmatrix}^\top \in \mathbb{R}^{18}, \qquad \boldsymbol{q}_i \in \mathbb{R}^3, \tag{2.3}$$

where $\boldsymbol{q}_i$ collects the three joint angles of leg i. The full kinematic model is developed in Chapter 3; here $\mathcal{R}$ denotes the robot as a physical entity whose body pose and joint state evolve when joint commands are applied in simulation.

In general the body has a six-dimensional $SE(3)$ pose. Since this thesis considers only flat, level terrain, the roll, pitch, and vertical position remain approximately constant, reducing

the effective body configuration to three degrees of freedom:

$$\boldsymbol{c}_t^{\text{body}} = \begin{bmatrix} x_t & y_t & \psi_t \end{bmatrix}^\top \in \mathcal{X}_{\text{free}} \times \mathbb{S}^1, \tag{2.4}$$

where (x_t, y_t) is the planar position of the body origin and ψ_t is the yaw angle. Together with the joint angles, the complete robot configuration is

$$\boldsymbol{c}_t = \left(\boldsymbol{c}_t^{\text{body}},\ \boldsymbol{q}_t \right) \in \left(\mathcal{X}_{\text{free}} \times \mathbb{S}^1 \right) \times \mathbb{R}^{18}. \tag{2.5}$$

Initial configuration and goal position.

Let ρ_0 be a probability measure over $\mathcal{X}_{\text{free}} \times \mathbb{S}^1$ and ρ_g be a probability measure over $\mathcal{X}_{\text{free}}$. The initial body configuration and the goal position are drawn from these distributions:

$$\boldsymbol{c}_0^{\text{body}} \sim \rho_0(\,\cdot \mid \mathcal{M}_{\text{map}})\,, \qquad \boldsymbol{p}_{\text{goal}} \sim \rho_g(\,\cdot \mid \mathcal{M}_{\text{map}})\,, \tag{2.6}$$

where both distributions are conditioned on the map $\mathcal{M}_{\text{map}}$ to ensure the samples lie within the collision-free region. No constraint is placed on the heading upon arrival at the goal.

Robot state representation.

The information available to the controller at time step t is organized into three groups:

$$\boldsymbol{s}_t = (\boldsymbol{\zeta}_t,\, \boldsymbol{\eta}_t,\, \boldsymbol{g}_t)\,, \tag{2.7}$$

where $\boldsymbol{\zeta}_t$ is visual sensing, $\boldsymbol{\eta}_t$ is proprioception, and $\boldsymbol{g}_t$ encodes the target direction and distance.

The visual component collects stereo RGB images and a depth estimate:

$$\boldsymbol{\zeta}_t = \left(\boldsymbol{I}_t^{\mathrm{L}},\, \boldsymbol{I}_t^{\mathrm{R}},\, \boldsymbol{D}_t\right), \tag{2.8}$$

where $\boldsymbol{I}_t^{\mathrm{L}}, \boldsymbol{I}_t^{\mathrm{R}} \in \mathbb{R}^{H_I \times W_I \times 3}$ are the left and right RGB images and $\boldsymbol{D}_t \in \mathbb{R}^{H_D \times W_D}$ is the depth estimate.

The proprioceptive vector stacks body-frame angular velocity, linear velocity, their finite differences, and the previous command:

$$\boldsymbol{\eta}_t = \begin{bmatrix} \left({}^{\boldsymbol{b}}\boldsymbol{\omega}_t\right)^\top & \left({}^{\boldsymbol{b}}\boldsymbol{v}_t\right)^\top & \left({}^{\boldsymbol{b}}\Delta\boldsymbol{\omega}_t\right)^\top & \left({}^{\boldsymbol{b}}\Delta\boldsymbol{v}_t\right)^\top & \boldsymbol{a}_{t-1}^\top \end{bmatrix}^\top. \tag{2.9}$$

The target input encodes the bearing and distance to the active waypoint in the body frame. Let ${}^{\boldsymbol{s}}_{\boldsymbol{b}}\boldsymbol{R}_t \in SO(3)$ be the rotation from the body frame to the world frame. The body-frame displacement to the active waypoint $\boldsymbol{w}_{m_t}$ and its norm are

$${}^{\boldsymbol{b}}\boldsymbol{r}_t = {}^{\boldsymbol{s}}_{\boldsymbol{b}}\boldsymbol{R}_t^\top (\boldsymbol{w}_{m_t} - \boldsymbol{p}_t), \qquad \ell_t = \left\| {}^{\boldsymbol{b}}\boldsymbol{r}_t \right\|_2. \tag{2.10}$$

The target input to the policy is then

$$\boldsymbol{g}_t = \begin{bmatrix} \cos\psi_t^{\boldsymbol{g}} & \sin\psi_t^{\boldsymbol{g}} & \ell_t \end{bmatrix}^\top \in \mathbb{R}^3, \qquad \psi_t^{\boldsymbol{g}} = \operatorname{atan2}\left({}^{\boldsymbol{b}}r_{t,2},\, {}^{\boldsymbol{b}}r_{t,1}\right), \tag{2.11}$$

where $\psi_t^{\boldsymbol{g}}$ is the bearing from the robot to the active waypoint in the body frame.

2.2 Discrete-Time Control and Optimization Formulation

With the physical setting fixed, the navigation task is written as a discrete-time control problem with terminal conditions. Let $\boldsymbol{s}_t \in \mathcal{S}$ denote the robot state at discrete time t. In the direct joint-level formulation, the control input is the commanded joint-angle vector

$$\boldsymbol{u}_t = \boldsymbol{q}_t^{\text{cmd}} \in \mathbb{R}^{18}, \tag{2.12}$$

where the dimension corresponds to the eighteen actuated joints of the hexapod. A policy maps the available state information to this joint command,

$$\boldsymbol{u}_t \sim \pi(\cdot \mid \boldsymbol{s}_t). \tag{2.13}$$

After the command is applied, the robot evolves according to

$$\boldsymbol{s}_{t+1} \sim P(\cdot \mid \boldsymbol{s}_t, \boldsymbol{u}_t; \mathcal{M}_{\text{map}}, \mathcal{R}), \tag{2.14}$$

where P represents the state transition model induced by the robot model, actuation, contact interaction, obstacle geometry, and sensing or execution uncertainty. The transition is written abstractly because this chapter is concerned with the problem-level formulation rather than a closed-form dynamics model.

The robot is assumed to continue moving until the state enters a terminal subset. The terminal set is written as

$$\mathcal{S}_{\text{term}} = \mathcal{S}_{\text{goal}} \cup \mathcal{S}_{\text{out}}, \tag{2.15}$$

where $\mathcal{S}_{\text{goal}}$ contains successful goal-reaching states, and $\mathcal{S}_{\text{out}}$ contains out-of-bound states. Obstacle contact — including both external obstacle collision and self-collision — is treated as a soft penalty in the edge cost rather than as a hard termination signal. The episode therefore continues after a collision, which allows the policy to learn recovery behaviors: the robot is implicitly encouraged to disengage from contact and resume progress toward the goal rather than treating any collision as an irrecoverable failure. The penalty weight is set large enough to strongly discourage contact, while still allowing the optimizer to discover recovery trajectories. The corresponding stopping time is

$$\tau = \inf\{t \geq 0 : \boldsymbol{s}_t \in \mathcal{S}_{\text{term}}\}. \tag{2.16}$$

This structure defines a *first exit Markov decision process (MDP)*: the episode runs until the state first enters $\mathcal{S}_{\text{term}}$, at which point the process absorbs. The stopping time τ is determined by the trajectory itself rather than a fixed horizon.

Because the initial configuration and goal vary across problem instances, the objective is not to optimize a single trajectory but to find a policy π that performs well on average over all configurations drawn from ρ_0 and all goals drawn from ρ_g. Formally, the optimal policy minimizes the expected discounted cost up to termination,

$$\begin{aligned}\pi^\star = \arg\min_{\pi} \; \mathbb{E}_{\substack{\boldsymbol{c}_0^{\text{body}} \sim \rho_0,\, \boldsymbol{p}_{\text{goal}} \sim \rho_g \\ \boldsymbol{u}_t \sim \pi,\, \boldsymbol{s}_{t+1} \sim P}} \Bigg[& \sum_{t=0}^{\tau-1} \gamma^t c\big(\boldsymbol{s}_t, \boldsymbol{u}_t, \boldsymbol{s}_{t+1}; \mathcal{M}_{\text{map}}, \boldsymbol{p}_{\text{goal}}\big) \\ & + \gamma^\tau c_{\text{term}}\big(\boldsymbol{s}_\tau, \boldsymbol{p}_{\text{goal}}\big) \Bigg].\end{aligned} \tag{2.17}$$

Here $c(\cdot)$ is an edge cost assigned to each transition and $c_{\text{term}}(\cdot)$ is a terminal cost applied at

stopping time τ. The discount factor $\gamma \in (0, 1)$ weights future costs relative to immediate ones. The edge cost aggregates several navigation-relevant terms: the distance from the robot to the goal position; a penalty for collision with external obstacles; a penalty for self-collision among the robot's own legs and body links; a penalty for deviation from an upright balanced posture; a penalty for leaving the valid workspace; the mechanical energy consumed by the actuators over the transition; the magnitude of the commanded velocity; and the change in command between consecutive steps for smoothness. The terminal cost assigns a success bonus when $\boldsymbol{s}_\tau \in \mathcal{S}_{\text{goal}}$ and a failure penalty when $\boldsymbol{s}_\tau \in \mathcal{S}_{\text{out}}$. The specific weights and functional forms of each term are defined in Chapter 5. Practical training may impose a maximum rollout length as an implementation truncation, but τ in Eq. (2.17) is the theoretical first exit time.

The formulation in Eq. (2.17) describes the original control problem, but solving it directly at the joint level is difficult. The control space is high-dimensional, the contact behavior is nonlinear and hybrid, and the same decision variable must represent locomotion generation, body stabilization, obstacle avoidance, and goal reaching. This coupling motivates a structured solution in which navigation decisions and locomotion execution are connected through an intermediate command representation.

2.3 Hierarchical Solution Strategy and Thesis Scope

The solution strategy adopted in this thesis introduces a compact body-velocity command interface between learning-based navigation and locomotion-level execution. Instead of directly selecting all eighteen joint commands, the navigation policy selects a low-dimensional

command

$$\boldsymbol{a}_t = \begin{bmatrix} v_t & \omega_t \end{bmatrix}^\top, \tag{2.18}$$

where v_t is the forward body-velocity command and ω_t is the yaw-rate command. This command is then converted into a sequence of joint commands executed over one command cycle, closing the loop as illustrated in Figure 2.1. In the waypoint-guided setting, a rapidly-exploring random tree with rewiring (RRT*) planner provides intermediate waypoints as route guidance.

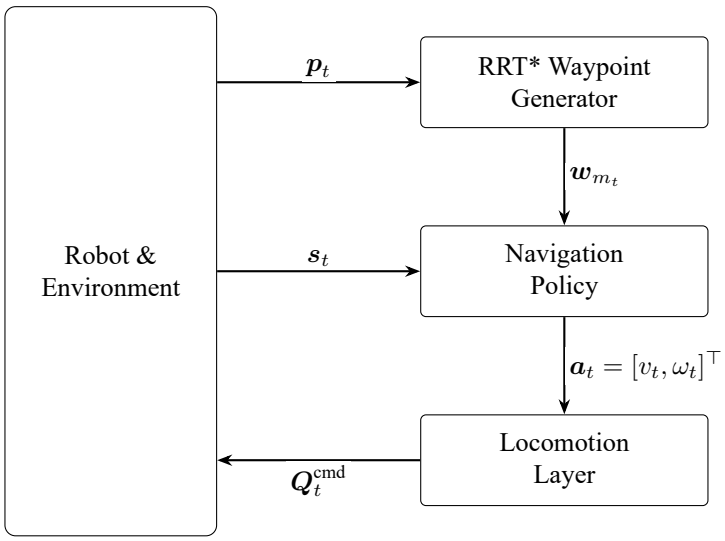


Figure 2.1. Hierarchical control loop. The RRT* planner provides waypoints $\boldsymbol{w}_{m_t}$ from the current robot position $\boldsymbol{p}_t$. The navigation policy maps the full state $\boldsymbol{s}_t$ and waypoint to a body-velocity command $\boldsymbol{a}_t$, which the locomotion layer converts to a joint-command sequence $\boldsymbol{Q}_t^{\text{cmd}}$ applied to the robot.

where $\boldsymbol{Q}_t^{\text{cmd}} = [\boldsymbol{q}_t^{\text{cmd}}(0), \boldsymbol{q}_t^{\text{cmd}}(1), \ldots, \boldsymbol{q}_t^{\text{cmd}}(N_{\text{c}} - 1)]$ is the full joint-command sequence for the command cycle.

This structure replaces the difficult mapping from high-dimensional sensing directly to high-dimensional joint commands with two more manageable mappings. The learning-based navigation component maps local state information and target information to the compact body-velocity command $\boldsymbol{a}_t$. The locomotion component maps $\boldsymbol{a}_t$ and gait phase to joint commands.

In Chapter 4, this command-to-joint mapping is constructed from model-based gait generation and then approximated by a supervised function approximator so that discrete command data can be generalized to continuous velocity and yaw-rate inputs.

The thesis therefore addresses several core questions: how to define the hexapod kinematic model consistently; how to convert compact body-velocity commands into feasible joint trajectories; how to learn obstacle-aware command selection from local visual, depth, proprioceptive, and target-state information; how structured learning-based navigation algorithms improve command selection; and how the resulting system performs in simulation and deployment-oriented real-robot testing.

Chapter 3

Kinematic Modeling of the Hexapod Robot

This chapter presents the kinematic representation used for the hexapod robot throughout this thesis. Robot kinematics is a mature and well-established foundation for analyzing articulated mechanisms [35, 17, 16, 36]; the purpose here is to apply that foundation carefully to the hexapod robot studied in this thesis. The formulation defines the coordinate frames, initial homogeneous transformations, screw axes, Jacobians, and inverse-kinematics conventions needed to describe the motion of each leg segment and end-effector consistently. These ingredients form the geometric foundation for the model-based locomotion layer and for the learning-based navigation methods developed in later chapters.

3.1 Robot Structure and Coordinate Frames

3.1.1 Coordinate System Hierarchy

The robot model is organized through a hierarchy of coordinate frames:

- $\{\boldsymbol{s}\}$: world coordinate system;
- $\{\boldsymbol{b}\}$: body-center coordinate system of the robot;
- $\{\boldsymbol{b}_i\}$: installation base frame of the i-th leg, rigidly attached to the body;

- $\{\boldsymbol{l}_{i1}\}, \{\boldsymbol{l}_{i2}\}, \{\boldsymbol{l}_{i3}\}$: local frames attached to the coxa, femur, and tibia of the i-th leg;

- $\{\boldsymbol{e}_i\}$: end-effector frame at the distal end of the i-th leg.

In the body frame, the positive y-axis points toward the front of the robot, the positive x-axis points toward the robot's right side, and the positive z-axis points upward. Therefore, a positive forward body velocity is represented along $+y_{\boldsymbol{b}}$.

Figure 3.1 illustrates the arrangement of the global and body-related coordinate systems used in the model.

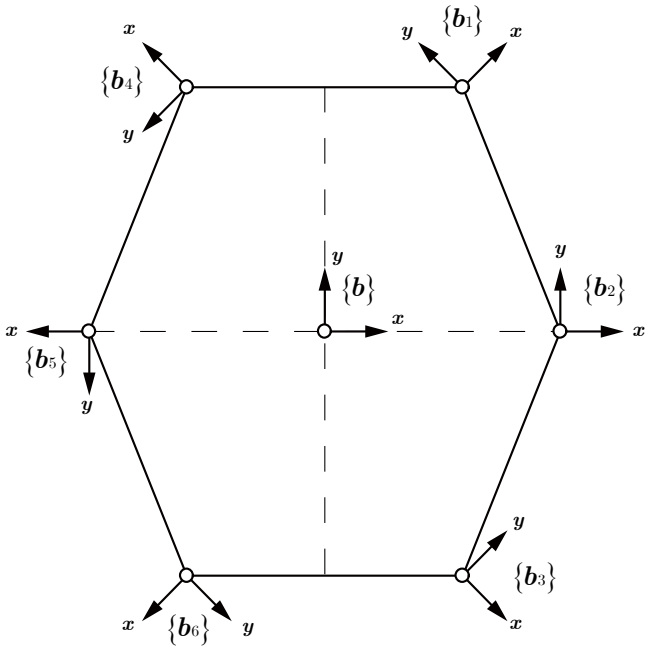


Figure 3.1. Body and leg-base coordinate frames (z-axes out of the page). The global frame $\{\boldsymbol{s}\}$ is the inertial reference, the body-centered frame $\{\boldsymbol{b}\}$ is attached to the robot trunk, and the six leg-base frames $\{\boldsymbol{b}_i\}$ define the mounting positions for the individual leg kinematic chains.

All initial rigid-body transformations are expressed with homogeneous transformation

matrices. In general,

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ \boldsymbol{0} & 1 \end{bmatrix}, \tag{3.1}$$

where $\boldsymbol{R} \in SO(3)$ and $\boldsymbol{p} \in \mathbb{R}^3$ are the rotation and translation blocks of $\boldsymbol{T}$. Throughout this chapter, the notation ${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{T}$ denotes the pose of frame $\{\boldsymbol{B}\}$ expressed in frame $\{\boldsymbol{A}\}$, with the top-left superscript as the reference frame and the bottom-left subscript as the described frame. ${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{R}$ and ${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{p}$ are the corresponding rotation and translation components of ${}^{\boldsymbol{A}}_{\boldsymbol{B}}\boldsymbol{T}$. Unless otherwise noted, angular quantities are expressed in radians and length quantities are expressed in millimeters.

3.1.2 Leg Numbering and Base Installation Frames

The six legs are numbered symmetrically from the body frame perspective:

- Legs 1–3 denote the front-right, right-middle, and rear-right legs;
- Legs 4–6 denote the front-left, left-middle, and rear-left legs.

This convention provides a compact way to index all geometric parameters and control variables consistently. The rotation matrix and translation vector of each body-to-leg-base transform ${}^{\boldsymbol{b}}_{\boldsymbol{b}_i}\boldsymbol{T}$ are:

- Leg 1 (front-right): ${}^{\boldsymbol{b}}_{\boldsymbol{b}_1}\boldsymbol{R} = \boldsymbol{R}_z(\gamma), \quad {}^{\boldsymbol{b}}_{\boldsymbol{b}_1}\boldsymbol{p} = \begin{bmatrix} a & b & h \end{bmatrix}^\top$
- Leg 2 (middle-right): ${}^{\boldsymbol{b}}_{\boldsymbol{b}_2}\boldsymbol{R} = \boldsymbol{I}, \quad {}^{\boldsymbol{b}}_{\boldsymbol{b}_2}\boldsymbol{p} = \begin{bmatrix} A & 0 & h \end{bmatrix}^\top$
- Leg 3 (rear-right): ${}^{\boldsymbol{b}}_{\boldsymbol{b}_3}\boldsymbol{R} = \boldsymbol{R}_z(-\gamma), \quad {}^{\boldsymbol{b}}_{\boldsymbol{b}_3}\boldsymbol{p} = \begin{bmatrix} a & -b & h \end{bmatrix}^\top$

- Leg 4 (front-left): ${}^{b}_{b_4}\boldsymbol{R} = \boldsymbol{R}_z(\pi - \gamma), \quad {}^{b}_{b_4}\boldsymbol{p} = \begin{bmatrix} -a & b & h \end{bmatrix}^\top$
- Leg 5 (middle-left): ${}^{b}_{b_5}\boldsymbol{R} = \boldsymbol{R}_z(\pi), \quad {}^{b}_{b_5}\boldsymbol{p} = \begin{bmatrix} -A & 0 & h \end{bmatrix}^\top$
- Leg 6 (rear-left): ${}^{b}_{b_6}\boldsymbol{R} = \boldsymbol{R}_z(\pi + \gamma), \quad {}^{b}_{b_6}\boldsymbol{p} = \begin{bmatrix} -a & -b & h \end{bmatrix}^\top$

The rotation matrix about the z-axis is

$$\boldsymbol{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The geometric parameters used for the six leg-base installation positions are summarized in Table 3.1.

Table 3.1. Geometric parameters defining the six leg-base installation positions

γ (rad)	a (mm)	b (mm)	A (mm)	h (mm)
$\pi/4$	79.5	113.6	121.0	27.5

3.1.3 Leg Segment Frames in the Initial Configuration

The local coordinate systems assigned to the leg segments are shown in Figure 3.2. In this arrangement, the rotation axis of each joint is aligned with the local z-axis, the extension direction of each link is aligned with the local x-axis, and the y-axis is determined by the right-hand rule.

The initial relative transformations between consecutive frames are defined in the home, or zero, configuration. The four transforms — from leg base to coxa, coxa to femur, femur to tibia, and tibia to end-effector — are given by their rotation matrix and translation vector:

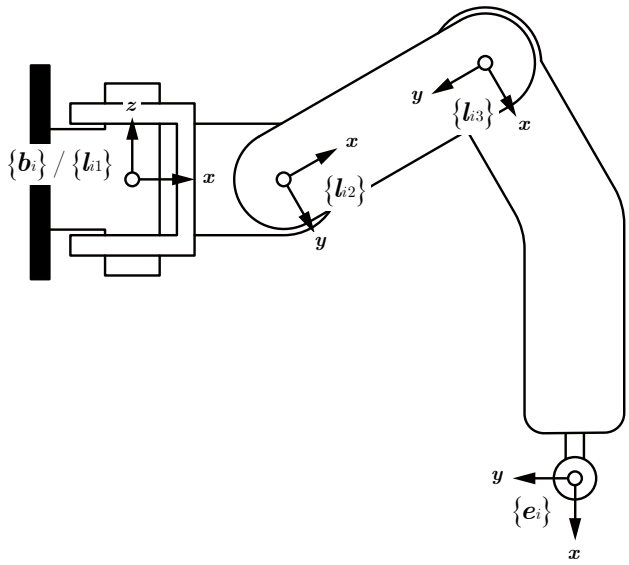


Figure 3.2. Coordinate frames along a representative leg (y-axes of $\{\boldsymbol{b}_i\}$, $\{\boldsymbol{l}_{i1}\}$ and z-axes of remaining frames point into the page). From the leg-base frame $\{\boldsymbol{b}_i\}$, successive frames $\{\boldsymbol{l}_{i1}\}$, $\{\boldsymbol{l}_{i2}\}$, and $\{\boldsymbol{l}_{i3}\}$ are attached to the coxa, femur, and tibia, with $\{\boldsymbol{e}_i\}$ at the foot-tip end-effector.

