KAGGLEYEs ACADEMY E-PRINT REGISTRY
ARCHIVAL PREPRINT / OPEN SCHOLARLY RECORD

KAGG RECORD
KAGG:2607.00008

Configuration Abstractions for Heterogeneous Legged Robot Fleets

Wenjie Zhao • **Mingxuan Li** • **Boyang Sun**

Intelligent Mobility Lab, University of Science and Technology of China, Hefei, China

PUBLISHED	REGISTERED	RECORD TYPE
July 27, 2026	August 1, 2026	Preprint

ABSTRACT

Heterogeneous legged robot fleets combine platforms that differ in body length, payload, locomotion mode, and sensor mounting, and these differences normally force each embodiment onto its own local planner. This fragmentation duplicates code, complicates validation, and allows planning behavior to drift between robots that share a mission. This paper extends the configuration-driven planning idea of AgniNav from a single robot to the fleet level. The proposed abstraction has three parts: a compact, typed configuration schema that separates collision geometry, kinematic and payload limits, locomotion class, and sensor rig; a configuration registry service that validates descriptors, issues content-addressed handles, and caches derived planning quantities; and a planner interface that accepts configuration handles instead of compiled-in robot parameters. One bit-identical planning binary was evaluated on a simulated fleet of four morphologies, namely Go2-class and B2-class quadrupeds, a 1.4 m humanoid, and a four-wheel-drive rover, each with a distinct sensor rig, across 15 mixed-fleet scenarios with 5 seeds each. Compared with four separate per-embodiment code paths, the shared planner raised shared module reuse from 83.2 to 87.2 percent, lowered configuration load latency from 1.9 to 1.8 ms and 95th-percentile replanning latency from 38 to 34 ms, and raised plan quality parity from 81.2 to 96.4 percent, whereas a shared planner driven by ad hoc configuration flags reached 88.2 percent. Ablations attribute the largest share of the gain to the sensor-rig abstraction. The remaining failures involve differential-drive behavior on steep slopes and registry cold starts under concurrent load.

INDEX TERMS heterogeneous robot fleets, legged robots, cross-embodiment navigation, local planning, configuration management, robot software architecture

KAGG NOTE A single local planner can serve physically different robots when every embodiment-dependent quantity enters through a validated, versioned configuration handle rather than through code paths or loosely typed flags.

01 Introduction

Operators of legged robots increasingly deploy mixed fleets rather than a single platform. A facility inspection team may combine compact quadrupeds that pass through narrow corridors, larger quadrupeds that carry heavier sensor payloads, humanoids that can operate in spaces designed for people, and wheeled rovers that cover long, flat distances efficiently. Each platform is attractive precisely because it is physically different, yet the same differences make it hard to share navigation software. Body length and width determine which gaps are traversable; payload mass changes attainable speed and braking distance; locomotion class determines whether lateral motion is available; and the sensor rig determines what part of the surroundings is observed at all.

In practice, these differences are usually absorbed by forking the local planner. Each embodiment receives its own code path, its own compiled-in constants, and its own sensor preprocessing. This works for one or two robots, but it scales poorly. Bug fixes must be ported across forks, parameter tuning diverges, and validation evidence collected on one robot does not transfer to another. A common intermediate solution is a single planner controlled by configuration flags and per-robot parameter files. That approach removes some duplicated code, but loosely typed flags provide no guarantee that a parameter set is internally consistent, that it matches the hardware actually present, or that it is current.

Configuration-driven cross-embodiment planning offers a more principled route. AgniNav [1] showed that a single local planning stack can be configured for different robots through an abstract robot description consisting of collision geometry, sensor parameters, and velocity limits, so that the planner does not need to be rewritten when the embodiment changes. AgniNav, however, addresses one robot at a time. A fleet adds requirements that a single-robot configuration file does not address: many descriptors must coexist at run time, descriptors must be distributed to robots and planners consistently, sensor rigs vary in number, type, and placement rather than only in mounting height, and a descriptor can become stale when hardware is changed in the field.

This paper develops a fleet-level configuration abstraction for local planning and evaluates whether it allows one unchanged planning binary to serve physically different robots without loss of plan quality. The specific gap addressed here is the absence of a run-time contract between fleet configuration and planning: existing configuration-driven planners define *what* an embodiment descriptor contains, but not how descriptors are validated, versioned, resolved, and kept consistent across a heterogeneous fleet.

The contributions are as follows.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

1. A compact, typed configuration schema for heterogeneous fleets that separates collision geometry, kinematic and payload limits, locomotion class, and the sensor rig, and that generalizes the four-parameter envelope of AgniNav to polygonal footprints and multi-sensor rigs (Section 4.1).
2. A configuration registry service that validates descriptors against static and cross-field constraints, issues immutable content-addressed handles, detects stale descriptors at run time, and caches derived planning quantities (Section 4.5).
3. A handle-based planner interface and a sampling-based local planner whose embodiment dependence enters only through derived quantities, so the same binary is executed for every morphology (Sections 4.3–4.4).
4. A simulation study with four morphologies, 15 mixed-fleet scenarios, and 5 seeds, measuring shared module reuse, configuration load latency, plan quality parity, and replanning latency against per-embodiment and flag-based baselines, together with an ablation and an analysis of recorded failure cases (Section 5).