- Leg base to coxa: ${}^{\boldsymbol{b}_i}_{\boldsymbol{l}_{i1}}\boldsymbol{R}^0 = \boldsymbol{I}, \quad {}^{\boldsymbol{b}_i}_{\boldsymbol{l}_{i1}}\boldsymbol{p}^0 = \boldsymbol{0}$
- Coxa to femur: ${}^{\boldsymbol{l}_{i1}}_{\boldsymbol{l}_{i2}}\boldsymbol{R}^0 = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ 0 & 0 & 1 \\ \sin\theta_1 & -\cos\theta_1 & 0 \end{bmatrix}, \quad {}^{\boldsymbol{l}_{i1}}_{\boldsymbol{l}_{i2}}\boldsymbol{p}^0 = \begin{bmatrix} 48.0 \\ 0 \\ 0 \end{bmatrix}$
- Femur to tibia: ${}^{\boldsymbol{l}_{i2}}_{\boldsymbol{l}_{i3}}\boldsymbol{R}^0 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad {}^{\boldsymbol{l}_{i2}}_{\boldsymbol{l}_{i3}}\boldsymbol{p}^0 = \begin{bmatrix} 90.2 \\ 0 \\ 0 \end{bmatrix}$
- Tibia to end-effector: ${}^{\boldsymbol{l}_{i3}}_{\boldsymbol{e}_i}\boldsymbol{R}^0 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad {}^{\boldsymbol{l}_{i3}}_{\boldsymbol{e}_i}\boldsymbol{p}^0 = \begin{bmatrix} 146.7 \\ 53.0 \\ 0 \end{bmatrix}$

In the initial geometric setup, the parameter θ_1 is fixed at $\pi/6$.

3.2 Screw Axis Representation

To describe the joint motions in a body-consistent form, the unit screw axis

$$\boldsymbol{\xi}_0 = [0, 0, 1, 0, 0, 0]^\top$$

is transported into the body frame through adjoint transformations. For the three joints of leg i, the resulting screw axes are

$$\mathcal{B}_{i1} = \mathrm{Ad}_{\left[{}^{\boldsymbol{b}}_{\boldsymbol{b}_i}\boldsymbol{T}^0 {}^{\boldsymbol{b}_i}_{\boldsymbol{l}_{i1}}\boldsymbol{T}^0\right]}\boldsymbol{\xi}_0, \tag{3.2}$$

$$\mathcal{B}_{i2} = \mathrm{Ad}_{\left[{}^{\boldsymbol{b}}_{\boldsymbol{b}_i}\boldsymbol{T}^0 {}^{\boldsymbol{b}_i}_{\boldsymbol{l}_{i1}}\boldsymbol{T}^0 {}^{\boldsymbol{l}_{i1}}_{\boldsymbol{l}_{i2}}\boldsymbol{T}^0\right]}\boldsymbol{\xi}_0, \tag{3.3}$$

$$\mathcal{B}_{i3} = \mathrm{Ad}_{\left[{}^{\boldsymbol{b}}_{\boldsymbol{b}_i}\boldsymbol{T}^0 {}^{\boldsymbol{b}_i}_{\boldsymbol{l}_{i1}}\boldsymbol{T}^0 {}^{\boldsymbol{l}_{i1}}_{\boldsymbol{l}_{i2}}\boldsymbol{T}^0 {}^{\boldsymbol{l}_{i2}}_{\boldsymbol{l}_{i3}}\boldsymbol{T}^0\right]}\boldsymbol{\xi}_0. \tag{3.4}$$

Given a rigid transformation

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{p} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \in SE(3), \qquad [\boldsymbol{p}]_\times = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix},$$

the corresponding adjoint matrix is defined by

$$\mathrm{Ad}_{[\boldsymbol{T}]} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{0} \\ [\boldsymbol{p}]_\times \boldsymbol{R} & \boldsymbol{R} \end{bmatrix}. \tag{3.5}$$

For later computation, each screw axis is written in six-dimensional vector form as

$$\mathcal{B}_{ij} = \begin{bmatrix} \boldsymbol{\omega}_{ij} \\ \boldsymbol{v}_{ij} \end{bmatrix} \in \mathbb{R}^6, \qquad \boldsymbol{\omega}_{ij}, \boldsymbol{v}_{ij} \in \mathbb{R}^3.$$

Its matrix representation is

$$[\hat{\mathcal{B}}_{ij}] = \begin{bmatrix} [\boldsymbol{\omega}_{ij}]_\times & \boldsymbol{v}_{ij} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix}, \qquad [\boldsymbol{\omega}_{ij}]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{3.6}$$

3.3 Forward Kinematics

Let $\boldsymbol{q}_i = (q_{i1}, q_{i2}, q_{i3})^\top$ denote the joint-angle vector of leg i, where q_{ij} is the commanded angle of joint j in that leg. Using the product-of-exponentials formulation, the pose of each leg segment relative to the body frame can be written as follows. Here $\boldsymbol{T}(\boldsymbol{q}_i)$ denotes the joint-dependent pose map, while $\boldsymbol{T}^0$ denotes the corresponding constant transform in the home, or zero, configuration.

These segment poses and the end-effector pose are written together as

$$\underset{\boldsymbol{l}_{i1}}{\boldsymbol{b}}\boldsymbol{T}(\boldsymbol{q}_i) = \underset{\boldsymbol{l}_{i1}}{\boldsymbol{b}}\boldsymbol{T}(q_{i1}) = e^{[\hat{\mathcal{B}}_{i1}]q_{i1}} \underset{\boldsymbol{l}_{i1}}{\boldsymbol{b}}\boldsymbol{T}^0, \tag{3.7}$$

$$\underset{\boldsymbol{l}_{i2}}{\boldsymbol{b}}\boldsymbol{T}(\boldsymbol{q}_i) = \underset{\boldsymbol{l}_{i2}}{\boldsymbol{b}}\boldsymbol{T}(q_{i1}, q_{i2}) = e^{[\hat{\mathcal{B}}_{i1}]q_{i1}} e^{[\hat{\mathcal{B}}_{i2}]q_{i2}} \underset{\boldsymbol{l}_{i2}}{\boldsymbol{b}}\boldsymbol{T}^0, \tag{3.8}$$

$$\underset{\boldsymbol{l}_{i3}}{\boldsymbol{b}}\boldsymbol{T}(\boldsymbol{q}_i) = \underset{\boldsymbol{l}_{i3}}{\boldsymbol{b}}\boldsymbol{T}(q_{i1}, q_{i2}, q_{i3}) = e^{[\hat{\mathcal{B}}_{i1}]q_{i1}} e^{[\hat{\mathcal{B}}_{i2}]q_{i2}} e^{[\hat{\mathcal{B}}_{i3}]q_{i3}} \underset{\boldsymbol{l}_{i3}}{\boldsymbol{b}}\boldsymbol{T}^0, \tag{3.9}$$

$$\underset{\boldsymbol{e}_i}{\boldsymbol{b}}\boldsymbol{T}(\boldsymbol{q}_i) = \underset{\boldsymbol{e}_i}{\boldsymbol{b}}\boldsymbol{T}(q_{i1}, q_{i2}, q_{i3}) = e^{[\hat{\mathcal{B}}_{i1}]q_{i1}} e^{[\hat{\mathcal{B}}_{i2}]q_{i2}} e^{[\hat{\mathcal{B}}_{i3}]q_{i3}} \underset{\boldsymbol{e}_i}{\boldsymbol{b}}\boldsymbol{T}^0. \tag{3.10}$$

The coxa, femur, and tibia poses therefore accumulate the first one, two, and three joint exponentials, while the end-effector pose shares the same full chain and differs only in the home transformation, as shown in Eq. (3.10).

This forward model provides the pose of every leg segment and the foot frame directly from the 18 joint variables of the hexapod.

3.4 Jacobian and Inverse Kinematics

The forward-kinematic map in the previous section provides the end-effector pose for a given joint vector. For gait generation and motion execution, the inverse problem is also needed: given a desired body-frame foot-tip position, compute the corresponding joint angles. This section defines the Jacobian and inverse-kinematics procedure used throughout the remainder of the thesis.

3.4.1 Space Jacobian Expressed in the Body Frame

For leg i, the end-effector pose has the product-of-exponentials form in Eq. (3.10), repeated here for clarity:

$$\underset{\boldsymbol{e}_i}{\overset{\boldsymbol{b}}{\boldsymbol{T}}}(\boldsymbol{q}_i) = e^{[\hat{\mathcal{B}}_{i1}]q_{i1}} e^{[\hat{\mathcal{B}}_{i2}]q_{i2}} e^{[\hat{\mathcal{B}}_{i3}]q_{i3}} \underset{\boldsymbol{e}_i}{\overset{\boldsymbol{b}}{\boldsymbol{T}}}{}^{0}. \tag{3.11}$$

Because the screw axes $\mathcal{B}_{ij}$ are expressed with respect to the robot body frame $\{\boldsymbol{b}\}$ and are left-multiplied in the product-of-exponentials expression, the corresponding Jacobian is the space Jacobian with $\{\boldsymbol{b}\}$ serving as the space frame. Following the standard Modern Robotics construction [17], the columns are obtained by transporting each screw axis through the preceding

joint exponentials:

$$\boldsymbol{J}_{s,i}(\boldsymbol{q}_i) = \begin{bmatrix} \mathcal{J}_{i1} & \mathcal{J}_{i2} & \mathcal{J}_{i3} \end{bmatrix} \in \mathbb{R}^{6\times 3}, \tag{3.12}$$

where

$$\mathcal{J}_{i1} = \mathcal{B}_{i1}, \qquad \mathcal{J}_{i2} = \mathrm{Ad}_{\left[e^{[\hat{\mathcal{B}}_{i1}]q_{i1}}\right]}\mathcal{B}_{i2}, \qquad \mathcal{J}_{i3} = \mathrm{Ad}_{\left[e^{[\hat{\mathcal{B}}_{i1}]q_{i1}}e^{[\hat{\mathcal{B}}_{i2}]q_{i2}}\right]}\mathcal{B}_{i3}. \tag{3.13}$$

Equivalently, the j-th Jacobian column can be written as

$$\mathcal{J}_{ij}(\boldsymbol{q}_i) = \begin{bmatrix} {}^{s}\boldsymbol{\omega}_{ij}(\boldsymbol{q}_i) \\ {}^{s}\boldsymbol{v}_{ij}(\boldsymbol{q}_i) \end{bmatrix},$$

where the superscript s indicates that the twist components are expressed in the space frame used for the product of exponentials, which in this thesis is the robot body frame $\{\boldsymbol{b}\}$.

3.4.2 Position Jacobian for the Foot Tip

The inverse-kinematics problem used in this thesis is position based: the desired command specifies the Cartesian foot-tip position rather than the full end-effector orientation. Let

$$\boldsymbol{p}_i(\boldsymbol{q}_i) \in \mathbb{R}^3$$

denote the translation component of ${}^{\boldsymbol{b}}_{\boldsymbol{e}_i}\boldsymbol{T}(\boldsymbol{q}_i)$. The corresponding position Jacobian is

$$\boldsymbol{J}_{p,i}(\boldsymbol{q}_i) = \frac{\partial \boldsymbol{p}_i(\boldsymbol{q}_i)}{\partial \boldsymbol{q}_i} \in \mathbb{R}^{3\times 3}. \tag{3.14}$$

It can be computed directly from the space Jacobian columns. For the j-th column,

$$\boldsymbol{J}_{p,i}^{(:,j)}(\boldsymbol{q}_i) = {}^{s}\boldsymbol{\omega}_{ij}(\boldsymbol{q}_i) \times \boldsymbol{p}_i(\boldsymbol{q}_i) + {}^{s}\boldsymbol{v}_{ij}(\boldsymbol{q}_i). \tag{3.15}$$

This relation follows from the spatial velocity of a point under a twist:

$$\dot{\boldsymbol{p}} = \boldsymbol{\omega} \times \boldsymbol{p} + \boldsymbol{v}.$$

Thus the same sequence of matrix exponentials and adjoint transformations used to compute the Modern Robotics space Jacobian in Eqs. (3.12)–(3.13) also provides the position Jacobian in Eq. (3.15) needed for iterative inverse kinematics.

3.4.3 Damped Inverse Kinematics

For a desired body-frame foot-tip position $\boldsymbol{p}_{\text{des},i} \in \mathbb{R}^3$, the single-leg inverse-kinematics problem is to find $\boldsymbol{q}_i \in \mathbb{R}^3$ such that

$$\boldsymbol{p}_i(\boldsymbol{q}_i) = \boldsymbol{p}_{\text{des},i}. \tag{3.16}$$

The desired position must be expressed in the same length unit as the forward-kinematic model before the error is evaluated. In the present implementation, the geometric model in this chapter uses millimeters; therefore, any gait trajectory generated in meters is converted to millimeters before applying inverse kinematics.

Define the position error

$$\boldsymbol{e}_i = \boldsymbol{p}_{\text{des},i} - \boldsymbol{p}_i(\boldsymbol{q}_i). \tag{3.17}$$

The damped least-squares update is

$$\Delta \boldsymbol{q}_i = \boldsymbol{J}_{p,i}^{\top} \left(\boldsymbol{J}_{p,i} \boldsymbol{J}_{p,i}^{\top} + \lambda^2 \boldsymbol{I} \right)^{-1} \boldsymbol{e}_i, \tag{3.18}$$

followed by the iterative correction

$$\boldsymbol{q}_i^{(m+1)} = \boldsymbol{q}_i^{(m)} + \eta \, \Delta \boldsymbol{q}_i^{(m)}, \tag{3.19}$$

where $\lambda > 0$ is the damping coefficient and $\eta \in (0, 1]$ is the step size. Damping, step-size scaling, and increment clamping improve numerical stability and reduce the chance of large discontinuous updates near singular or poorly conditioned configurations in the update defined by Eq. (3.18).

The one-leg inverse-kinematics routine is summarized in Algorithm 1.

3.5 Implementation and Visualization

The reference implementation uses the following function to construct the robot model from a given joint configuration:

```
build_hexapod(q, T, B, meshes)
```

The inputs are:

- `q`: $\boldsymbol{q} \in \mathbb{R}^{6\times 3}$, the joint-angle array for the six legs, with three components per leg.
- `T`: $\{\boldsymbol{T}^0\}$, the set of home-configuration homogeneous transformations.
- `B`: $\{\mathcal{B}_{ij}\}$, the set of screw axes for the three joints of each leg.

Algorithm 1. Damped inverse kinematics for one leg

Require: Desired tip position $\boldsymbol{p}_{\text{des},i}$, initial guess $\boldsymbol{q}_i^{(0)}$, damping λ, step size η, tolerance ε, maximum iteration count M

Ensure: Joint solution $\boldsymbol{q}_i$

$m \leftarrow 0$
$\boldsymbol{q}_i \leftarrow \boldsymbol{q}_i^{(0)}$
while $m < M$ **do**
    Compute $\boldsymbol{p}_i(\boldsymbol{q}_i)$
    $\boldsymbol{e}_i \leftarrow \boldsymbol{p}_{\text{des},i} - \boldsymbol{p}_i(\boldsymbol{q}_i)$
    if $\|\boldsymbol{e}_i\| < \varepsilon$ **then**
        return $\boldsymbol{q}_i$
    end if
    Compute $\boldsymbol{J}_{s,i}(\boldsymbol{q}_i)$ using adjoint-transformed screw axes
    Compute $\boldsymbol{J}_{p,i}(\boldsymbol{q}_i)$ from the point-velocity relation
    $\Delta\boldsymbol{q}_i \leftarrow \boldsymbol{J}_{p,i}^{\top}\left(\boldsymbol{J}_{p,i}\boldsymbol{J}_{p,i}^{\top} + \lambda^2\boldsymbol{I}\right)^{-1}\boldsymbol{e}_i$
    Apply step-size scaling, and if necessary clamp $\Delta\boldsymbol{q}_i$
    $\boldsymbol{q}_i \leftarrow \boldsymbol{q}_i + \eta\,\Delta\boldsymbol{q}_i$
    $m \leftarrow m + 1$
end while
return $\boldsymbol{q}_i$

- `meshes`: the geometric mesh models of the robot body and links used for visualization.

The implementation computes the required transformations from q, applies them to the corresponding mesh objects, and then assembles the resulting geometry into a complete robot model for visualization. A representative result is shown in Figure 3.3. In the end-effector frame, the red, green, and blue axes denote the x-, y-, and z-directions, respectively.

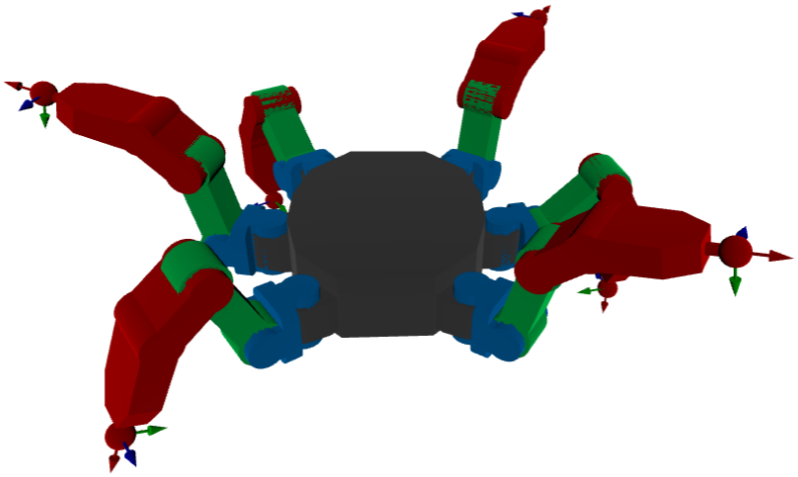


Figure 3.3. The visualization of the hexapod model generated from the kinematic description.

Chapter 4

Kinematics-Based Gait Generation

This chapter develops the model-based locomotion layer used in this thesis. As introduced in Chapter 2, the learning-based navigation policy selects a compact body command, while the locomotion layer converts that command into executable joint trajectories for the hexapod robot. The command considered here is represented by the desired forward velocity v and yaw rate ω.

The formulation builds on established ideas in legged gait generation [1] and inverse-kinematics-based robot motion execution [37, 17]. Given a desired body motion and a set of foot-tip trajectories, inverse kinematics can compute the corresponding joint commands for one gait cycle. Performing this computation online for many possible velocity and yaw-rate commands, however, can be inefficient, while a purely precomputed command table only represents a finite set of discrete commands.

The contribution of this chapter is to construct a practical command-to-joint mapping for the hexapod platform. Model-based gait generation is first used to produce joint-command data at sampled command values. A supervised approximation model is then trained to generalize this discrete command data into a continuous mapping from (v, ω) to coordinated joint-command

sequences. This keeps the locomotion layer interpretable while making it efficient enough to be used by the learning-based navigation methods developed later.

4.1 Method Overview

The key idea of this chapter is to provide the model-based locomotion side of the compact body-command interface. Instead of generating gait motion directly from complex observations through a single mapping, the locomotion layer receives a compact body command and converts it into joint-space motion through a structured sequence. Figure 4.1 summarizes the overall proposed command-generation architecture.

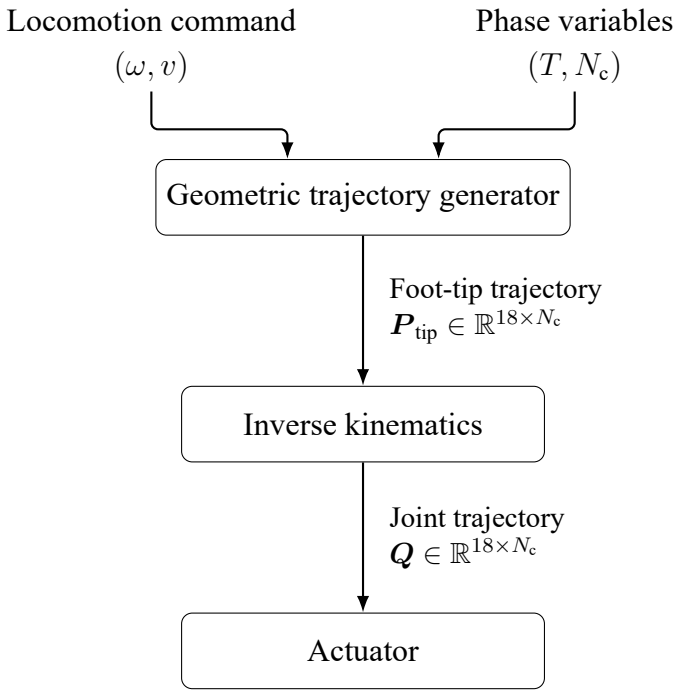


Figure 4.1. Model-based command-generation pipeline. Motion command (ω, v) and period parameters (T, N_c) feed a geometric trajectory generator, whose foot-tip output $\boldsymbol{P}_{tip} \in \mathbb{R}^{18\times N_c}$ is converted by inverse kinematics to the joint-command trajectory $\boldsymbol{Q} \in \mathbb{R}^{18\times N_c}$ for actuator execution.

Here, (ω, v) describes the average planar motion over one gait cycle, T denotes the dura-

tion of that cycle, and N_{c} denotes the number of discrete time steps used to represent that cycle. These two input groups play different roles. The pair (ω, v) specifies the desired body-level turning and forward behavior under the unicycle approximation, whereas (T, N_{c}) determines the temporal resolution used to sample one gait period.

The first computational block in Figure 4.1 is the geometric foot-tip trajectory generator. Its role is to convert the cycle-level command and phase definition into time-indexed Cartesian foot-tip trajectories for all six legs. This mapping may be written as

$$\mathcal{G} : \big\{(\omega, v), (T, N_{\mathrm{c}})\big\} \mapsto \boldsymbol{P}_{\mathrm{tip}}, \tag{4.1}$$

where the resulting trajectory is

$$\boldsymbol{P}_{\mathrm{tip}} = \Big[\boldsymbol{p}_{\mathrm{tip}}(0) \quad \boldsymbol{p}_{\mathrm{tip}}(1) \quad \cdots \quad \boldsymbol{p}_{\mathrm{tip}}(N_{\mathrm{c}} - 1)\Big] \in \mathbb{R}^{18 \times N_{\mathrm{c}}}, \tag{4.2}$$

where each column $\boldsymbol{p}_{\mathrm{tip}}(k) \in \mathbb{R}^{18}$ contains the desired 3D foot-tip positions of all six legs at the k-th time step. Throughout this chapter, these foot-tip positions are expressed in the robot body-centered frame $\{\boldsymbol{b}\}$ unless stated otherwise. Equivalently, $\boldsymbol{P}_{\mathrm{tip}}$ in Eq. (4.2) may be viewed as a $6 \times 3 \times N_{\mathrm{c}}$ trajectory tensor with respect to leg index, Cartesian coordinate, and time step.

The second block is inverse kinematics (IK), which transforms the Cartesian foot-tip trajectory into the corresponding joint-command sequence. This second mapping is written as

$$\mathcal{I} : \boldsymbol{P}_{\mathrm{tip}} \mapsto \boldsymbol{Q}, \tag{4.3}$$

with output

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}(0) & \boldsymbol{q}(1) & \cdots & \boldsymbol{q}(N_{\mathrm{c}}-1) \end{bmatrix} \in \mathbb{R}^{18\times N_{\mathrm{c}}}, \tag{4.4}$$

where each $\boldsymbol{q}(k) \in \mathbb{R}^{18}$ denotes the full 18-dimensional joint command at the k-th time step of the gait cycle.

Although the mappings $\mathcal{G}$ and $\mathcal{I}$ in Eqs. (4.1) and (4.3) are well-defined and physically interpretable, generating commands through them online is computationally expensive. Each query requires trajectory construction over the gait cycle, followed by repeated inverse-kinematics updates across all legs and all time steps. In other words, the model-based command law is available only through an implicit online generation process rather than through a lightweight closed-form expression. This computational burden makes direct online generation undesirable for real-time command output. The role of learning-based approximation in this chapter is therefore to construct a fast surrogate model, trained in a supervised learning manner from the model-based library, that approximates the mapping from body-level commands and gait phase to joint-space commands.

4.2 Geometric Foot-Tip Trajectory Generation

4.2.1 Tripod Gait Representation

Before introducing phase variables, it is useful to state the gait pattern assumed throughout this chapter. A gait specifies the temporal coordination of stance and swing among the six legs. In this work, the representative gait is the tripod gait, in which the six legs are partitioned

into two alternating groups,

$$\mathcal{T}_1 = \{1, 3, 5\}, \qquad \mathcal{T}_2 = \{2, 4, 6\}.$$

When one tripod group is in stance, the other is in swing. This alternating structure is widely used for hexapod locomotion because it provides a simple and effective support pattern while remaining easy to parameterize by the body-level command pair (ω, v) [2].

In the convention adopted in this work, the cycle origin is chosen so that $\mathcal{T}_1$ starts from mid-stance, while $\mathcal{T}_2$ starts from mid-swing. This convention is convenient because the anchor configuration is fixed and independent of the commanded motion. Therefore, for any commanded linear velocity v and angular velocity ω, the gait cycle starts from the same balanced reference configuration, with each leg initialized at either mid-stance or mid-swing. Figure 4.2 recalls the leg-indexing convention used throughout the chapter, and Figure 4.3 shows the corresponding tripod-gait timing pattern over repeated gait cycles.

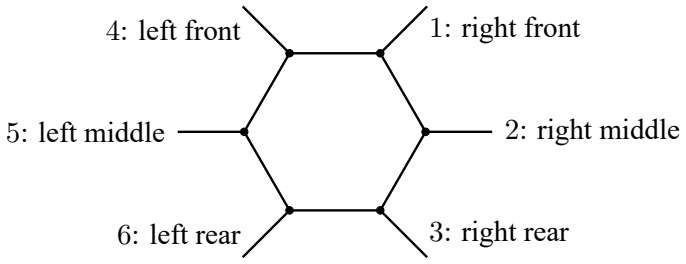


Figure 4.2. Leg-indexing convention (top view). Right-side legs are numbered $1, 2, 3$ front-to-rear; left-side legs are numbered $4, 5, 6$ front-to-rear.

4.2.2 Body-Frame Foot-Tip Trajectory

This subsection develops the geometric shape of the foot-tip trajectory in the robot body frame. The hexapod is treated at the gait-cycle level as a planar unicycle: the body command

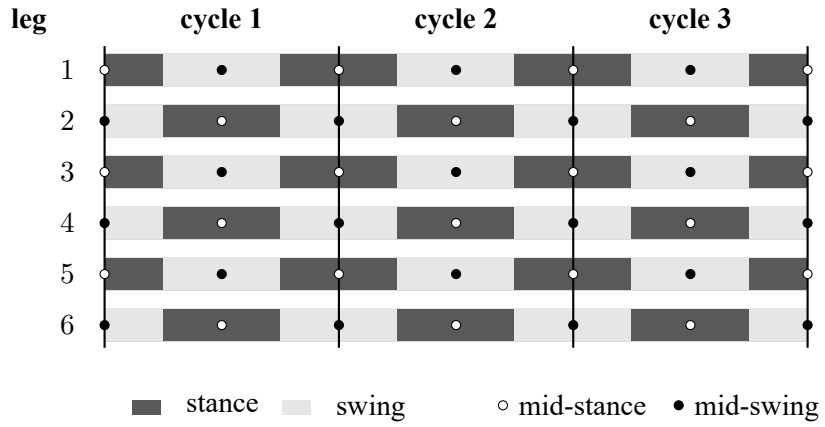


Figure 4.3. Tripod-gait schedule over three repeated cycles. Legs $\mathcal{T}_1 = \{1, 3, 5\}$ and $\mathcal{T}_2 = \{2, 4, 6\}$ are offset by half a cycle, with the origin $g = 0$ at mid-stance of $\mathcal{T}_1$ and mid-swing of $\mathcal{T}_2$.

(ω, v) specifies the desired average yaw rate and forward speed over one cycle. From these two quantities the instantaneous center of rotation and the resulting foot-tip path shape are determined. The trajectory is first described in the world frame, then transferred to the body frame via a relative-motion argument. The stance path follows directly from this transfer; the swing path is the planar recovery of the stance path with an added vertical lifting arc. The normalized progress variables and smooth polynomial profiles that parameterize this motion along these paths are introduced in Section 4.2.3.

Unless otherwise noted, all lengths are in metres and angular quantities in radians. The positive v direction is the positive y-axis of the body frame $\{\boldsymbol{b}\}$, and positive yaw follows the right-hand rule about $+z_{\boldsymbol{b}}$.

Relative Motion and Body-Frame Equivalence

Because the foot-tip trajectory must be expressed in the body-centered frame $\{\boldsymbol{b}\}$, it is necessary to understand how a world-fixed stance contact appears when viewed from the moving body. Let $\{\boldsymbol{s}\}$ denote the world frame and ${}^{\boldsymbol{s}}_{\boldsymbol{b}}\boldsymbol{R}$ the rotation from $\{\boldsymbol{b}\}$ to $\{\boldsymbol{s}\}$. Differentiating the

position relation ${}^{s}\boldsymbol{p}_{\text{tip},i} = {}^{s}_{b}\boldsymbol{p} + {}^{s}_{b}\boldsymbol{R}\,{}^{b}\boldsymbol{p}_{\text{tip},i}$ and solving for the body-frame foot-tip velocity gives

$$ {}^{b}\boldsymbol{v}_{\text{tip},i} = {}^{b}_{s}\boldsymbol{R}({}^{s}\boldsymbol{v}_{\text{tip},i} - {}^{s}\boldsymbol{v}_{b}) - {}^{b}\boldsymbol{\omega}_{b} \times {}^{b}\boldsymbol{p}_{\text{tip},i}. \tag{4.5} $$

During stance the foot is fixed to the ground, so ${}^{s}\boldsymbol{v}_{\text{tip},i} = \mathbf{0}$ and Eq. (4.5) reduces to

$$ {}^{b}\boldsymbol{v}_{\text{tip},i} = -{}^{b}_{s}\boldsymbol{R}\,{}^{s}\boldsymbol{v}_{b} - {}^{b}\boldsymbol{\omega}_{b} \times {}^{b}\boldsymbol{p}_{\text{tip},i}. $$

The body-frame velocity of a stance contact is therefore the negative of the body's own velocity. Integrating this relation shows that a world-fixed contact traces in $\{\boldsymbol{b}\}$ the same geometric path as the body, but in the opposite direction: a straight body translation produces a straight contact path, and a circular body arc produces a circular contact arc. The planar projection of these paths is sufficient to determine the trajectory geometry; the vertical foot position is treated separately through the swing lift profile.

Straight-Motion Case $(\omega = 0)$

When $\omega = 0$ the body advances in a straight line. The body translates by

$$ \Delta s = \frac{vT}{2} \tag{4.6} $$

over one half-cycle. By the relative-motion result, the world-fixed stance contact traces in $\{\boldsymbol{b}\}$ a straight segment of the same length, directed opposite to the body translation and centered at the mid-stance anchor ${}^{b}\boldsymbol{p}_{\text{a},i}$. Using a normalized planar progress variable $\mu_{\text{st},i} \in [-1, 1]$ decreasing

from 1 to -1 during stance, the body-frame stance trajectory is

$$\boldsymbol{p}_{\text{st},i}(\mu_{\text{st},i}) = {}^{\boldsymbol{b}}\boldsymbol{p}_{\text{a},i} + \begin{bmatrix} 0 \\ \dfrac{\Delta s}{2}\,\mu_{\text{st},i} \\ 0 \end{bmatrix}, \qquad \omega = 0. \tag{4.7}$$

The swing phase recovers the foot from lift-off to touch-down along the same straight path in the opposite planar direction, adding a smooth vertical arc. Using a swing progress variable $\mu_{\text{sw},i} \in [-1, 1]$ increasing from -1 to 1 and a normalized vertical lift $\zeta_{\text{sw},i} \in [0, 1]$, the swing trajectory is

$$\boldsymbol{p}_{\text{sw},i}(\mu_{\text{sw},i}, \zeta_{\text{sw},i}) = {}^{\boldsymbol{b}}\boldsymbol{p}_{\text{a},i} + \begin{bmatrix} 0 \\ \dfrac{\Delta s}{2}\,\mu_{\text{sw},i} \\ H\zeta_{\text{sw},i} \end{bmatrix}, \qquad \omega = 0, \tag{4.8}$$

where H is the maximum foot clearance height. Figure 4.4 illustrates this geometry in both the world frame and the body frame.

Turning-Motion Case $(\omega \neq 0)$

When $\omega \neq 0$ the body follows a circular arc. The turning radius, yaw increment per half-cycle, and instantaneous center expressed in $\{\boldsymbol{b}\}$ are

$$R = \frac{v}{\omega}, \qquad \Delta\theta = \frac{\omega T}{2}, \qquad \boldsymbol{c} = \begin{bmatrix} -R \\ 0 \\ 0 \end{bmatrix}. \tag{4.9}$$

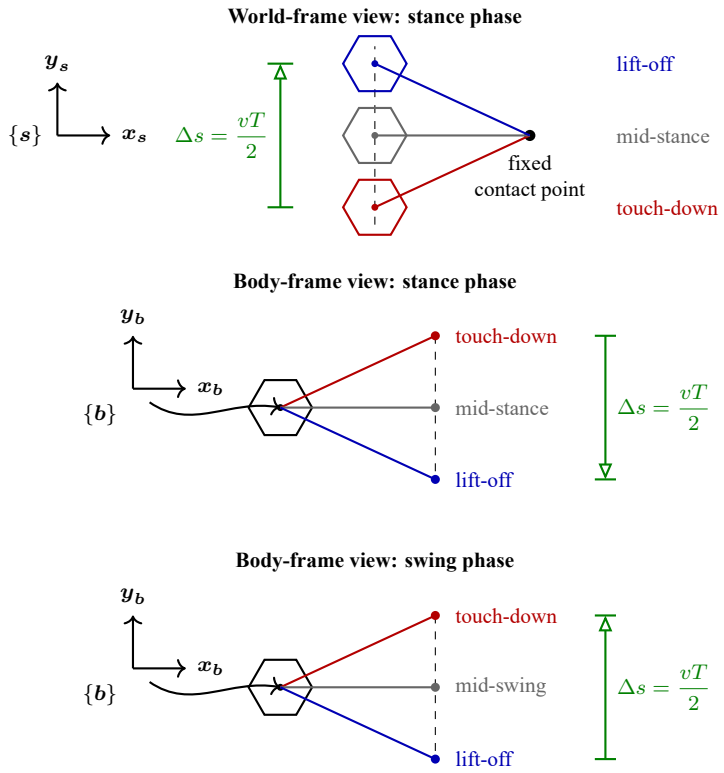


Figure 4.4. Straight-motion foot-tip trajectory in body frame $\{\boldsymbol{b}\}$. The stance tip, fixed in world frame $\{\boldsymbol{s}\}$, traces a reversed straight path centered at mid-stance; the swing phase returns the tip along the same line.

Each leg's mid-stance anchor defines a relative vector

$$\boldsymbol{r}_i = {}^{\boldsymbol{b}}\boldsymbol{p}_{\mathrm{a},i} - \boldsymbol{c}, \tag{4.10}$$

so leg i traces an arc of radius $\|\boldsymbol{r}_i\|$ about $\boldsymbol{c}$, sweeping in the direction opposite to the body yaw.

Using $\mu_{\mathrm{st},i}$ decreasing from 1 to -1, the turning-motion stance trajectory is

$$\boldsymbol{p}_{\mathrm{st},i}(\mu_{\mathrm{st},i}) = \boldsymbol{c} + \boldsymbol{R}_z\left(\frac{\Delta\theta}{2}\mu_{\mathrm{st},i}\right)\boldsymbol{r}_i, \qquad \omega \neq 0, \tag{4.11}$$

and the swing recovery with vertical arc is

$$\boldsymbol{p}_{\mathrm{sw},i}(\mu_{\mathrm{sw},i},\zeta_{\mathrm{sw},i}) = \boldsymbol{c} + \boldsymbol{R}_z\left(\frac{\Delta\theta}{2}\mu_{\mathrm{sw},i}\right)\boldsymbol{r}_i + \begin{bmatrix} 0 \\ 0 \\ H\zeta_{\mathrm{sw},i} \end{bmatrix}, \qquad \omega \neq 0. \tag{4.12}$$

The straight case is the limiting case in which $R \to \infty$ and the arc degenerates to the line segments of Eqs. (4.7)–(4.8). Figure 4.5 illustrates the turning geometry in both frames.

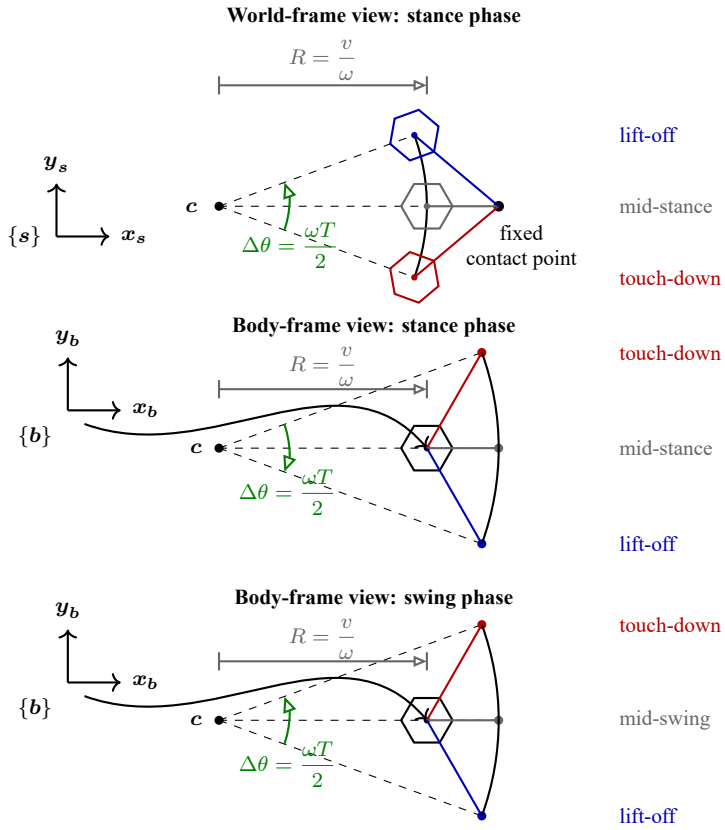


Figure 4.5. Turning-motion foot-tip trajectory in body frame $\{\boldsymbol{b}\}$. The stance tip, fixed in world frame $\{\boldsymbol{s}\}$, traces a circular arc about the instantaneous center $\boldsymbol{c}$ with reversed angular direction; the swing phase returns along the same arc.

4.2.3 Phase Parameterization and Smooth Profiles

The trajectory equations above express foot-tip positions in terms of two normalized progress variables: $\mu \in [-1, 1]$ for planar motion and $\zeta \in [0, 1]$ for vertical lift. This subsection specifies how these variables are assigned as functions of time over the gait cycle and how they are shaped to produce smooth transitions at phase boundaries.

Normalized Cycle Coordinate and Local Process Variable

Let

$$g = \frac{k}{N_{\mathrm{c}}}, \qquad k = 0, 1, \ldots, N_{\mathrm{c}} - 1, \tag{4.13}$$

denote the normalized cycle coordinate at time step k. Following the tripod-gait convention of Section 4.2.1, the cycle origin is placed at mid-stance for $\mathcal{T}_1$ and at mid-swing for $\mathcal{T}_2$, giving the stance and swing intervals

$$\mathcal{T}_1: \quad \text{stance: } g \in \left[0, \tfrac{1}{4}\right) \cup \left[\tfrac{3}{4}, 1\right), \quad \text{swing: } g \in \left[\tfrac{1}{4}, \tfrac{3}{4}\right),$$

with these two roles reversed for $\mathcal{T}_2$.

Within each phase, the relative progress is captured by a local process variable $\xi \in [0, 1]$ obtained by affine rescaling of g. For $i \in \mathcal{T}_1$:

$$\xi_{\mathrm{st},i} = \begin{cases} 2g + \dfrac{1}{2}, & 0 \le g < \dfrac{1}{4}, \\[2ex] 2g - \dfrac{3}{2}, & \dfrac{3}{4} \le g < 1, \end{cases} \qquad \xi_{\mathrm{sw},i} = 2g - \frac{1}{2}, \quad \frac{1}{4} \le g < \frac{3}{4}.$$

For $i \in \mathcal{T}_2$ the same maps apply with stance and swing interchanged. Thus $\xi = 0$ marks the

start of the current phase and $\xi = 1$ marks its end.

Smooth Polynomial Profiles

If μ were assigned by linear interpolation over ξ, the foot velocity and acceleration would be nonzero at phase boundaries, producing abrupt transitions at lift-off and touch-down. To enforce zero velocity and zero acceleration at both boundaries, μ is assigned through a fifth-order polynomial

$$\psi(\xi) = a_0 + a_1\xi + a_2\xi^2 + a_3\xi^3 + a_4\xi^4 + a_5\xi^5$$

subject to

$$\psi(0) = -1, \quad \psi(1) = 1,$$

$$\psi'(0) = 0, \quad \psi'(1) = 0,$$

$$\psi''(0) = 0, \quad \psi''(1) = 0.$$

Solving gives $\psi(\xi) = 20\xi^3 - 30\xi^4 + 12\xi^5 - 1$, and the normalized planar progress variables are

$$\mu_{\text{st}}(\xi) = -\psi(\xi), \qquad \mu_{\text{sw}}(\xi) = \psi(\xi). \tag{4.14}$$

Stance thus runs from $\mu = 1$ to $\mu = -1$ and swing from $\mu = -1$ to $\mu = 1$, consistent with the trajectory equations above.

For the vertical lift, a sixth-order polynomial satisfying zero height and zero velocity and acceleration at lift-off and touch-down, with unit normalized height at mid-swing, gives

$$\zeta_{\text{sw}}(\xi) = 64\xi^3(1-\xi)^3. \tag{4.15}$$

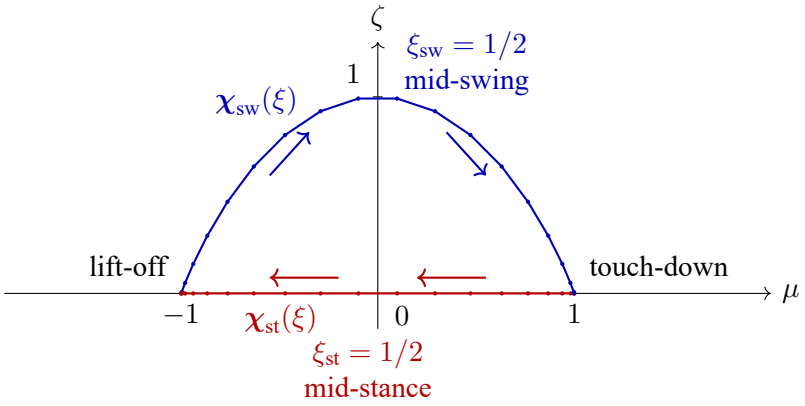


Figure 4.6. Normalized phase plane for one leg. Coordinate $\mu \in [-1, 1]$ is planar progression and $\zeta \in [0, 1]$ is normalized lift height. The foot moves $\mu : 1 \to -1$ on $\zeta = 0$ during stance and $\mu : -1 \to 1$ with peak lift at mid-swing.

The two normalized trajectory functions are collected as

$$\boldsymbol{\chi}_{\text{st}}(\xi) = \begin{bmatrix} \mu_{\text{st}}(\xi) \\ 0 \end{bmatrix} = \begin{bmatrix} 1 - 20\xi^3 + 30\xi^4 - 12\xi^5 \\ 0 \end{bmatrix},$$

$$\boldsymbol{\chi}_{\text{sw}}(\xi) = \begin{bmatrix} \mu_{\text{sw}}(\xi) \\ \zeta_{\text{sw}}(\xi) \end{bmatrix} = \begin{bmatrix} 20\xi^3 - 30\xi^4 + 12\xi^5 - 1 \\ 64\xi^3(1 - \xi)^3 \end{bmatrix}.$$

Substituting $\mu_{\text{st}}(\xi)$, $\mu_{\text{sw}}(\xi)$, and $\zeta_{\text{sw}}(\xi)$ into Eqs. (4.7)–(4.12) completes the foot-tip trajectory specification for any command (ω, v). Figure 4.6 shows $\boldsymbol{\chi}_{\text{st}}$ and $\boldsymbol{\chi}_{\text{sw}}$ in the abstract (μ, ζ) phase plane.

4.2.4 Assembly of the Full Tip Trajectory

For each leg i and sample k, the tripod-group membership of leg i together with the interval containing $g = k/N_{\mathrm{c}}$ selects the stance or swing construction:

$$\boldsymbol{p}_{\mathrm{des},i}(k) = \begin{cases} \boldsymbol{p}_{\mathrm{st},i}(\xi_{\mathrm{st},i}), & i \in \mathcal{T}_1,\ g \in \left[0, \frac{1}{4}\right) \cup \left[\frac{3}{4}, 1\right), \\ \boldsymbol{p}_{\mathrm{sw},i}(\xi_{\mathrm{sw},i}), & i \in \mathcal{T}_1,\ g \in \left[\frac{1}{4}, \frac{3}{4}\right), \\ \boldsymbol{p}_{\mathrm{st},i}(\xi_{\mathrm{st},i}), & i \in \mathcal{T}_2,\ g \in \left[\frac{1}{4}, \frac{3}{4}\right), \\ \boldsymbol{p}_{\mathrm{sw},i}(\xi_{\mathrm{sw},i}), & i \in \mathcal{T}_2,\ g \in \left[0, \frac{1}{4}\right) \cup \left[\frac{3}{4}, 1\right). \end{cases} \tag{4.16}$$

The six desired foot-tip positions at sample k are stacked into $\boldsymbol{p}_{\mathrm{tip}}(k) \in \mathbb{R}^{18}$, and the full cycle is collected as

$$\boldsymbol{P}_{\mathrm{tip}} = \begin{bmatrix} \boldsymbol{p}_{\mathrm{tip}}(0) & \boldsymbol{p}_{\mathrm{tip}}(1) & \cdots & \boldsymbol{p}_{\mathrm{tip}}(N_{\mathrm{c}} - 1) \end{bmatrix} \in \mathbb{R}^{18 \times N_{\mathrm{c}}}. \tag{4.17}$$

Equivalently, this object can be viewed as a $6 \times 3 \times N_{\mathrm{c}}$ collection indexed by leg, Cartesian coordinate, and time. As above, all entries of $\boldsymbol{P}_{\mathrm{tip}}$ are expressed in $\{\boldsymbol{b}\}$. This assembled $\boldsymbol{P}_{\mathrm{tip}}$ is the output of $\mathcal{G}$ in Eq. (4.1) and becomes the input to the inverse-kinematics stage.

4.3 Joint Command Generation via Inverse Kinematics

This stage realizes the inverse-kinematics mapping from the generated foot-tip trajectory to the joint-command trajectory,

$$\boldsymbol{Q} = \mathcal{I}(\boldsymbol{P}_{\mathrm{tip}}), \tag{4.18}$$

where $\boldsymbol{P}_{\text{tip}} \in \mathbb{R}^{18\times N_{\text{c}}}$ and $\boldsymbol{Q} \in \mathbb{R}^{18\times N_{\text{c}}}$. Since the leg-level Jacobian and damped inverse-kinematics solver were defined in Chapter 3, this chapter treats $\mathcal{I}$ in Eq. (4.18) as the repeated application of that solver to the generated trajectory.

For sample $k = 0, 1, \ldots, N_{\text{c}} - 1$ and leg $i = 1, \ldots, 6$, the desired body-frame foot-tip position $\boldsymbol{p}_{\text{des},i}(k) \in \mathbb{R}^3$ is converted into a leg-level joint solution $\boldsymbol{q}_i(k) \in \mathbb{R}^3$. Across the gait cycle, samples are processed sequentially. Let $\boldsymbol{q}_{i,\text{init}}(k)$ denote the initial guess used by the inverse-kinematics solver at sample k. The initialization is chosen as

$$\boldsymbol{q}_{i,\text{init}}(0) = \boldsymbol{q}_{i,0}, \qquad \boldsymbol{q}_{i,\text{init}}(k) = \boldsymbol{q}_i(k-1), \quad k = 1, \ldots, N_{\text{c}} - 1, \tag{4.19}$$

where $\boldsymbol{q}_{i,0}$ is the nominal initial configuration of leg i. With the initialization in Eq. (4.19), the leg-level solve can therefore be written as

$$\boldsymbol{q}_i(k) = \mathcal{I}_i\big(\boldsymbol{p}_{\text{des},i}(k), \boldsymbol{q}_{i,\text{init}}(k)\big)\,. \tag{4.20}$$

Since adjacent desired foot-tip positions are close along the generated trajectory, this initialization supports smooth and consistent joint evolution.

The six leg-level solutions at sample k are stacked as

$$\boldsymbol{q}(k) = \begin{bmatrix} \boldsymbol{q}_1(k)^\top & \boldsymbol{q}_2(k)^\top & \cdots & \boldsymbol{q}_6(k)^\top \end{bmatrix}^\top \in \mathbb{R}^{18}. \tag{4.21}$$

Repeating this hierarchical operation over the full gait cycle yields

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}(0) & \boldsymbol{q}(1) & \cdots & \boldsymbol{q}(N_{\mathrm{c}}-1) \end{bmatrix} \in \mathbb{R}^{18\times N_{\mathrm{c}}}, \tag{4.22}$$

which is precisely the trajectory-level realization of $\mathcal{I}(\boldsymbol{P}_{\mathrm{tip}})$ in Eq. (4.18).

4.4 Learning-Based Approximation of the Command Library

4.4.1 Model-Based Library as Supervision

The geometric generator and inverse-kinematics solver together define a composite model-based command map,

$$\mathcal{F} = \mathcal{I} \circ \mathcal{G}, \qquad \boldsymbol{Q} = \mathcal{F}(\omega, v, T, N_{\mathrm{c}}). \tag{4.23}$$

The composite map in Eq. (4.23) is physically interpretable, but generating commands through it online requires geometric trajectory construction and repeated iterative IK solves. Therefore, the model-based procedure is used offline to generate supervision for a compact learned approximation.

To construct the command library, admissible command pairs are sampled from a bounded region in (ω, v). In the reference implementation, this region is represented by an ellipse of the form

$$\beta\omega^2 + v^2 \le C,$$

where β is a weighting coefficient and C determines the command bound. For each sampled

command (ω_j, v_j), the model-based procedure produces a full-cycle joint trajectory

$$\boldsymbol{Q}_j = \begin{bmatrix} \boldsymbol{q}_j(0) & \boldsymbol{q}_j(1) & \cdots & \boldsymbol{q}_j(N_{\mathrm{c}}-1) \end{bmatrix}.$$

The resulting library provides teacher data for learning a fast surrogate of the model-based command rule.

4.4.2 Periodic Phase Encoding

For real-time command output, the learned model predicts a time-local joint command rather than regenerating the entire trajectory through $\mathcal{G}$ and $\mathcal{I}$. The training data are therefore formed at the sample level. For command pair j and sample k, define the phase angle

$$\varphi_k = 2\pi \frac{k}{N_{\mathrm{c}}}. \tag{4.24}$$

Because gait phase is periodic, using raw time or a scalar phase coordinate would introduce an artificial discontinuity at the cycle boundary. The phase is therefore represented by its sine and cosine components:

$$\boldsymbol{x}_{j,k} = \begin{bmatrix} \omega_j & v_j & \sin\varphi_k & \cos\varphi_k \end{bmatrix}^{\top}, \qquad \boldsymbol{y}_{j,k} = \boldsymbol{q}_j(k) \in \mathbb{R}^{18}. \tag{4.25}$$

The encoding in Eq. (4.25) preserves the circular structure of the gait phase, so samples near the end and beginning of a cycle remain close in the input space.

4.4.3 Supervised Approximation

The learned surrogate is implemented as a multilayer perceptron (MLP) that maps the low-dimensional command and phase encoding directly to an 18-dimensional joint command. It is written as

$$\hat{\boldsymbol{q}} = f_{\boldsymbol{\theta}}(\omega, v, \sin\varphi, \cos\varphi), \qquad f_{\boldsymbol{\theta}} : \mathbb{R}^4 \to \mathbb{R}^{18}, \tag{4.26}$$

where $\boldsymbol{\theta}$ denotes the neural-network parameters. The parameters are trained by minimizing the supervised regression loss

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{j,k} \left\| f_{\boldsymbol{\theta}}(\omega_j, v_j, \sin\varphi_k, \cos\varphi_k) - \boldsymbol{q}_j(k) \right\|^2, \tag{4.27}$$

and the training problem is

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{4.28}$$

The optimization problem in Eq. (4.28) is solved using stochastic gradient descent (SGD), which updates $\boldsymbol{\theta}$ from mini-batches sampled from the model-based command library. This mini-batch training procedure is well suited to the large number of sample-level command pairs generated over different (ω, v) values and gait phases. The resulting supervised training routine is summarized in Algorithm 2. After training, the joint command for a desired (ω, v) and phase φ can be generated by a single neural-network forward pass. Thus, the learned model serves as a fast approximation of the physically interpretable model-based generator, avoiding repeated online trajectory construction and iterative inverse kinematics during command execution.

Algorithm 2. Supervised SGD training for gait approximation

Require: Model-based library $\mathcal{D} = \{(\omega_j, v_j, \varphi_k, \boldsymbol{q}_j(k))\}$, network $f_{\boldsymbol{\theta}}$, learning rate α, mini-batch size B, epoch count E
Ensure: Trained parameters $\boldsymbol{\theta}^\star$

Initialize $\boldsymbol{\theta}$
for $e = 1, 2, \ldots, E$ **do**
  Shuffle the samples in $\mathcal{D}$
  for each mini-batch $\mathcal{B} \subset \mathcal{D}$ of size B **do**
    Encode phase samples in $\mathcal{B}$ as $(\sin\varphi, \cos\varphi)$
    Compute the mini-batch loss

$$\mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta}) = \sum_{(\omega, v, \varphi, \boldsymbol{q}) \in \mathcal{B}} \left\| f_{\boldsymbol{\theta}}(\omega, v, \sin\varphi, \cos\varphi) - \boldsymbol{q} \right\|^2$$

    Evaluate $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta})$
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}}(\boldsymbol{\theta})$
  end for
end for
return $\boldsymbol{\theta}^\star \leftarrow \boldsymbol{\theta}$

4.5 Summary

This chapter developed a structured locomotion layer that converts a compact body-velocity command $\boldsymbol{a}_t = [v_t, \omega_t]^\top$ into a full 18-DOF joint-command sequence. The layer is organized in two stages. The first stage is a model-based gait generator: given a body command, it constructs a tripod gait schedule, computes stance and swing foot-tip trajectories in the body frame using smooth phase profiles, and assembles the full six-leg Cartesian trajectory. The second stage is inverse kinematics (IK), which maps each foot-tip position to the corresponding joint angles. Together, these two stages provide a physically grounded and interpretable mapping from body-level commands to joint-level execution.

Because running this model-based procedure online at every command cycle would require repeated geometric trajectory construction and IK solves, a compact learned approximation

is trained to replicate the mapping offline. The model-based generator is applied across a discrete grid of (v, ω) command pairs to produce a supervision dataset. A neural network with periodic phase encoding then learns the command-to-joint mapping from this data through supervised regression. At execution time, only the learned network is evaluated, so the full joint-command sequence is obtained efficiently without online model-based computation.

The locomotion layer established in this chapter provides a solid structural foundation for the learning-based navigation methods developed in Chapter 5. Because joint-level execution is fully handled by this layer, the navigation policy only needs to select a two-dimensional body command from sensory input. This separation keeps the learning problem focused and tractable: the policy never reasons about foot placement, swing phases, or joint angles, and the locomotion layer never reasons about obstacles, goals, or perception. The clean interface between the two layers is what makes the overall system both modular and learnable.

Chapter 5

Vision-Guided Learning-Based Navigation

Chapter 3 and Chapter 4 define how a compact body-velocity command is converted into executable hexapod motion. The role of this chapter is to study the complementary observation-to-command problem: using visual input, depth information, proprioception, and target information to select the body-velocity command for waypoint tracking. Recent work in visual and visual-proprioceptive legged navigation shows that perception can be coupled with locomotion for goal-directed robot movement [9], and perceptive locomotion research demonstrates how sensory information can support terrain-aware planning and control on legged platforms [33]. The formulation used here adapts this perception-guided navigation idea to the body-velocity command interface of the hexapod system.

Another contribution of this chapter is to improve the command-learning method by adapting existing reinforcement-learning ideas to this hexapod waypoint-tracking problem. Proximal policy optimization (PPO) is used as the baseline policy-gradient method [28]. A mixture-of-experts (MoE) structure organizes command selection among specialized behaviors [34], and Replay-Enhanced Clipped Actor-Critic Policy Learning (RECAP) introduces replay-enhanced clipped policy optimization to improve sample usage, drawing on replay-based value learning

and twin-critic ideas [25, 38, 39]. RRT* provides waypoint guidance from geometric map information [13], while the learned policy remains responsible for local tracking and command selection from sensory feedback.

5.1 Route-Conditioned Locomotion

The closed-loop structure is shown in Fig. 5.1. The planner generates a route, the waypoint manager selects the current target, the policy maps local sensing to the command $[v_t, \omega_t]^\top$, and the gait generator converts this command to joint-level motion. Vision and proprioception close the loop from the robot back to the policy.

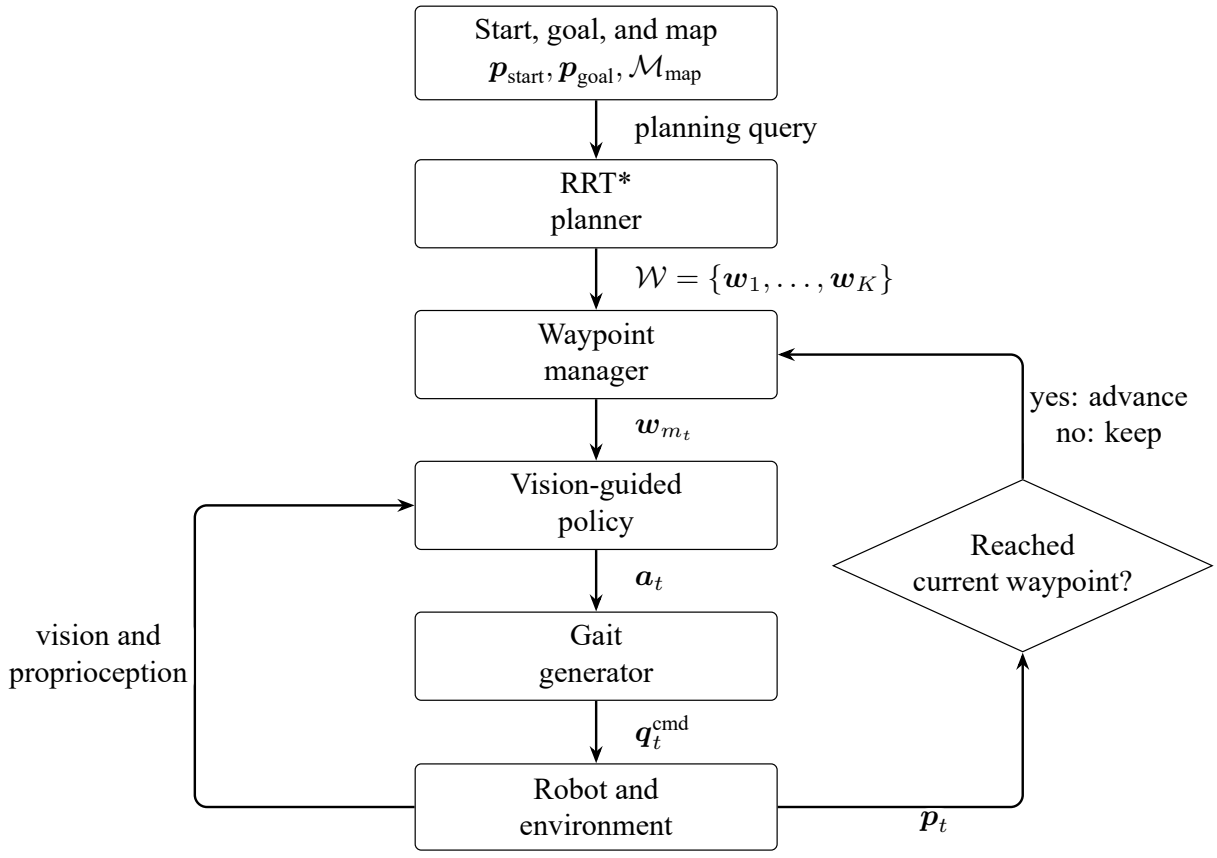


Figure 5.1. Route-conditioned navigation loop. A planner generates a waypoint sequence from the start, goal, and map; a waypoint manager advances or holds the active target; the learned policy closes the inner loop using vision, proprioception, and the active waypoint.

Let $\boldsymbol{p}_t \in \mathbb{R}^3$ denote the robot body-center position in the world frame. The planner outputs a waypoint sequence

$$\mathcal{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K\}, \qquad \boldsymbol{w}_k \in \mathbb{R}^3. \tag{5.1}$$

The waypoint manager stores the active waypoint index m_t. The index is updated by

$$m_{t+1} = \begin{cases} \min(m_t + 1, K), & \|\boldsymbol{p}_t - \boldsymbol{w}_{m_t}\|_2 \le \varepsilon_w, \\ m_t, & \|\boldsymbol{p}_t - \boldsymbol{w}_{m_t}\|_2 > \varepsilon_w, \end{cases} \tag{5.2}$$

where ε_w is the waypoint tolerance. This rule converts a global route into a sequence of local tracking objectives.

5.2 MDP Formulation for Planar Waypoint Tracking

The waypoint-tracking problem is solved using reinforcement learning [22]. To apply reinforcement learning, the closed-loop interaction between the robot and the environment is first written as a Markov decision process (MDP). In this formulation, the policy receives a state, selects an action, and then receives the next state and reward after the action is executed. The MDP is

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma), \tag{5.3}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the state transition model, $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ is the one-step transition reward, and $\gamma \in (0, 1)$ is the discount factor. In implementation, rollouts are truncated after a maximum number of command cycles for training and evaluation.

5.2.1 State Space and Action Space

The state $\boldsymbol{s}_t = (\boldsymbol{\zeta}_t, \boldsymbol{\eta}_t, \boldsymbol{g}_t)$ is defined in full in Chapter 2 (Eqs. (2.7)–(2.11)), where $\boldsymbol{\zeta}_t$ is visual sensing, $\boldsymbol{\eta}_t$ is proprioception, and $\boldsymbol{g}_t$ encodes the bearing and distance to the active waypoint. The learned MDP action is the body-velocity command

$$\boldsymbol{a}_t = \begin{bmatrix} v_t & \omega_t \end{bmatrix}^\top \in \mathcal{A}, \qquad \mathcal{A} = [v_{\min}, v_{\max}] \times [\omega_{\min}, \omega_{\max}], \tag{5.4}$$

where v_t is the forward body velocity command and ω_t is the yaw-rate command.

5.2.2 State Transition Model

Given the body-velocity action $\boldsymbol{a}_t$, the command is converted by the gait-generation map into a sequence of joint commands. Executing this sequence for one control cycle produces the next state for the following control loop:

$$\begin{aligned} \boldsymbol{Q}_t^{\text{cmd}} &= \mathcal{F}\left(\omega_t, v_t, T, N_{\text{c}}\right) = \begin{bmatrix} \boldsymbol{q}_t^{\text{cmd}}(0) & \boldsymbol{q}_t^{\text{cmd}}(1) & \cdots & \boldsymbol{q}_t^{\text{cmd}}(N_{\text{c}}-1) \end{bmatrix}, \\ \boldsymbol{s}_{t+1} &\sim P(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t) = P(\cdot \mid \boldsymbol{s}_t, \boldsymbol{Q}_t^{\text{cmd}}). \end{aligned} \tag{5.5}$$

Here $\mathcal{F}$ is the composite command map from Chapter 4, and $\boldsymbol{Q}_t^{\text{cmd}} \in \mathbb{R}^{18 \times N_{\text{c}}}$ is the full joint-command sequence executed during the command cycle. The state transition model P is therefore induced by the current state and the executed command sequence.

5.2.3 Reward and Termination

The reward is assigned after the action has been executed, so it is written as a function of the state transition:

$$r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = -\left(c_t + c_t^{\text{term}}\right), \tag{5.6}$$

where the edge cost is

$$c_t = c(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = \sum_{i=1}^{5} c_{t,i}. \tag{5.7}$$

$$c_{t,1} = w_g \left\| \boldsymbol{p}_{t+1} - \boldsymbol{p}_{\text{goal}} \right\|_2, \tag{5.8}$$

$$c_{t,2} = w_E E_t, \tag{5.9}$$

$$c_{t,3} = w_{\text{col}} \mathbb{I}_{\text{col,t}}, \tag{5.10}$$

$$c_{t,4} = w_u \left\| \boldsymbol{a}_t \right\|_2^2, \tag{5.11}$$

$$c_{t,5} = w_{\Delta u} \left\| \boldsymbol{a}_t - \boldsymbol{a}_{t-1} \right\|_2^2. \tag{5.12}$$

The terms in Eqs. (5.8)– (5.12) are, respectively, the goal-distance cost, energy cost, collision cost, action-magnitude cost, and action-change cost. The goal-distance, action-magnitude, and action-change costs are computed directly from the post-transition position and the current and previous actions. The energy and collision costs require auxiliary quantities accumulated over

the transition. The work term used in $c_{t,2}$ is

$$E_t = \sum_{k=1}^{N_c} \sum_{j=1}^{18} |\tau_{j,k} \dot{q}_{j,k}| \, \Delta t, \tag{5.13}$$

where $\tau_{j,k}$ and $\dot{q}_{j,k}$ are the actuator torque and joint velocity of joint j at inner simulation step k.

The collision indicator used in $c_{t,3}$ is

$$\mathbb{I}_{\text{col,t}} = \begin{cases} 1, & \text{if a collision occurs during } (\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}), \\ 0, & \text{otherwise.} \end{cases} \tag{5.14}$$

After the edge cost is computed, the terminal term adds the effect of success or failure:

$$c_t^{\text{term}} = \begin{cases} -R_{\text{suc}}, & \text{if the success condition is satisfied,} \\ C_{\text{out}}, & \text{if } \boldsymbol{p}_{t+1} \notin \mathcal{X}_{\text{work}}, \\ 0, & \text{otherwise,} \end{cases} \tag{5.15}$$

where $R_{\text{suc}} > 0$ is the success bonus and $C_{\text{out}} > 0$ is the out-of-workspace penalty.

The success condition is evaluated after the transition:

$$\left\| \boldsymbol{p}_{t+1} - \boldsymbol{p}_{\text{goal}} \right\|_2 \leq \varepsilon_{\text{goal}}, \qquad \| \boldsymbol{v}_{t+1} \|_2 \leq \varepsilon_v. \tag{5.16}$$

5.2.4 Optimization Objective

The policy is represented as

$$\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t), \tag{5.17}$$

where $\pi_{\boldsymbol{\theta}}$ maps the current state to a distribution over actions. It is represented by a neural network, and $\boldsymbol{\theta}$ denotes the collection of trainable network parameters. The objective is to choose these parameters to maximize the expected discounted return. The discount factor γ weights future rewards relative to immediate rewards:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \qquad J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\tau} \gamma^t r_t\right], \tag{5.18}$$

Here τ is the first exit time defined in Chapter 2 (Eq. (2.16)): the episode terminates when the robot first reaches a goal or out-of-bound state. Since the reward in Eq. (5.6) is defined as the negative of the cost, maximizing $J(\boldsymbol{\theta})$ is equivalent to minimizing the expected discounted navigation cost over the first exit horizon.

5.3 Reinforcement Learning for MDP Optimization

The MDP in Eq. (5.3) defines the optimization problem that the learning algorithm must solve. Reinforcement learning addresses this problem by improving a policy from sampled state transitions rather than requiring an analytic transition model. In this setting, a sampled transition provides the state, the selected action, the next state, and the reward; repeated interaction then supplies the data used to estimate long-term return and improve the policy in Eq. (5.18).

5.3.1 Value Functions and Bellman Equations

For a policy π, the value function is

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid \boldsymbol{s}_t = \boldsymbol{s}\right], \tag{5.19}$$

and the action-value function is

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \mid \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}\right]. \tag{5.20}$$

These functions satisfy Bellman expectation equations [40, 22]. For the state value,

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi,\, \boldsymbol{s}'\sim P}\left[r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V^{\pi}(\boldsymbol{s}')\right]. \tag{5.21}$$

For the action value,

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}'\sim P,\, \boldsymbol{a}'\sim\pi}\left[r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma Q^{\pi}(\boldsymbol{s}', \boldsymbol{a}')\right]. \tag{5.22}$$

These recursive relations show how the immediate reward and the value of the next state or next action determine the long-term value under a fixed policy. In policy-gradient algorithms, these value estimates are often used as baselines or action-value estimates for improving the policy without requiring an explicit analytic model of the transition dynamics.

5.3.2 Policy Gradient Methods with Value Baselines

Policy-gradient methods directly improve the parameters of a stochastic policy [26]. Let the sampled discounted return from time t be

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{5.23}$$

A basic policy-gradient estimator has the form

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) G_t\right]. \tag{5.24}$$

This estimator increases the probability of sampled actions that lead to high return, but the sampled return can have high variance. A state-dependent baseline can reduce this variance without changing the expected gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) \left(G_t - b(\boldsymbol{s}_t)\right)\right]. \tag{5.25}$$

In practice, the baseline is commonly represented by a learned value function, such as $V_{\boldsymbol{\phi}}(\boldsymbol{s})$, or by an action-value function $Q_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a})$. This policy-and-value structure is the basis of actor-critic algorithms [27]. With the learned value estimate as the baseline, the learning signal is the advantage function

$$A^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - V^{\pi}(\boldsymbol{s}_t) \tag{5.26}$$

which measures whether the sampled action is better or worse than the policy's average action

at the same state. The policy update then uses an estimated advantage:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)\widehat{A}_t\right]. \tag{5.27}$$

Actions with positive advantage become more likely, while actions with negative advantage become less likely. The learned value function therefore provides a variance-reduced learning signal for updating the policy.

5.4 Deep Reinforcement Learning Algorithms for Navigation

This section introduces the deep reinforcement learning algorithms used for learning-based navigation through the body-velocity command interface. The continuous PPO policy is the baseline policy trained with a learned value-function baseline. The mixture-of-experts architecture provides a structured policy representation that lets the policy choose among command experts [34]. RECAP addresses PPO's sample inefficiency by combining replay, twin Q critics, and expected-policy value backups.

5.4.1 Policy and Value Function Representation

Before introducing the individual learning algorithms, this subsection defines how the policy and value function are represented. The state branches are: stereo RGB images are encoded by a convolutional neural network (CNN) branch, depth is encoded by a second CNN branch, and proprioception and the target input are mapped by multilayer perceptrons (MLPs). All branch encoders and the final map are trainable. For the policy network (Fig. 5.2), the branch

parameters are grouped into the encoder parameter set

$$\boldsymbol{\theta}^{\pi}_{\text{enc}} = \left\{\boldsymbol{\theta}^{\pi}_{\text{rgb}}, \boldsymbol{\theta}^{\pi}_{\text{dep}}, \boldsymbol{\theta}^{\pi}_{\text{prop}}, \boldsymbol{\theta}^{\pi}_{\text{goal}}\right\}, \qquad \boldsymbol{\theta} = \left\{\boldsymbol{\theta}^{\pi}_{\text{enc}}, \boldsymbol{\theta}^{\pi}_{\text{head}}\right\}. \tag{5.28}$$

The policy latent representation is

$$\boldsymbol{z}^{\pi}_{t} = f^{\pi}_{\text{enc}}\left(\boldsymbol{s}_t; \boldsymbol{\theta}^{\pi}_{\text{enc}}\right). \tag{5.29}$$

Here f^{π}_{enc} denotes the branch encoding and feature-fusion operation shown in Fig. 5.2.

The policy map sends this latent vector to the Gaussian parameters:

$$(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) = h_{\boldsymbol{\theta}^{\pi}_{\text{head}}}(\boldsymbol{z}^{\pi}_{t}). \tag{5.30}$$

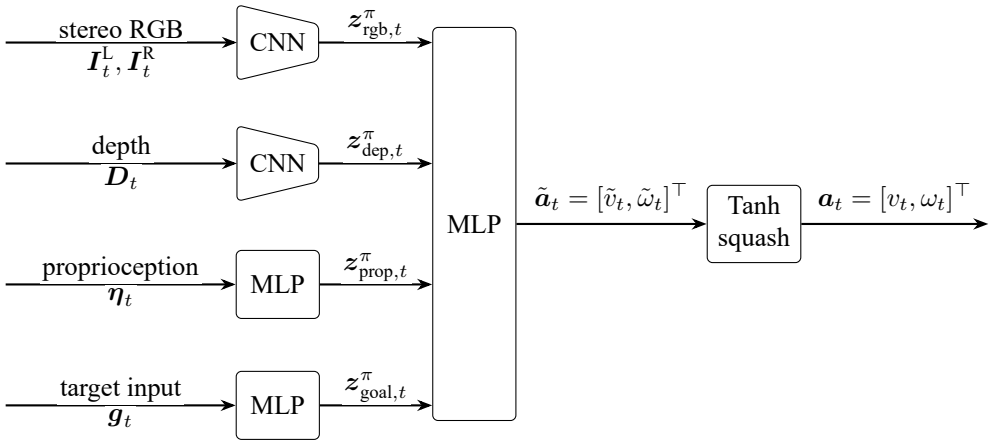


Figure 5.2. Policy network architecture. Branch encoders compress visual, depth, proprioceptive, and target inputs into feature vectors fused by the policy MLP, then passed through a Tanh squash to produce the body-velocity command $\boldsymbol{a}_t = [v_t, \omega_t]^{\top}$.

The policy then represents the continuous action distribution in a pre-squash action co-

ordinate:

$$\tilde{\boldsymbol{a}}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\right), \qquad \boldsymbol{a}_t = \boldsymbol{a}_{\mathrm{c}} + \boldsymbol{a}_{\mathrm{r}} \odot \tanh(\tilde{\boldsymbol{a}}_t), \tag{5.31}$$

where $\boldsymbol{a}_{\mathrm{c}}$ and $\boldsymbol{a}_{\mathrm{r}}$ are the center and half-range of the action box, and $\odot$ denotes element-wise multiplication. The log probability of the executed action includes the change-of-variables correction:

$$\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) = \log p_{\boldsymbol{\theta}}(\tilde{\boldsymbol{a}}_t \mid \boldsymbol{s}_t) - \log \left| \det \frac{\partial \boldsymbol{a}_t}{\partial \tilde{\boldsymbol{a}}_t} \right|. \tag{5.32}$$

The value network (Fig. 5.3) is encoded separately but uses the same branch organization. Its branch parameters are grouped into $\boldsymbol{\phi}^V_{\mathrm{enc}}$:

$$\boldsymbol{\phi}^V_{\mathrm{enc}} = \left\{\boldsymbol{\phi}^V_{\mathrm{rgb}}, \boldsymbol{\phi}^V_{\mathrm{dep}}, \boldsymbol{\phi}^V_{\mathrm{prop}}, \boldsymbol{\phi}^V_{\mathrm{goal}}\right\}, \qquad \boldsymbol{\phi} = \left\{\boldsymbol{\phi}^V_{\mathrm{enc}}, \boldsymbol{\phi}^V_{\mathrm{head}}\right\}. \tag{5.33}$$

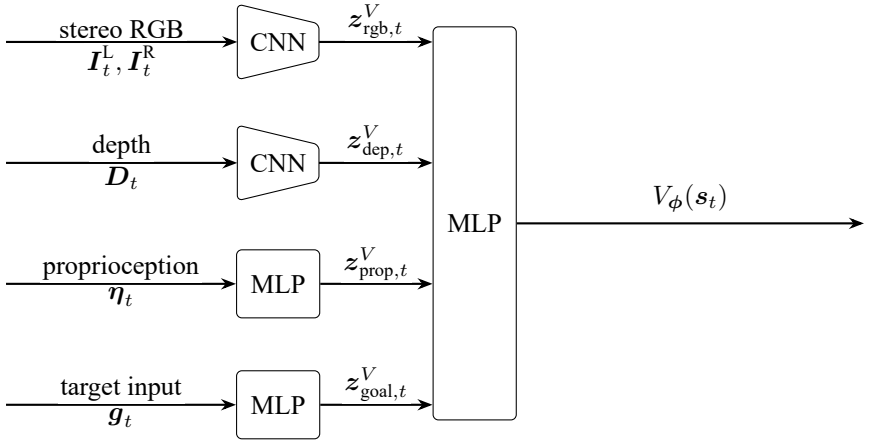


Figure 5.3. Value network architecture. The same input-branch organization as Fig. 5.2 with separate parameters $\boldsymbol{\phi}$; the MLP output is a scalar state-value estimate $V_{\boldsymbol{\phi}}(\boldsymbol{s}_t)$ with no squash stage.

The value network maps the state to a scalar state-value estimate:

$$\boldsymbol{z}_t^V = f_{\text{enc}}^V\left(\boldsymbol{s}_t; \boldsymbol{\phi}_{\text{enc}}^V\right), \tag{5.34}$$

$$V_{\boldsymbol{\phi}}(\boldsymbol{s}_t) = h_{\boldsymbol{\phi}_{\text{head}}^V}(\boldsymbol{z}_t^V). \tag{5.35}$$

5.4.2 Continuous PPO Baseline

The first concrete baseline trains the continuous body-velocity policy with proximal policy optimization (PPO), a policy-gradient method with a learned value-function baseline [28]. PPO compares the current policy to the frozen rollout policy through the probability ratio

$$\rho_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}. \tag{5.36}$$

In implementation, the equivalent log-difference form is often used for numerical stability, since the squashed Gaussian policy naturally provides log probabilities. The advantage can be estimated using generalized advantage estimation (GAE) [41]. Let $d_t \in \{0, 1\}$ denote the termination indicator for the transition at time t. Then

$$\delta_t = r_t + \gamma\,(1 - d_t)\,V_{\boldsymbol{\phi}}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\phi}}(\boldsymbol{s}_t), \tag{5.37}$$

$$\widehat{A}_t^{\text{GAE}} = \sum_{\ell=0}^{T-t-1} (\gamma\lambda)^{\ell}\delta_{t+\ell}. \tag{5.38}$$

The corresponding value target is written as

$$\widehat{R}_t = \widehat{A}_t^{\text{GAE}} + V_{\boldsymbol{\phi}_{\text{old}}}(\boldsymbol{s}_t), \tag{5.39}$$

where $V_{\phi_{\text{old}}}$ denotes the value function used during rollout evaluation.

Given the advantage estimate, the PPO policy objective is

$$L_{\pi}^{\text{PPO}}(\boldsymbol{\theta}) = \mathbb{E}_t \left[\min \left(\rho_t(\boldsymbol{\theta}) \widehat{A}_t, \text{clip}(\rho_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) \widehat{A}_t \right) \right] + \alpha_H \mathbb{E}_t \left[\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)) \right]. \tag{5.40}$$

The clipping term prevents a single batch of rollouts from moving the policy too far. The entropy term encourages exploration. The value function is trained by regression to the target in Eq. (5.39):

$$L_{V}^{\text{PPO}}(\boldsymbol{\phi}) = \mathbb{E}_t \left[\left(V_{\boldsymbol{\phi}}(\boldsymbol{s}_t) - \widehat{R}_t \right)^2 \right]. \tag{5.41}$$

In this project implementation, the policy network and the value network are trained with separate objectives: the policy parameters $\boldsymbol{\theta}$ maximize Eq. (5.40), while the value parameters $\boldsymbol{\phi}$ minimize Eq. (5.41). In practice the policy update is also monitored by the approximate KL divergence

$$D_{\text{KL}}(\pi_{\boldsymbol{\theta}_{\text{old}}} \| \pi_{\boldsymbol{\theta}}) = \mathbb{E}_t \left[\log \pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) - \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) \right]. \tag{5.42}$$

If this value exceeds a threshold, the policy update is stopped early. The complete PPO baseline update used in this chapter is summarized in Algorithm 3.

The PPO baseline is useful because the clipped ratio limits large policy changes, the Gaussian policy naturally supports the continuous action $[v_t, \omega_t]^\top$, and the method gives a direct policy-gradient baseline with a learned value function for learning-based navigation. However, two limitations are important for this task:

Algorithm 3. Continuous PPO baseline

Initialize policy $\pi_{\boldsymbol{\theta}}$ and value function $V_{\boldsymbol{\phi}}$.
for PPO iteration $k = 1, 2, \ldots$ **do**
    Collect on-policy rollouts using $\pi_{\boldsymbol{\theta}}$.
    Compute rewards r_t using Eq. (5.6).
    Estimate advantages $\widehat{A}_t^{\mathrm{GAE}}$ using Eqs. (5.37)–(5.38).
    Compute value targets $\widehat{R}_t$ using Eq. (5.39).
    Freeze the rollout policy as $\pi_{\boldsymbol{\theta}_{\mathrm{old}}}$.
    for epoch $e = 1, \ldots, E$ **do**
        Update the policy network parameters $\boldsymbol{\theta}$ by gradient ascent on Eq. (5.40).
        Update the value network parameters $\boldsymbol{\phi}$ by gradient descent on Eq. (5.41).
        Stop the policy update if Eq. (5.42) exceeds the KL threshold.
    end for
end for

- **Convergence and local optimality.** A Gaussian policy may require many iterations to move probability mass from the current action region toward useful command modes, and the policy can settle into locally suboptimal behaviors such as oscillation or overuse of turning actions.

- **Sample efficiency.** PPO is on-policy, so each policy update relies mainly on fresh rollouts and has limited reuse of past transitions.

The next subsection first introduces the mixture-of-experts policy architecture, which structures command selection through a coordinator and a library of experts. When this architecture is trained with PPO, it gives the PPO-MoE variant. The following subsection then introduces RECAP, which improves sample reuse through replay-enhanced policy optimization with learned Q functions.

5.4.3 Mixture-of-Experts Policy Architecture

The mixture-of-experts (MoE) architecture addresses the convergence and local-optimality limitation of the continuous policy by replacing direct discovery of all action patterns with discrete selection among structured experts. In this project, the MoE policy defines a coordinator that selects among a fixed library of command experts. When this policy architecture is trained with PPO, the resulting method is denoted PPO-MoE; when it is combined with RECAP, the resulting method is denoted RECAP-MoE. Each expert proposes a candidate action,

$$\boldsymbol{a}_t^{(i)} = \mathcal{E}_i(\boldsymbol{s}_t), \qquad i = 1, \ldots, N_e, \tag{5.43}$$

and the coordinator selects one expert. The implementation uses six experts:

- $\mathcal{E}_1$: baseline learned continuous policy;
- $\mathcal{E}_2$: PD waypoint-tracking controller;
- $\mathcal{E}_3$: constant forward command;
- $\mathcal{E}_4$: constant backward command;
- $\mathcal{E}_5$: positive in-place yaw command;
- $\mathcal{E}_6$: negative in-place yaw command.

The PD expert uses the body-frame target vector to produce a goal-directed command:

$$v_t^{\mathrm{PD}} = \mathrm{clip}\left(k_{p,v} r_{y,t}^{\boldsymbol{b}} - k_{d,v} v_{y,t}^{\boldsymbol{b}}, v_{\min}, v_{\max}\right),$$
$$\omega_t^{\mathrm{PD}} = \mathrm{clip}\left(-k_{p,\omega} e_{\psi,t} - k_{d,\omega} \omega_{z,t}^{\boldsymbol{b}}, \omega_{\min}, \omega_{\max}\right), \tag{5.44}$$

where $r_{y,t}^{\boldsymbol{b}}$ is the forward component of the body-frame target vector and $e_{\psi,t}$ is the planar heading error.

The coordinator receives the state, the expert commands, and optionally the previous expert indicator:

$$\boldsymbol{z}_t = f_{\boldsymbol{\nu}}\left(\boldsymbol{s}_t, \boldsymbol{a}_t^{(1)}, \ldots, \boldsymbol{a}_t^{(N_e)}, \boldsymbol{e}_{t-1}\right), \qquad \pi_{\boldsymbol{\nu}}(i \mid \boldsymbol{s}_t) = \mathrm{softmax}(\boldsymbol{z}_t)_i. \tag{5.45}$$

The selected command is

$$i_t \sim \mathrm{Categorical}\left(\pi_{\boldsymbol{\nu}}(\cdot \mid \boldsymbol{s}_t)\right), \qquad \boldsymbol{a}_t = \boldsymbol{a}_t^{(i_t)}. \tag{5.46}$$

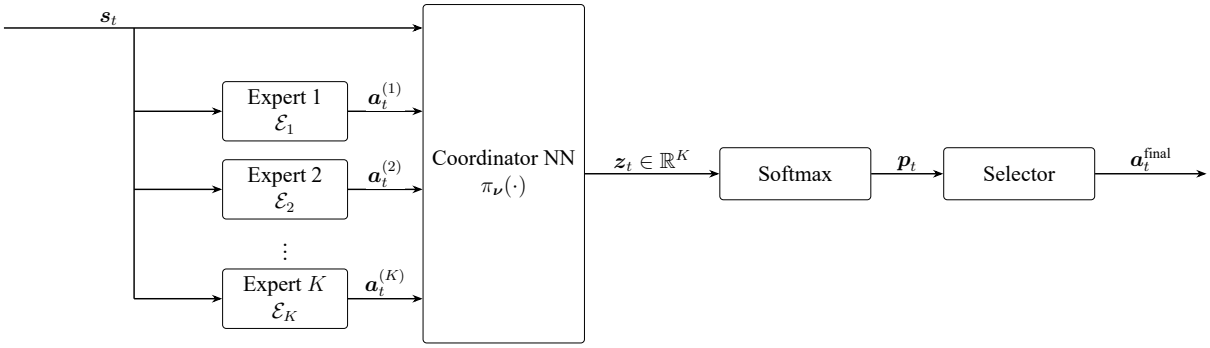


Figure 5.4. Mixture-of-experts coordinator policy. Each expert produces a candidate command from the current state; the coordinator scores the experts using the state and candidates; the selector outputs the final command.

The baseline MoE learner applies PPO to the discrete expert index. Its probability ratio is

$$\rho_t(\boldsymbol{\nu}) = \frac{\pi_{\boldsymbol{\nu}}(i_t \mid \boldsymbol{s}_t)}{\pi_{\boldsymbol{\nu}_{\text{old}}}(i_t \mid \boldsymbol{s}_t)}. \tag{5.47}$$

This structure makes early learning easier because the coordinator chooses among meaningful behaviors rather than discovering every action pattern from scratch.

5.4.4 RECAP: Replay-Enhanced Clipped Actor-Critic Policy Optimization

The PPO baseline in Section 5.4.2 estimates the state value $V_{\boldsymbol{\phi}}(\boldsymbol{s})$ from sampled trajectory returns through generalized advantage estimation. This design has a structural limitation: value information propagates slowly because only states visited in the current on-policy rollout batch are updated, and each policy improvement step must wait for a fresh set of trajectories. RECAP makes two targeted modifications to address this. The first replaces the V critic with twin Q critics, which support off-policy learning from a replay buffer. The second extends the actor update to exploit data from both online rollouts and offline sources (replay buffer and teacher demonstrations). These two modifications together improve sample efficiency while preserving the conservative policy-update behavior of PPO.

Modification 1: Q-Based Critic with Replay

RECAP replaces the V critic with twin Q critics:

$$\pi_{\boldsymbol{\theta}}(a \mid \boldsymbol{s}), \qquad Q_{\boldsymbol{\phi}_1}(\boldsymbol{s}, a), \qquad Q_{\boldsymbol{\phi}_2}(\boldsymbol{s}, a). \tag{5.48}$$

A Q function provides an action-conditioned value estimate, which is important in RECAP because online, replayed, and teacher transitions contain specific executed actions whose effects must be evaluated directly. The critic scores each transition through $Q(\boldsymbol{s}, a)$, and the resulting bootstrapped targets affect both the critic update and the policy advantage computation. This makes the learned value signal directly tied to the action used in each stored or demonstrated transition. To reduce overestimation bias [42, 39], the conservative minimum of the two critics is used wherever a Q value is needed:

$$Q_{\min,\boldsymbol{\phi}}(\boldsymbol{s}, a) = \min\left\{Q_{\boldsymbol{\phi}_1}(\boldsymbol{s}, a), Q_{\boldsymbol{\phi}_2}(\boldsymbol{s}, a)\right\}. \tag{5.49}$$

Since V is not directly parameterized, the state value is recovered from Q by taking the expectation over actions sampled from the policy:

$$V_{\pi,Q}(\boldsymbol{s}) = \mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{s})}\left[Q_{\min,\boldsymbol{\phi}}(\boldsymbol{s}, a)\right]. \tag{5.50}$$

For the MoE coordinator, where the expert set is finite, this expectation is computed as an exact sum:

$$V^{\text{MoE}}(\boldsymbol{s}) = \sum_{i=1}^{N_e} \pi_{\boldsymbol{\nu}}(i \mid \boldsymbol{s})\, Q_{\min,\boldsymbol{\phi}}(\boldsymbol{s}, i). \tag{5.51}$$

For the continuous command policy, the expectation is approximated by a Monte Carlo average over K actions sampled from the policy:

$$\widehat{V}^{(K)}(\boldsymbol{s}) = \frac{1}{K}\sum_{k=1}^{K} Q_{\min,\boldsymbol{\phi}}\left(\boldsymbol{s}, \boldsymbol{a}^{(k)}\right), \qquad \boldsymbol{a}^{(k)} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}). \tag{5.52}$$

The sampled actions use the squashed Gaussian transformation in Eq. (5.31). In both cases no additional network parameters beyond the two Q critics are required.

To enable replay, the cycle data pool combines three sources:

$$\mathcal{B}_{\text{cycle}} = \mathcal{B}_{\text{on}} \cup \mathcal{B}_{\text{rep}} \cup \mathcal{B}_{\text{tea}}, \tag{5.53}$$

where $\mathcal{B}_{\text{on}}$ is freshly collected online rollout data, $\mathcal{B}_{\text{rep}}$ is a batch sampled from the historical replay buffer, and $\mathcal{B}_{\text{tea}}$ is an optional set of teacher demonstrations. At the beginning of each training cycle, the current policy and Q-function parameters are frozen as reference copies:

$$\pi_{\text{old}} \leftarrow \pi_{\boldsymbol{\theta}}, \qquad \bar{\boldsymbol{\phi}}_i \leftarrow \boldsymbol{\phi}_i, \quad i \in \{1, 2\}. \tag{5.54}$$

All critic targets and actor advantages within the cycle are computed from these frozen references, so that updates inside the cycle are conditionally independent. Fresh online data is appended to the replay buffer only after the cycle ends.

The shared reference value derived from the frozen pair is

$$V_{\text{old}}(\boldsymbol{s}) = \mathbb{E}_{a \sim \pi_{\text{old}}(\cdot|\boldsymbol{s})} \left[Q_{\min, \bar{\boldsymbol{\phi}}}(\boldsymbol{s}, a) \right], \tag{5.55}$$

evaluated by Eq. (5.51) for MoE actions and by Eq. (5.52) for continuous actions, with $\boldsymbol{\phi}$ replaced by $\bar{\boldsymbol{\phi}}$ and $\pi_{\boldsymbol{\theta}}$ replaced by π_{old}.

Critic targets are constructed differently for replay transitions and trajectory data. For a replay transition $(\boldsymbol{s}, a, r, \boldsymbol{s}', d)$, the expected TD target replaces the hard-max backup with an

expected-policy backup:

$$y_{\text{rep}} = r + \gamma(1-d)V_{\text{old}}(\boldsymbol{s}'). \tag{5.56}$$

For online and teacher trajectory data, the temporal structure of the rollout is exploited by combining V_{old} with a GAE advantage estimate $\widehat{A}_t$, defined in Eqs. (5.59)–(5.60), to produce a compatible Q target:

$$y_t^{\text{traj}} = V_{\text{old}}(\boldsymbol{s}_t) + \widehat{A}_t. \tag{5.57}$$

Each critic is trained independently by minimizing the mean squared error on the combined source-labeled targets:

$$L_{Q_i}(\boldsymbol{\phi}_i) = \mathbb{E}_{(\boldsymbol{s},a,y)\sim\mathcal{B}_Q}\left[\left(Q_{\boldsymbol{\phi}_i}(\boldsymbol{s},a) - y\right)^2\right], \quad i \in \{1,2\}. \tag{5.58}$$

Modification 2: Extended Actor Update

The second modification extends the actor update to handle all three data sources and introduces a source-aware advantage computation. Two ingredients are required: the advantage estimate and the probability ratio.

Advantage by data source. The advantage is computed differently for each source to reflect its temporal structure and staleness.

For online rollouts, the one-step TD residual and GAE advantage are computed using

V_{old} as the value baseline:

$$\delta_t = r_t + \gamma(1 - d_t)V_{\text{old}}(\boldsymbol{s}_{t+1}) - V_{\text{old}}(\boldsymbol{s}_t), \tag{5.59}$$

$$\widehat{A}_t^{\text{GAE}} = \sum_{\ell=0}^{T-t-1} (\gamma\lambda)^\ell \delta_{t+\ell}. \tag{5.60}$$

Both positive and negative advantages are retained for online data.

For replay transitions, the temporal context of the original trajectory is unavailable, so the advantage is evaluated directly from the Q critics:

$$\widehat{A}_Q(\boldsymbol{s}, a) = Q_{\min,\bar{\boldsymbol{\phi}}}(\boldsymbol{s}, a) - V_{\text{old}}(\boldsymbol{s}). \tag{5.61}$$

Because replay transitions may be stale, negative advantages can be down-weighted by a coefficient $\beta \in [0, 1]$:

$$\widehat{A}_{\text{rep}}^{(\beta)} = \max(\widehat{A}_Q, 0) + \beta \min(\widehat{A}_Q, 0). \tag{5.62}$$

Setting $\beta = 1$ retains all advantages; $\beta = 0$ suppresses negative ones entirely.

For teacher demonstrations, only positive GAE advantages are kept so that teacher data encourages good actions without penalizing weaker ones:

$$\widehat{A}_{\text{tea}}^{+} = \max\left(\widehat{A}_{\text{tea}}^{\text{GAE}}, 0\right). \tag{5.63}$$

Before the policy update, all advantages from all sources are normalized by their root mean square to stabilize gradient magnitudes across heterogeneous data sources.

Probability ratio. The second ingredient is the probability ratio. At the cycle start, π_{old} is frozen to provide the reference action probability $\pi_{\text{old}}(a_t \mid \boldsymbol{s}_t)$. The ratio of the current policy probability to the reference is

$$\rho_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t \mid \boldsymbol{s}_t)}{\pi_{\text{old}}(a_t \mid \boldsymbol{s}_t)}. \tag{5.64}$$

Given the normalized advantage $\widetilde{A}_t$ and the ratio $\rho_t(\boldsymbol{\theta})$, the clipped policy objective is

$$\begin{aligned} L_{\pi}^{\text{RECAP}}(\boldsymbol{\theta}) = \mathbb{E}\Big[\min\Big(&\rho_t(\boldsymbol{\theta})\,\text{sg}(\widetilde{A}_t), \\ &\text{clip}(\rho_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\,\text{sg}(\widetilde{A}_t)\Big)\Big] \\ &+ \alpha_H \mathbb{E}\left[\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t))\right], \end{aligned} \tag{5.65}$$

where $\text{sg}(\cdot)$ denotes stop-gradient. The clipping prevents the policy from moving too far from π_{old} in a single cycle, and the entropy term encourages continued exploration. Policy updates are stopped early if the KL divergence to π_{old} exceeds a threshold.

The complete RECAP training flow is illustrated in Fig. 5.5, and the full update procedure is summarized in Algorithm 4.

The complete RECAP update corresponding to the flow in Fig. 5.5 is summarized in Algorithm 4.

Applying RECAP to the continuous body-velocity action space $\boldsymbol{a}_t = [v_t, \omega_t]^\top$ gives the RECAP-Continuous algorithm. Applying it to the discrete MoE coordinator, where the action is an expert index drawn from the finite set $\{1, \ldots, N_e\}$, gives the RECAP-MoE algorithm. Both instantiations share the same two-modification structure and differ only in how the expected value in Eq. (5.50) is computed and how the action log probability in Eq. (5.64) is evaluated.

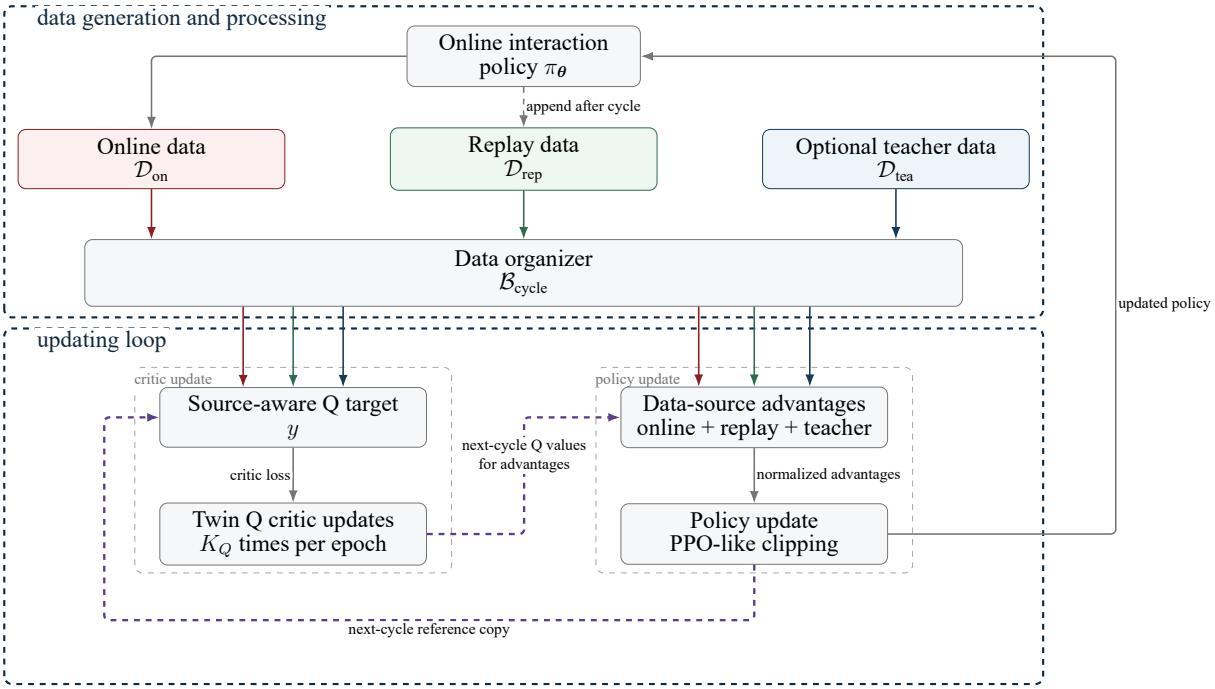


Figure 5.5. RECAP training flow. Online, replay, and teacher data are split between the Q-function branch (source-aware targets, twin-critic update) and the policy branch (source-aware advantages, clipped PPO update). Dashed arrows indicate quantities carried to the next cycle.

5.5 RRT* Waypoint Planning

RRT* generates a waypoint sequence for the learned local controller [12, 13]. Compared with the original RRT, RRT* adds parent selection and rewiring steps so that, under standard sampling assumptions, the path cost converges asymptotically toward the optimal path cost as the number of samples increases [13]. The planner operates in the planar free space

$$\mathcal{P}_{\text{free}} \subset \mathbb{R}^2. \tag{5.66}$$

Algorithm 4. RECAP algorithm

Initialize policy $\pi_{\boldsymbol{\theta}}$, Q functions $Q_{\boldsymbol{\phi}_1}, Q_{\boldsymbol{\phi}_2}$, and replay buffer $\mathcal{D}_{\text{rep}}$.
for cycle $k = 1, 2, \ldots$ **do**
    Freeze reference policy and target Q parameters using Eq. (5.54).
    Collect online rollouts $\mathcal{B}_{\text{on}}$ using $\pi_{\boldsymbol{\theta}}$.
    Sample replay batch $\mathcal{B}_{\text{rep}}$ from $\mathcal{D}_{\text{rep}}$; optionally sample teacher batch $\mathcal{B}_{\text{tea}}$.
    Compute reference value V_{old} using Eq. (5.55).
    Build critic targets using Eqs. (5.56)–(5.57).
    Build source-aware advantages using Eqs. (5.59)–(5.63).
    for epoch $e = 1, \ldots, E$ **do**
        Update the Q-critic parameters $\boldsymbol{\phi}_i$ by gradient descent on Eq. (5.58).
        Update the policy parameters $\boldsymbol{\theta}$ by gradient ascent on Eq. (5.65).
        Stop policy updates early if KL divergence to π_{old} exceeds threshold.
    end for
    Append $\mathcal{B}_{\text{on}}$ to replay buffer $\mathcal{D}_{\text{rep}}$.
end for

RRT* builds a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. At each iteration it samples a point, finds a nearby tree node, and steers toward the sample:

$$\boldsymbol{p}_{\text{new}} = \text{Steer}\left(\boldsymbol{p}_{\text{near}}, \boldsymbol{p}_{\text{rand}}, \Delta_{\text{rrt}}\right). \tag{5.67}$$

If the edge is collision-free, RRT* chooses the lowest-cost parent among nearby nodes,

$$\text{parent}(\boldsymbol{p}_{\text{new}}) = \arg\min_{\boldsymbol{p} \in \mathcal{N}(\boldsymbol{p}_{\text{new}})} \left[J(\boldsymbol{p}) + \|\boldsymbol{p}_{\text{new}} - \boldsymbol{p}\|_2\right], \tag{5.68}$$

and rewires neighboring nodes when the new node reduces their cost-to-come:

$$J(\boldsymbol{p}_{\text{new}}) + \|\boldsymbol{p} - \boldsymbol{p}_{\text{new}}\|_2 < J(\boldsymbol{p}). \tag{5.69}$$

The resulting path is converted into the waypoint sequence $\mathcal{W}$ used in Eq. (5.1). The RRT* waypoint construction procedure is summarized in Algorithm 5.

Algorithm 5. RRT* waypoint construction

Initialize $\mathcal{T} \leftarrow (\{\boldsymbol{p}_{\text{start}}\}, \emptyset)$.
for $n = 1, \ldots, N_{\text{rrt}}$ **do**
    With probability p_{goal} set $\boldsymbol{p}_{\text{rand}} \leftarrow \boldsymbol{p}_{\text{goal}}$; otherwise sample $\boldsymbol{p}_{\text{rand}}$ uniformly from $\mathcal{P}_{\text{free}}$.
    Find the nearest tree node $\boldsymbol{p}_{\text{near}}$.
    Compute $\boldsymbol{p}_{\text{new}} \leftarrow \text{Steer}(\boldsymbol{p}_{\text{near}}, \boldsymbol{p}_{\text{rand}}, \Delta_{\text{rrt}})$.
    if the edge $(\boldsymbol{p}_{\text{near}}, \boldsymbol{p}_{\text{new}})$ is collision-free **then**
        Find neighboring nodes $\mathcal{N}(\boldsymbol{p}_{\text{new}})$.
        Choose the lowest-cost parent using Eq. (5.68).
        Add $\boldsymbol{p}_{\text{new}}$ and its parent edge to $\mathcal{T}$.
        Rewire neighboring nodes when Eq. (5.69) is satisfied.
    end if
end for
Extract a path from $\boldsymbol{p}_{\text{start}}$ to $\boldsymbol{p}_{\text{goal}}$.
Return waypoint sequence $\mathcal{W} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K\}$.

After the waypoint sequence is constructed, the learned policy tracks the active waypoint and advances the waypoint index when the current target is reached. The complete tracking loop is summarized in Algorithm 6.

Algorithm 6. Waypoint tracking with RRT*

Compute an RRT* path using geometric obstacle information.
Extract waypoints $\mathcal{W} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K\}$.
Set active waypoint index $m \leftarrow 1$.
while the final goal has not been reached **do**
    Compute $\boldsymbol{g}_t$ from the active waypoint $\boldsymbol{w}_m$.
    Form state $\boldsymbol{s}_t$ from vision, proprioception, and $\boldsymbol{g}_t$.
    Select command $\boldsymbol{a}_t = [v_t, \omega_t]^\top$ using the learned policy.
    Execute the command through the gait generator.
    if $\|\boldsymbol{p}_t - \boldsymbol{w}_m\|_2 \leq \varepsilon_w$ **then**
        $m \leftarrow \min(m + 1, K)$.
    end if
end while

5.6 Summary

This chapter develops a learning-based navigation policy for legged robots that maps onboard visual and proprioceptive observations directly to body-velocity commands. A path planner supplies intermediate waypoints that break the long-horizon goal into short-horizon tracking tasks, allowing the learned policy to focus on local obstacle avoidance and terrain-adaptive locomotion without reasoning about global topology.

Two contributions improve the learning process. First, a mixture-of-experts architecture promotes more structured exploration, helping the optimizer escape local optima that a plain Gaussian policy tends to settle into. Second, RECAP improves sample efficiency by enabling off-policy learning from a replay buffer alongside online data, overcoming the data reuse limitation of standard PPO.

Chapter 6

Results and Analysis

This chapter evaluates the methods developed in Chapters 3–5. The preceding chapters introduced the kinematic model, the model-based locomotion method, and the learning-based navigation framework. The purpose of this chapter is to show how these components behave in navigation experiments and to analyze the strengths and limitations observed in practice. The four learned policies evaluated are proximal policy optimization (PPO) Continuous, PPO with mixture-of-experts (MoE), Replay-Enhanced Clipped Actor-Critic Policy Learning (RECAP) Continuous, and RECAP-MoE.

The results are organized in two parts. Section 6.1 focuses on controlled simulation experiments, including the problem setup, training behavior, and testing performance of the learned command policies. Section 6.2 discusses real-robot deployment and sim-to-real behavior.

6.1 Simulation Results

The simulation experiments evaluate whether the proposed framework can generate stable locomotion, track waypoint targets, and avoid obstacles using the visual/depth, proprioceptive

tive, and target-state inputs defined for learning-based navigation in Chapter 5. Simulation provides a repeatable setting for comparing PPO Continuous, PPO-MoE, RECAP Continuous, and RECAP-MoE under controlled task conditions.

6.1.1 Simulation Problem Setup

The training environment is a $30\,\mathrm{m} \times 30\,\mathrm{m}$ square workspace designed for controlled simulation rollout collection. High walls are placed around the workspace to define the navigable region and provide a consistent visual background. Figures 6.1 and 6.2 show the training world used for rollout collection.

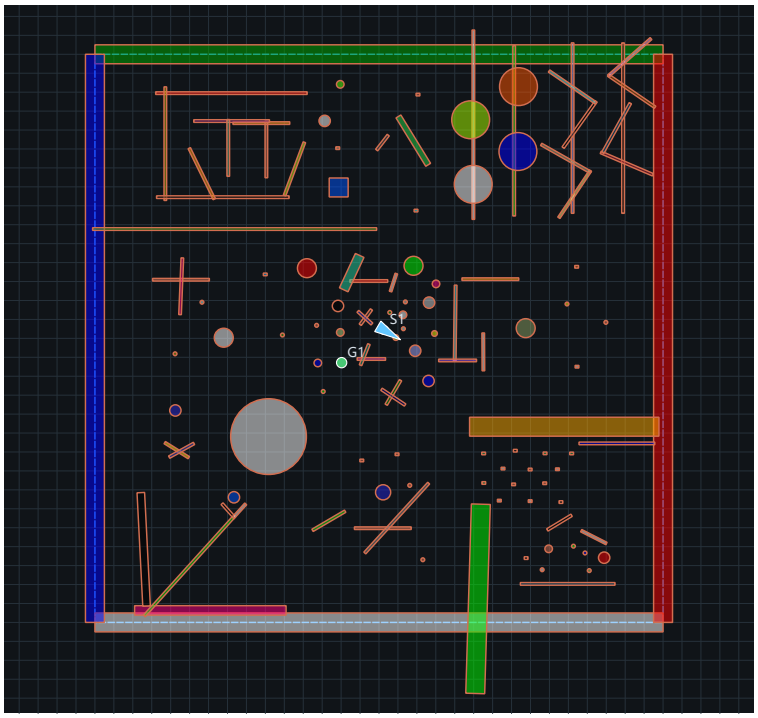


Figure 6.1. Training map layout. The environment uses a bounded $30\,\mathrm{m} \times 30\,\mathrm{m}$ workspace with high outer walls and an obstacle arrangement formed from rectangular and cylindrical primitives.

Obstacles are generated from two primitive shapes: rectangular boxes and cylinders. More complex local obstacle layouts can be formed by combining these two shape families.

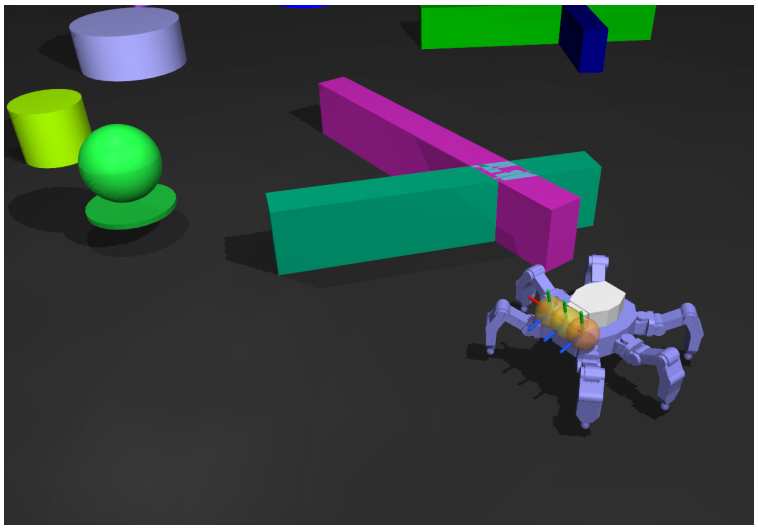


Figure 6.2. MuJoCo training environment visualization. The scene shows the hexapod robot and obstacle geometry used in the simulated rollout environment.

The obstacle colors are also varied, so the learned policy is encouraged to respond to obstacle geometry and spatial arrangement rather than becoming sensitive to a particular color pattern.

MuJoCo is used as the physics simulator [43]. The robot is defined by an XML model that specifies the kinematic structure, actuators, collision geometries, and sensors. Local STL files are used for the robot visual meshes, while contact and collision response are handled by the simulator through the defined collision geometries. The policy network receives a state composed of stereo RGB images, depth information, proprioceptive measurements, and target-state information. The visual observation is shown in Fig. 6.3: it consists of seven channels at 64×48 pixels each, comprising six RGB channels from the left and right stereo views respectively, plus the depth channel. The composite color images in the bottom row are for visual reference only. During rollout, the policy generates an action from the current state, the robot interacts with the simulated environment under this action, and the physics simulator returns the next state. This recursive interaction produces the trajectory used for training and evaluation.

Training trajectories are generated from randomly sampled start–goal tasks inside the valid workspace. For each rollout, the start position and goal position are sampled from obstacle-free regions, and the initial yaw angle is sampled from $[0, 2\pi)$. A sampled pose is accepted only when a clearance circle around the robot center does not intersect any obstacle. The start and goal are also constrained to be within $5\,\mathrm{m}$, so the sampled tasks remain achievable during training while still requiring meaningful waypoint-seeking and obstacle-avoidance behavior.

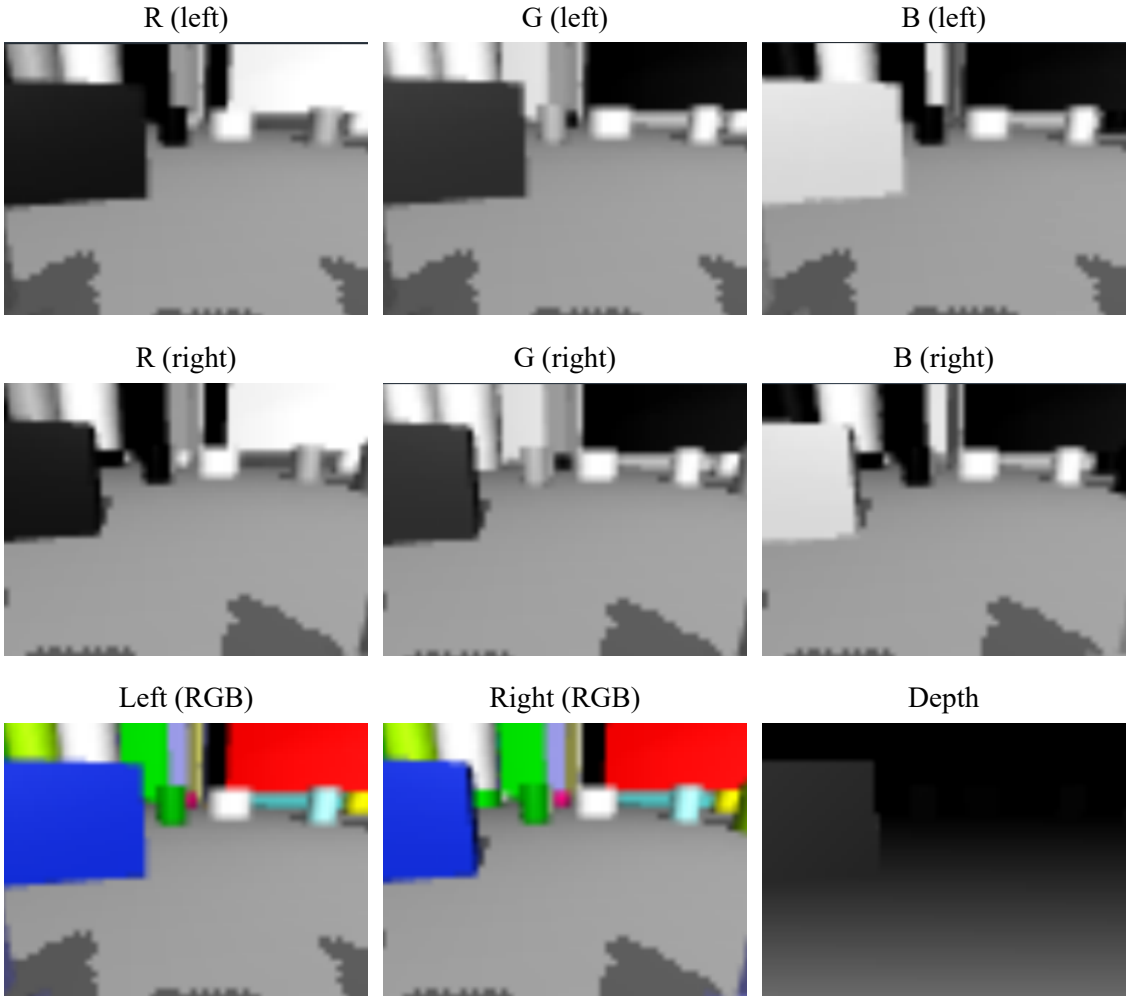


Figure 6.3. Visual observation at a single time step (64×48 pixels per channel). Rows 1–2 show the seven scalar network inputs: R, G, B channels of the left view $\boldsymbol{I}_t^{\mathrm{L}}$, right view $\boldsymbol{I}_t^{\mathrm{R}}$, and depth $\boldsymbol{D}_t$. Row 3 shows composite RGB images for visual reference only; not fed to the network.

6.1.2 Policy Training Behavior

The four learned policies are trained using repeated rollout-and-update iterations. Each iteration collects online trajectories from the current policy and then updates the corresponding actor and critic networks. Table 6.1 summarizes the rollout, replay, and teacher-data settings used for the four learned policies.

Table 6.1. Training data settings for the four learned policies.

Method	Online eps.	Max steps	Replay traj.	Teacher traj.	Teacher pool
PPO Continuous	4	256	0	0	–
PPO-MoE	4	256	0	0	–
RECAP Continuous	4	256	up to 16	4	12
RECAP-MoE	4	256	up to 16	4	12

For the RECAP runs, the replay entry denotes the target number of replay trajectories used after the replay pool is populated; the first iterations use fewer replay trajectories while the pool is being filled. The teacher pool consists of human-demonstrated trajectories generated by manually controlling the robot to navigate around obstacles.

The resulting training behavior is shown in Fig. 6.4. The horizontal axis is the training iteration index. For readability, the dark curves show a 10-iteration moving average, while the lighter curves show the raw values.

From Fig. 6.4(a), we first compare the return behavior of the continuous policies. In the early iterations, PPO Continuous performs slightly better than RECAP Continuous. After the initial phase, however, RECAP Continuous clearly surpasses PPO Continuous and reaches the near-optimal return region earlier. One possible explanation is that replay and human-demonstrated teacher trajectories provide additional useful samples after the initial stage, so the policy may benefit from sample reuse instead of relying only on newly collected on-policy rollouts.

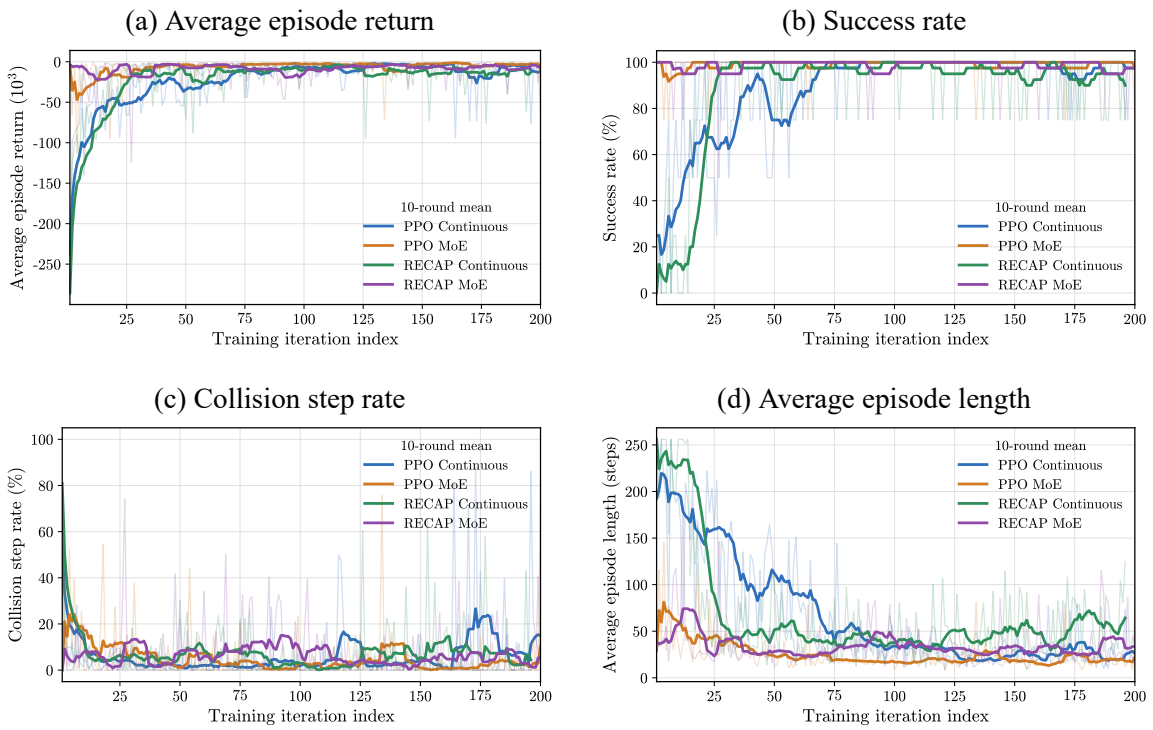


Figure 6.4. Training curves of the four learned navigation policies. PPO Continuous, PPO-MoE, RECAP Continuous, and RECAP-MoE are compared using average episode return, success rate, collision step rate, and average episode length over the first 200 training iterations.

The MoE policies show a different early-training behavior. Both PPO-MoE and RECAP-MoE begin from a better return level than the continuous policies because the expert library already contains meaningful command behaviors. The coordinator therefore does not need to discover all useful motion modes from scratch, which makes early training easier. Compared with the continuous policies, the MoE policies approach the high-return region in fewer iterations. By the end of the first 200 iterations, all four methods reach a similar high-return range, indicating that all four learned policies are functional, while their convergence speed and early training behavior are different.

The success-rate curves in Fig. 6.4(b) support the same interpretation. PPO-MoE and RECAP-MoE reach high success rates earlier than the continuous policies, and the RECAP-

based methods remain competitive with their PPO counterparts. This indicates that the structured expert selection and replay-enhanced updates are both useful for improving waypoint-reaching behavior during training.

Figures 6.4(c) and 6.4(d) also provide evidence for safety and efficiency. The collision step rate is noisier than the return and success curves because it depends strongly on local obstacle geometry and sampled start–goal pairs, but the smoothed curves generally decrease after the early training phase. The average episode length also decreases as the policies learn to reach the target using fewer command cycles. Together, the four plots show that the learned policies improve not only in final success rate, but also in convergence speed, collision tendency, and navigation efficiency.

6.1.3 Testing Results and Analysis

The testing environment uses obstacle styles similar to those in training, but the scene is different from the training world. Compared with the $30\,\mathrm{m} \times 30\,\mathrm{m}$ training world, the testing map is a smaller $10\,\mathrm{m} \times 10\,\mathrm{m}$ bounded workspace with rectangular and circular obstacles. The obstacle arrangement, individual obstacle shapes, and obstacle colors are different from the training cases. This evaluates whether the learned controllers can transfer to a related but unseen scene rather than only memorizing the training layout. The start and goal poses are randomly sampled in the safe region and checked before evaluation. Random sampling produces tasks with different levels of difficulty, which gives a broader evaluation of navigation performance. After the task set is generated, the same start–goal pairs are fixed and reused across all learned controllers for fair comparison. Figure 6.5 shows the testing scene and 10 sampled start–goal pairs used as navigation tasks. In the waypoint-guided setting, the planned waypoint sequence is

also fixed for each task so that the comparison mainly reflects command selection and waypoint tracking behavior.



Figure 6.5. Representative testing scene and start–goal pairs. Triangular markers indicate initial robot poses and heading directions; circular markers indicate goal positions. Obstacle colors are varied to test policy robustness to visual appearance changes.

Quantitative Metrics.

After training, the learned controllers are evaluated on testing navigation episodes under two evaluation settings. In the direct setting, the target-state input is always computed from the final goal. In the waypoint-guided setting, a rapidly-exploring random tree with rewiring (RRT*) planner first generates a waypoint sequence, and the robot tracks the waypoints one by one in order. Evaluation uses the same stochastic policy form as training, so the reported results also reflect robustness under policy randomness. For the current evaluation, each of the 10 start–goal pairs is repeated 5 times for each controller and each evaluation setting to reduce

variance and provide a more reliable comparison. The quantitative comparison includes PPO Continuous, PPO-MoE, RECAP Continuous, and RECAP-MoE, as summarized in Tables 6.2 and 6.3. Units are shown in the metric names. Bold entries indicate the best value in each row; ties are bolded.

Table 6.2. Testing metrics in the direct-goal setting.

Metric	PPO Cont.	PPO-MoE	RECAP Cont.	RECAP-MoE
Success rate (%)	**100.0**	**100.0**	**100.0**	**100.0**
Collision episode rate (%)	**22.0**	28.0	24.0	30.0
Collision step rate (%)	**2.87**	2.97	3.68	4.49
Horizon use (%)	15.25	**12.27**	22.32	20.55
Mean path length (m)	6.57	5.75	5.91	**5.58**
Mean clearance (m)	0.17	0.10	**0.20**	0.18
Cost (10^3)	13.84	**11.32**	18.92	18.64

Table 6.3. Testing metrics in the waypoint-guided setting.

Metric	PPO Cont.	PPO-MoE	RECAP Cont.	RECAP-MoE
Success rate (%)	**100.0**	**100.0**	**100.0**	**100.0**
Collision episode rate (%)	12.0	16.0	12.0	**4.0**
Collision step rate (%)	0.90	1.56	0.34	**0.28**
Horizon use (%)	**13.41**	14.00	28.71	32.05
Mean path length (m)	6.71	6.35	6.58	**6.02**
Mean clearance (m)	0.23	0.22	0.22	**0.29**
Cost (10^3)	**10.15**	11.03	22.51	23.65

The most important result is that all four learned controllers reach the goal in all sampled testing cases. Both the direct-goal and waypoint-guided settings achieve a 100% success rate for every method, which shows that the learned navigation policies can complete the assigned tasks within the required horizon. The remaining metrics therefore describe how the methods reach the goal, rather than whether they can reach it.

Different policy structures lead to different motion characteristics. The RECAP variants generally use a larger fraction of the available horizon, especially in the waypoint-guided setting,

which indicates a more cautious style of motion. This additional horizon use can be associated with shorter or safer motion in some cases; for example, RECAP-MoE gives the shortest mean path length in both settings. The MoE structure also tends to shorten the path relative to the corresponding continuous policy. These results suggest that the structured expert representation changes the navigation behavior, but not every metric is improved by the same method.

The clearest effect of waypoint guidance is on obstacle interaction. Compared with direct goal tracking, the waypoint-guided setting reduces both collision episode rate and collision step rate for all controllers. The mean clearance also increases for all methods, showing that the planned waypoints provide safer intermediate targets around obstacles. This improvement does not necessarily reduce path length, horizon use, or accumulated cost in every case. Instead, the main role of planning in these experiments is to provide safer route-level guidance, while the learned controller still determines the local motion used to follow the route.

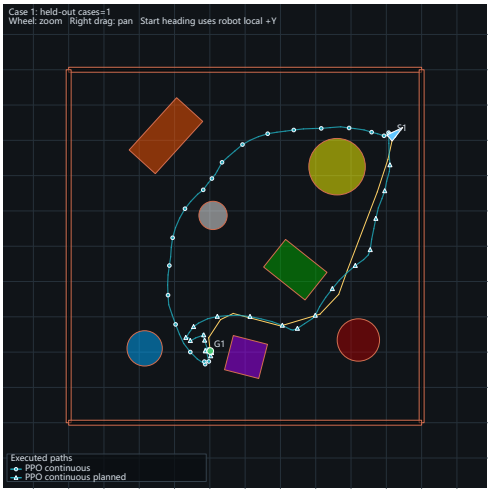
Qualitative Trajectory Analysis.

Representative trajectory plots provide qualitative evidence that is not fully captured by aggregate metrics. Figure 6.6 shows one representative testing case with a moderate obstacle-avoidance requirement. The same start–goal pair and obstacle arrangement are used for all four learned controllers, and each subplot compares direct goal tracking with waypoint-guided tracking.

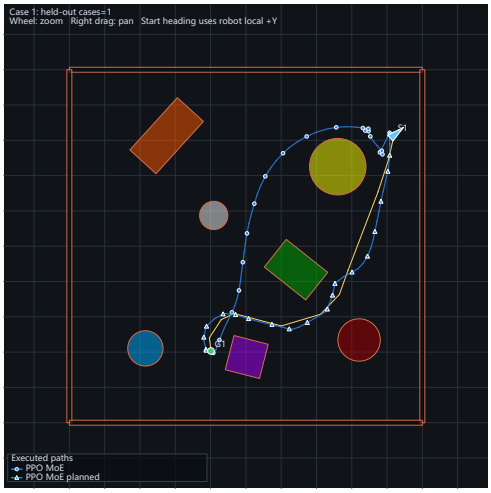
The trajectories in Fig. 6.6 show that all four learned controllers complete the representative task in both evaluation settings. The executed paths remain generally safe and goal-directed. The RECAP-MoE rollout contains a small contact event, as indicated by the red-highlighted segment, but the robot quickly recovers and continues toward the goal rather than becoming stuck

at the obstacle. The MoE-based policies also show slightly shorter executed paths in this case, which is consistent with the shorter mean path length observed for the MoE variants in the quantitative evaluation.

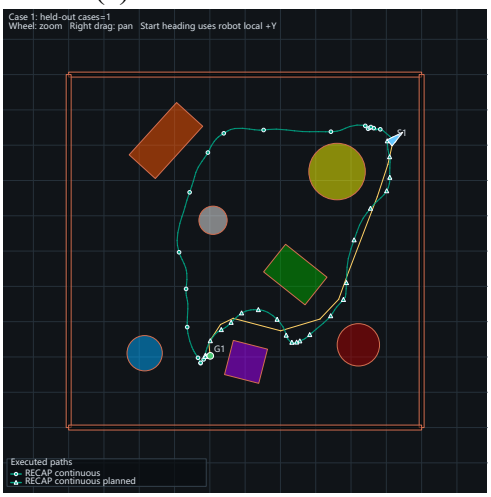
(a) PPO Continuous



(b) PPO-MoE



(c) RECAP Continuous



(d) RECAP-MoE



Figure 6.6. Qualitative trajectory comparison on a testing map. Each subplot overlays direct (circular markers) and waypoint-guided (triangular markers) executions for the same start–goal pair; the yellow curve shows the planned route. Red highlighting indicates contact with an obstacle.

Expert Selection.

For the MoE policies, the selected expert provides an additional diagnostic signal. Figures 6.7 and 6.8 show the expert probability distribution, the sampled expert at each trajectory step, and the total expert-selection share for the representative qualitative rollouts.

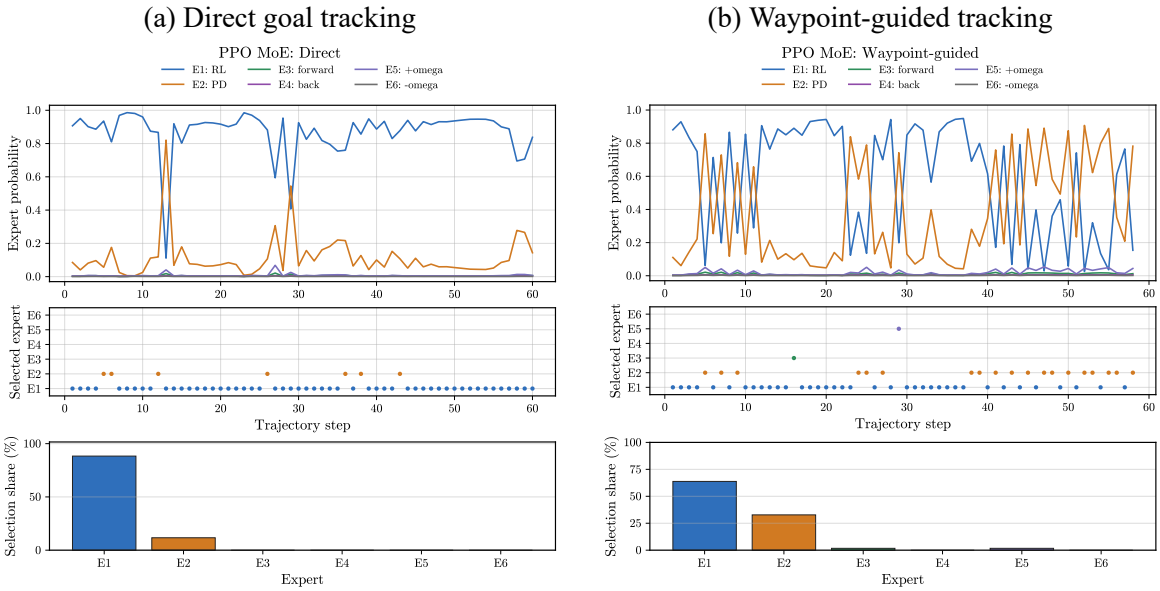


Figure 6.7. Expert-selection diagnostics for PPO-MoE. Each subplot shows the expert probabilities, the actually selected expert at each trajectory step, and the percentage of steps assigned to each expert.

The expert-selection plots reveal a useful difference between the two MoE-based algorithms. In PPO-MoE, the selected experts are concentrated mainly on the two broadly useful experts: the learned continuous-policy expert and the PD tracking expert. This behavior is reasonable because both experts already provide general goal-reaching ability. The learned continuous expert can handle a wide range of navigation states, while the PD expert is effective when the local target can be approached more directly. Thus, the PPO-MoE behavior is not unsuccessful; rather, it shows a tendency to rely on general-purpose experts instead of strongly using the full expert library in this representative rollout.

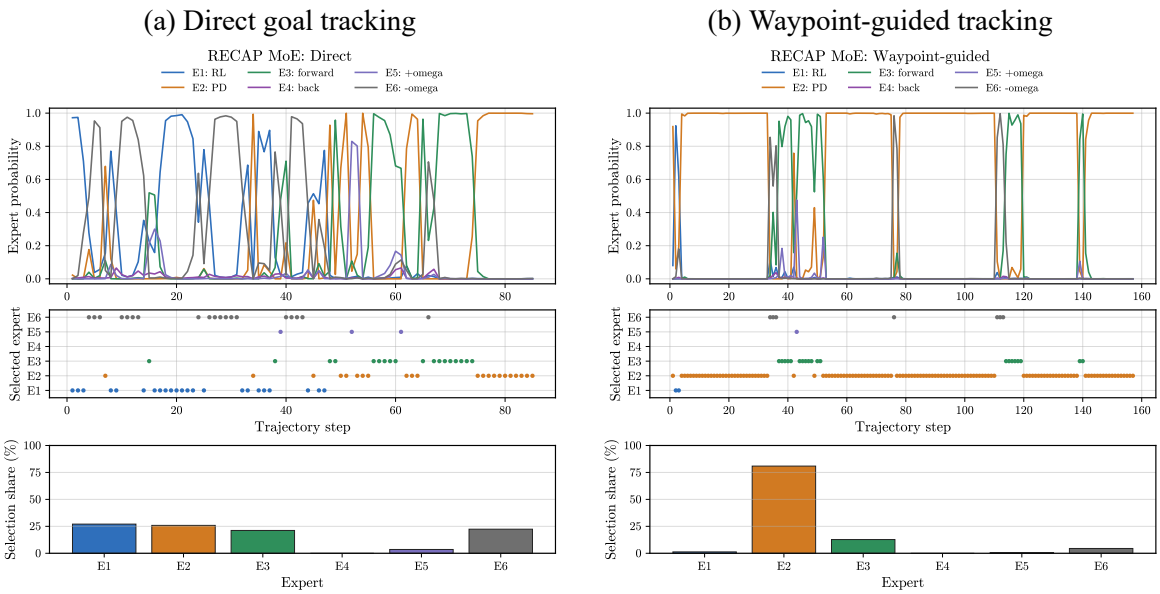


Figure 6.8. Expert-selection diagnostics for RECAP-MoE. Each subplot shows the categorical policy distribution, sampled expert decisions per step, and expert usage share over the trajectory.

RECAP-MoE shows a different usage pattern. The coordinator calls a wider range of specialized experts, including forward and turning behaviors, and these selections are often made with high probability when they are used. This suggests that the replay and teacher-demonstration data help preserve situation-dependent expert usage. Some experts may be selected only for short parts of the trajectory, but their selection is interpretable: they are used when a specific motion primitive is useful for the local navigation situation. This makes the learned behavior more transparent without preventing successful goal reaching.

The comparison between direct and waypoint-guided execution also has a clear interpretation. With waypoint guidance, the PD expert is selected more often for both MoE methods. This is expected because the planned waypoints divide the global task into simpler local tracking segments. When the local target is easier to approach, the PD tracking expert becomes a more efficient and reasonable choice. Overall, the expert-selection diagnostics support the role of the

MoE structure as an interpretable command-selection mechanism, and they show how teacher data and waypoint guidance can influence the internal policy behavior.

6.2 Real-Robot Deployment

The real-robot experiments evaluate whether the proposed method can be executed on the physical hexapod platform. Transferring policies trained in simulation to physical hardware is a known challenge in robot learning [44, 45, 46]. The current deployment addresses this by using a simulator-assisted observation channel rather than direct policy transfer: a motion-capture system measures the poses of the robot and obstacles, these poses are used to reconstruct the scene in simulation, and the policy input is generated from the reconstructed scene. In this way, the robot motion, actuation, and state transition occur in the real world, while the observation processing is made more controlled through motion capture and simulation.

6.2.1 Deployment Setup

The deployment tests are conducted on a bounded flat laboratory floor with an effective area of $6\,\mathrm{m} \times 4\,\mathrm{m}$. Because the available indoor test region is limited, the deployment scene uses two cubic obstacles rather than a dense obstacle field. Each obstacle has a known side length l_{block}, approximately $0.7\,\mathrm{m}$ in the current setup. For safety, the corresponding obstacle geometry used inside the reconstructed MuJoCo scene is enlarged to an edge length of $0.8\,\mathrm{m}$, so the policy reacts to a more conservative obstacle model than the physical block alone.

Motion-capture markers are attached to both the robot and the obstacles. During deployment, the measured robot and obstacle poses are used to reconstruct the scene in MuJoCo. The reconstructed scene then provides the visual/depth feedback $\boldsymbol{\zeta}_t$, proprioceptive information $\boldsymbol{\eta}_t$,

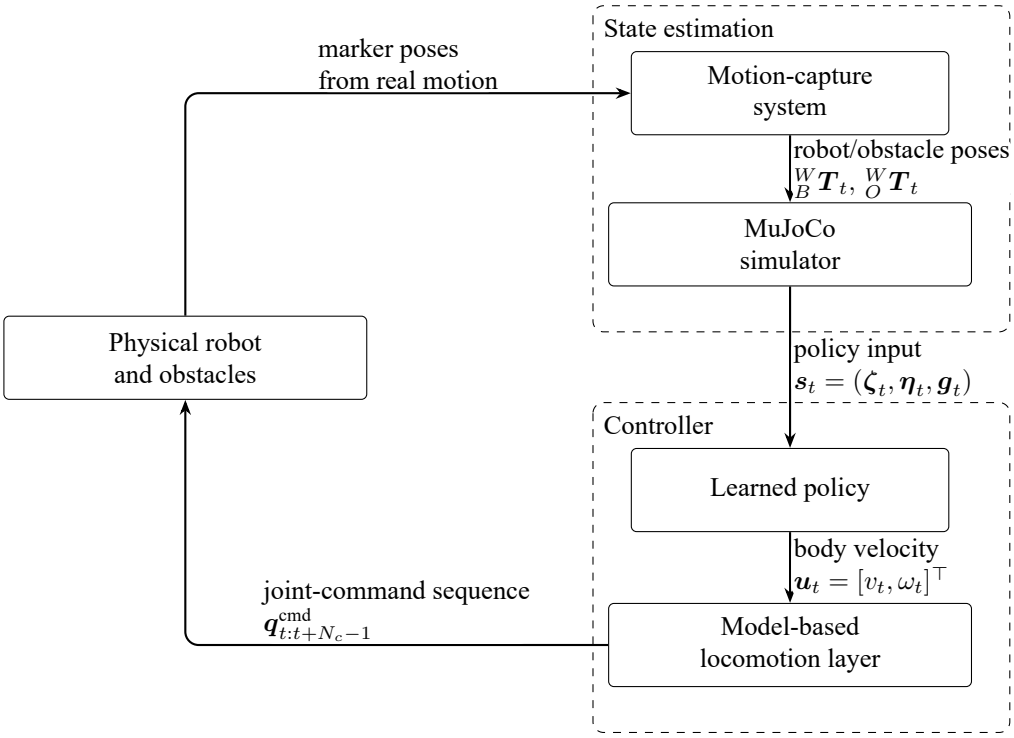


Figure 6.9. Motion-capture-assisted real-robot deployment loop. Motion capture measures robot and obstacle poses; the scene is reconstructed in MuJoCo to generate policy inputs $\boldsymbol{\zeta}_t$ (visual/depth), $\boldsymbol{\eta}_t$ (proprioceptive), and $\boldsymbol{g}_t$ (target state); the controller returns joint-command sequences to the robot.

and robot-centered target-state information $\boldsymbol{g}_t$ used by the policy. The computed command is sent to the physical robot, and the resulting real motion is measured again by the motion-capture system. This closed feedback process is the deployment loop shown in Fig. 6.9.

6.2.2 Deployment Behavior

The real-robot deployment uses the RECAP-MoE policy, which is the most advanced learned policy considered in this thesis. The purpose of this part is therefore not to repeat the full algorithm comparison from the simulation study, but to examine whether the final learned policy can be executed with the physical robot and the motion-capture-assisted observation loop.

Static Obstacle Case.

In the static-obstacle case, the robot passes around the two fixed blocks and reaches the goal region, as shown in Fig. 6.10. The reconstructed simulation view contains a short red segment, which indicates contact with the enlarged obstacle model used for conservative perception. In the physical scene, however, the robot keeps a small clearance from the actual block and does not visibly hit the obstacle. This result shows safe goal-reaching behavior under the motion-capture-assisted feedback setting.

Waypoint-Guided Case.

In the waypoint-guided case, the robot follows the planned route and passes the obstacles smoothly, as shown in Fig. 6.11. The executed trajectory does not pass through every intermediate waypoint strictly. This is an expected behavior because intermediate waypoints are used as route guidance with a looser acceptance condition than the final goal-reaching criterion. The deployment result therefore demonstrates that the planned waypoints provide route guidance while the local policy still determines the executed motion.

Moving Obstacle Case.

The moving-obstacle case shows the response to an online change in the scene. At the beginning of the trial, the robot has a relatively direct route toward the goal. During execution, an obstacle is moved into this direct path, and the updated motion-capture measurement changes the reconstructed observation. As shown in Fig. 6.12, the robot reacts to the changed obstacle location, adjusts its route, and continues toward the goal rather than following the original direct segment. The robot can respond to a moving obstacle through repeated sensing, reconstruction, policy evaluation, and physical execution.

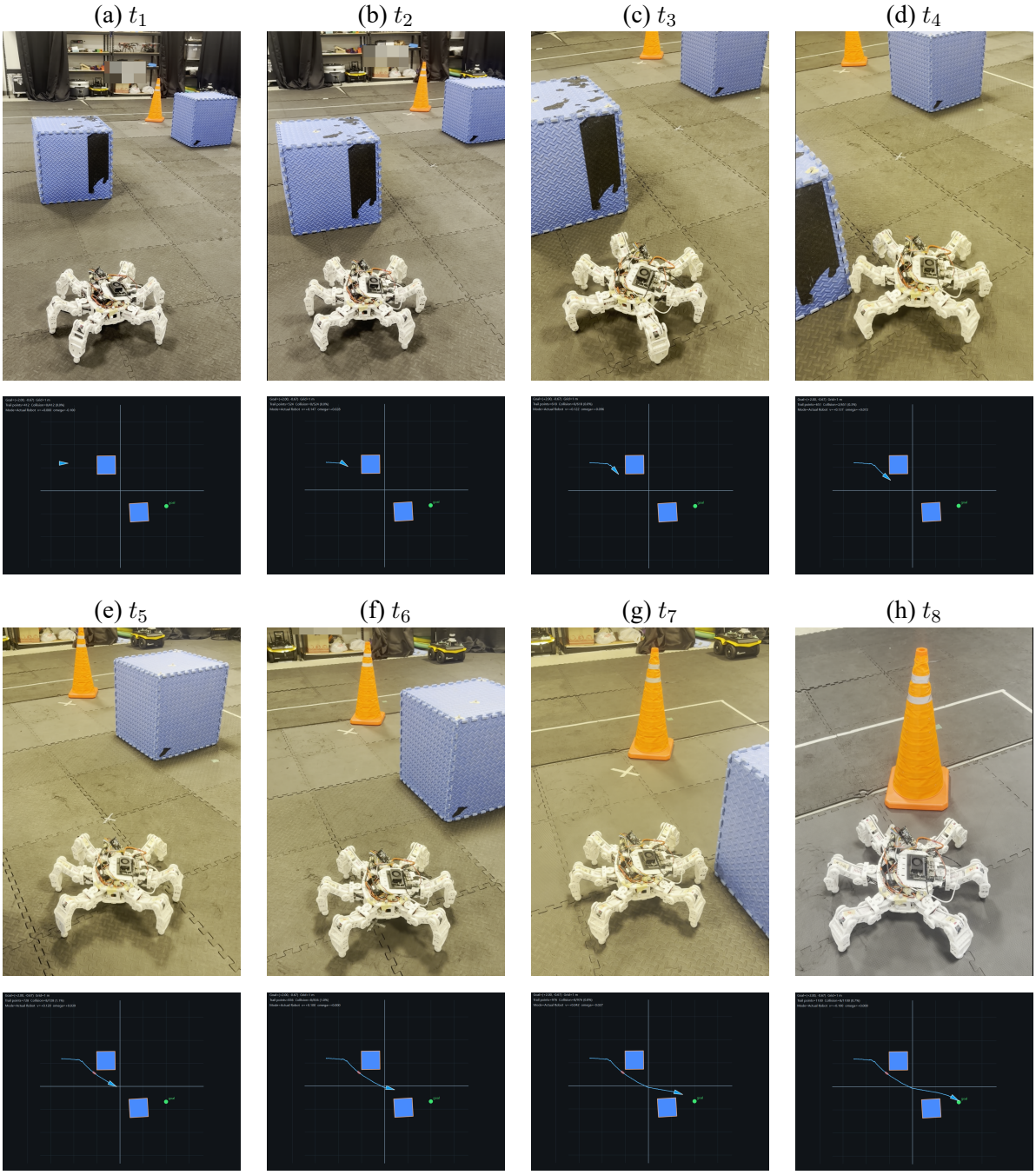


Figure 6.10. Static-obstacle deployment sequence. Each column pairs the physical robot snapshot (top) with the MuJoCo reconstruction used for policy observation (bottom). Blue boxes are obstacles, the orange cone is the goal, and red trajectory segments indicate contact.

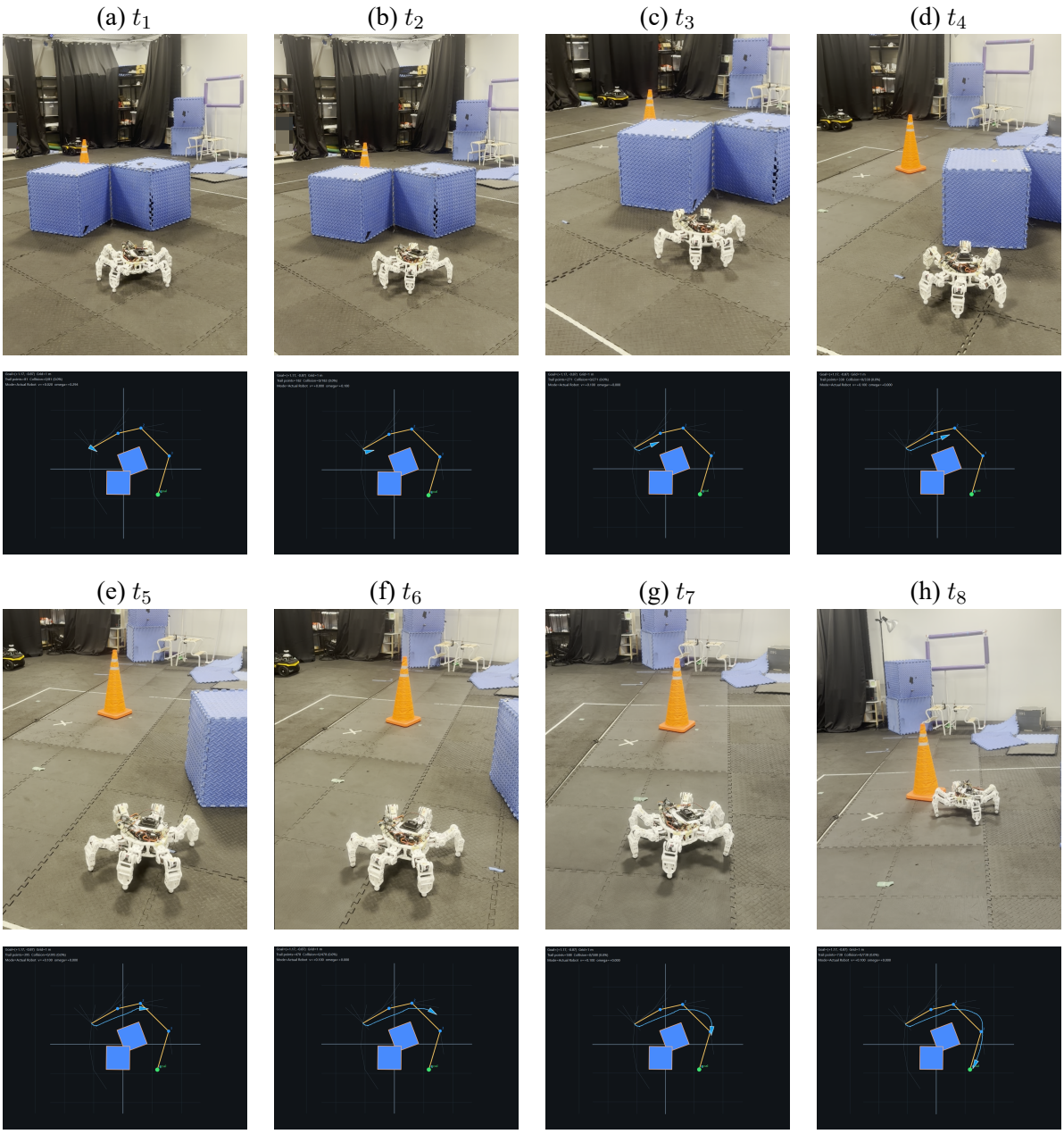


Figure 6.11. Representative waypoint-guided deployment sequence. For each time step, the physical robot snapshot is paired with the corresponding reconstructed scene used for waypoint-conditioned policy execution.

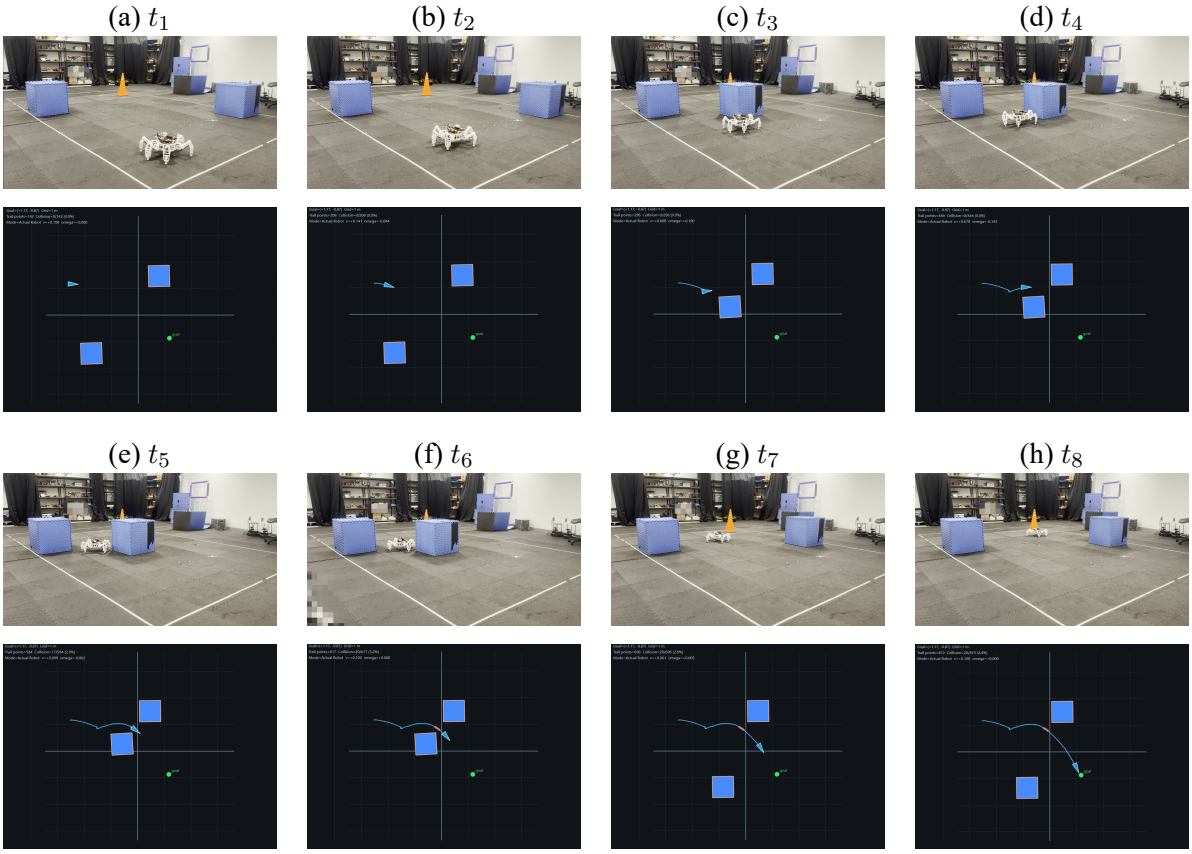


Figure 6.12. Representative moving-obstacle deployment sequence. For each time step, the physical robot snapshot is placed above the corresponding reconstructed scene used by the policy observation and command update.

6.3 Summary and Analysis

All four learned policies achieve a 100% success rate in the testing environment, confirming that local visual, depth, and proprioceptive feedback can be mapped to compact body commands that produce goal-directed hexapod navigation. The remaining metrics describe the quality of navigation rather than its feasibility.

The MoE structure provides better initial performance and more structured exploration: the expert library gives the coordinator meaningful action choices from the start, rather than requiring motion modes to be discovered from scratch. Faster convergence follows from this. The expert-selection diagnostics support this interpretation — RECAP-MoE activates a wider, more situation-dependent range of experts compared with PPO-MoE, which concentrates on two general-purpose ones.

RECAP reaches the high-return region earlier than PPO in both variants, reflecting the sample efficiency gained from replay and teacher-demonstration data. RECAP-MoE combines both benefits and achieves the strongest collision and clearance metrics in the waypoint-guided setting.

Waypoint guidance from RRT* consistently reduces collision rates and increases clearance across all methods. Its role is safer route-level guidance rather than path shortening; the planner improves safety while the learned controller determines the executed motion along the route.

The real-robot deployment confirms physical feasibility across three cases: static-obstacle goal-reaching, waypoint-guided routing, and reactive rerouting when an obstacle is moved mid-

trial. The remaining limitation is the mocap-assisted observation channel; onboard camera deployment is the natural next step.

Chapter 7

Conclusion and Future Work

7.1 Summary and Engineering Significance

This thesis developed a framework that combines navigation-level decision making with locomotion-level execution in an efficient and modular form. The key value of this structure is that the navigation policy and the locomotion generator do not need to be learned as one large monolithic policy. Instead, each part can be designed, trained, or adapted within its own role, while the overall system remains connected through a compact command interface.

For hexapod robots, this organization has clear engineering significance. Their multiple legs provide stability, redundancy, and terrain-traversal potential, which makes them suitable candidates for unstructured and hazardous environments. A framework that connects sensing, navigation decisions, command generation, and locomotion is therefore meaningful for scenarios such as outdoor field navigation, forests, damaged buildings, disaster-response ruins, narrow passages, industrial inspection, and search-and-rescue tasks. The work in this thesis provides a practical foundation for these hexapod navigation scenarios, while future improvements can further strengthen robustness and field applicability.

Although the implementation in this thesis focuses on a hexapod platform, the same de-

sign principle can also be useful beyond this specific robot. Other legged robots, and more generally robotic systems whose motion requires coordination among many joints or dynamic components, can benefit from separating navigation decisions from movement execution. This separation reduces coupling between modules and keeps the system design interpretable, reusable, and easier to adapt.

7.2 Limitations

The first limitation is open-loop locomotion. The current locomotion layer can generate coordinated gait motion, but it does not yet adapt foot placement, gait timing, swing trajectories, stance timing, or gait transitions online according to detailed terrain feedback. The current motion set also does not fully exploit the mobility of the hexapod, such as sideways walking, body shifting, adaptive support polygons, and terrain-dependent gait patterns. In addition, the current locomotion model is closer to quasi-static or quasi-kinematic modeling than to full dynamic locomotion.

The second limitation is environmental perception. The current system mainly relies on visual and depth information together with proprioception. This is useful for the present experiments, but a limited forward sensing field may miss side obstacles, rear obstacles, occluded structures, or complex surrounding geometry. This becomes important for forests, damaged buildings, narrow passages, and other cluttered environments.

The third limitation is the memoryless policy representation. The policy mainly uses the current observation and does not explicitly preserve previously observed terrain, obstacles, contact behavior, or body-motion history. This can limit navigation in partially observable settings

where past information remains relevant after leaving the current sensor view.

7.3 Future Work

The first future direction is closed-loop and terrain-adaptive locomotion. Future work should introduce an online adaptation layer that adjusts foot placement, gait timing, swing trajectories, stance selection, and gait transitions according to robot state and terrain information. This would allow the robot to handle slopes, bumps, holes, uneven ground, foothold constraints, and damaged environments. More complete dynamics, including contact forces, actuator effects, body inertia, and dynamic stability, would further improve robustness.

The second direction is multi-modal perception for real-world navigation. Future systems can combine camera images, depth estimates, LiDAR, proprioception, and contact feedback to obtain a richer description of the surrounding environment. This would improve obstacle understanding, terrain assessment, foot-placement selection, and navigation reliability in unstructured or hazardous environments.

The third direction is memory-based policy learning. Frame stacking can provide short-term temporal context, while more structured memory can be introduced through recurrent neural networks (RNNs) [47], gated recurrent units (GRUs) [48], or long short-term memory (LSTM) networks [49]. Memory would allow the robot to retain information about recently observed obstacles, terrain boundaries, body motion, and contact patterns, leading to more consistent navigation and locomotion decisions.

Overall, this thesis provides a practical foundation for learning-based hexapod waypoint navigation connected with model-based locomotion. Future improvements in terrain-adaptive

locomotion, multi-modal perception, and memory-based learning can further move the framework toward robust engineering deployment in complex real-world environments.

Appendix A

Supplementary Hardware Design

A.1 Hardware Modules for Real-Robot Deployment

This appendix summarizes the hardware modules used in the real-robot deployment platform. execution.

Table A.1. Hardware modules used in the real-robot deployment platform.

Module class	Module	Role
Power supply	Logic power supply	Dedicated battery supply for the logic-side electronics, including the onboard computing and control electronics.
	Actuator power supply	Dedicated battery supply for the servo-actuation side, separated from the logic supply to reduce electrical loading on the computation and control modules.
Power regulation	Voltage conversion	Step-up and step-down converter modules regulate the battery voltages to the levels required by the onboard computer, controller electronics, servo drivers, and servo actuators.
Actuator control	Servo driver boards	Two servo driver boards distribute command signals to the eighteen actuators. Each board drives nine servos, which reduces the electrical and signal burden on a single driver board.
Actuation	Servo actuator set	The robot uses eighteen servos, with three actuators on each leg. The root and distal joints use $20\,\mathrm{kg}$-class servos, while the middle joint uses a $40\,\mathrm{kg}$-class servo because this joint empirically carries the largest torque demand.
Computation	Onboard computer	Jetson onboard computer used as the robot-side computation interface for receiving commands and coordinating communication with the lower-level electronics.
	External PC	External PC used for policy inference, MuJoCo scene reconstruction, motion-capture communication, and experiment monitoring. The PC connects to the Jetson through a remote SSH interface during deployment.

References

[1] Marc H. Raibert. *Legged Robots That Balance*. MIT Press, Cambridge, MA, 1986.

[2] R. McN. Alexander. The gaits of bipedal and quadrupedal animals. *The International Journal of Robotics Research*, 3(2):49–59, 1984.

[3] Robert B. McGhee and Geoffrey I. Iswandhi. Adaptive locomotion of a multilegged robot over rough terrain. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(4):176–182, 1979.

[4] Rodney A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural Computation*, 1(2):253–262, 1989.

[5] Uluc Saranli, Martin Buehler, and Daniel E. Koditschek. RHex: A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7):616–631, 2001.

[6] Holk Cruse. What mechanisms coordinate leg movement in walking arthropods? *Trends in Neurosciences*, 13(1):15–21, 1990.

[7] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642–653, 2008.

[8] Daniel E. Koditschek, Robert J. Full, and Martin Buehler. Mechanical aspects of legged locomotion control. *Arthropod Structure & Development*, 33(3):251–272, 2004.

[9] Zipeng Fu, Ashish Kumar, Ananye Agarwal, Haozhi Qi, Jitendra Malik, and Deepak Pathak. Coupling vision and proprioception for navigation of legged robots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[10] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[12] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998.

[13] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[14] Junhyeok Ahn, Orion Campbell, Donghyun Kim, and Luis Sentis. Fast kinodynamic bipedal locomotion planning with moving obstacles. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[15] Ziwon Yoon, Lawrence Y. Zhu, Jingxi Lu, Lu Gan, and Ye Zhao. STATE-NAV: Stability-aware traversability estimation for bipedal navigation on rough terrain. *arXiv preprint arXiv:2506.01046*, 2025.

[16] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, Upper Saddle River, NJ, 3 edition, 2005.

[17] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[18] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine Dario Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, 2018.

[19] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the MIT cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2018.

[20] Zhaoyuan Gu, Yuntian Zhao, Yipu Chen, Rongming Guo, Jennifer K. Leestma, Gregory S. Sawicki, and Ye Zhao. Robust-locomotion-by-logic: Perturbation-resilient bipedal locomotion via signal temporal logic guided model predictive control. *IEEE Transactions on Robotics*, 2025.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

[23] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[24] Xiaoyu Huang, Yufeng Chi, Ruofeng Wang, Zhongyu Li, Xue Bin Peng, Sophia Shao, Borivoje Nikolic, and Koushil Sreenath. DiffuseLoco: Real-time legged locomotion control with diffusion from offline datasets. In *Proceedings of The 8th Conference on Robot Learning (CoRL)*, 2025.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[26] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[27] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4):1 – 14, 2018.

[30] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. AMP: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics*, 40(4):1–20, 2021.

[31] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[32] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020.

[33] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[34] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[35] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.

[36] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.

[37] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, Cham, Switzerland, 2 edition, 2016.

[38] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[39] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 2018.

[40] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[41] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

[42] Hado van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, volume 23, 2010.

[43] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[44] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation*, pages 1–8. IEEE, 2018.

[45] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30. IEEE, 2017.

[46] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018.

[47] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[48] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.