02 Related Work

2.1 Configuration-driven and geometric cross-embodiment planning

The most direct predecessor of this work is AgniNav [1], a configuration-driven local planner for cross-embodiment robot navigation. AgniNav represents the robot through an abstract configuration that includes body geometry, sensor parameters, and velocity limits. Its central idea is that perception and planning should consume this configuration explicitly: a collision-relevant height conditions the conversion of monocular observations into a scan-like representation, and the body envelope, described by front length, rear length, and half width, configures collision checking inside the local planner. As a result, one planning stack can run on robots of different sizes without retraining or code changes. The present work adopts this principle without modification and builds on it in three directions. First, it replaces the rectangular envelope with a polygonal footprint and a circle cover, which is needed for humanoids and rovers whose outlines are poorly approximated by a single box. Second, it treats the sensor rig as a set of heterogeneous sensors with explicit extrinsics rather than a single camera with a configured height. Third, and most importantly, it adds the fleet-level machinery that AgniNav does not target: a registry, immutable handles, validation, staleness detection, and caching. In short, AgniNav defines how a planner should use a configuration, whereas this paper addresses how many configurations are managed and served to one planner binary at run time.

CeRLP [2] pursues a related geometric strategy for visual navigation. It converts heterogeneous camera observations into a unified geometric representation, including scale-corrected depth and a height-adaptive scan abstraction, before local planning. CeRLP and AgniNav share the view that an explicit geometric interface between perception and control makes cross-embodiment behavior easier to diagnose than end-to-end mappings. The sensor-rig abstraction in Section 4.3 follows the same view but fuses multiple sensors per robot into a shared elevation representation, and it lets the descriptor, rather than the perception module, determine which height band is collision-relevant.

2.2 Learned cross-embodiment navigation

A large body of work treats embodiment diversity as a data problem. GNM [3] trains a single goal-conditioned navigation policy on trajectories from several robots by using a normalized action space and temporal context. ViNT [4] scales this approach to a Transformer foundation model that can be adapted to new tasks, and NoMaD [5] unifies goal-directed navigation and exploration with a diffusion policy. These models demonstrate substantial transfer of visual priors across platforms, but body geometry enters only implicitly through the data distribution. RING [6] randomizes body size, rotation pivot, and camera placement in simulation to obtain a policy that generalizes across indoor robots, and NavFoM [7] trains a navigation foundation model across quadrupeds, drones, wheeled robots, and vehicles with embodiment and camera identifiers. EA-Nav [8] adds embodiment geometry as conditioning tokens and trains risk-aware corrections to reduce observation-action ambiguity. In manipulation and navigation more broadly, Yang et al. [9] and CrossFormer [10] train single policies over many embodiments, building on large multi-robot datasets such as Open X-Embodiment [11] and generalist policies such as Octo [12]. BADGR [13] shows that navigation affordances can be learned from a robot's own experience.

These approaches are complementary to the present work. Learned policies can supply goals, subgoals, or traversability priors, but their embodiment conditioning is either implicit or represented by learned tokens whose physical meaning is not checked at deployment. The configuration abstraction proposed here operates at the layer where physical dimensions determine hard feasibility, and its descriptor could serve as the conditioning signal for such policies, with the benefit that the signal is validated and versioned.

2.3 Local planning and terrain representation for legged robots

Sampling-based model predictive control, in particular model predictive path integral (MPPI) control [14], evaluates many stochastic control sequences under a motion model and combines them by exponentiated cost. It is attractive for heterogeneous fleets because the motion model and cost can be swapped without changing the optimizer. Gradient-based local planners such as EGO-Planner [15] achieve low latency for aerial robots by avoiding a full distance field. Nav2 [16] provides a widely used plugin architecture for planners and controllers on ROS 2, but robot geometry and kinematics are still supplied as per-robot parameters to individual plugins rather than through a common validated descriptor.

For legged platforms, terrain representation is as important as the optimizer. GPU-based elevation mapping [17] fuses range data from multiple sensors into a robot-centric 2.5D map, and Wild Visual Navigation [18] learns traversability online from self-supervision. STEP [19] propagates traversability uncertainty into risk-aware planning for off-road robots. Perceptive nonlinear MPC [20] couples terrain perception with whole-body locomotion, and end-to-end methods learn locomotion and local navigation jointly [21, 22]. Adaptive safety margins for visual navigation [23] learn context-dependent clearance budgets instead of fixed inflation. The planner in Section 4.4 uses a deliberately conventional MPPI formulation on a fused elevation grid, because the focus of this paper is the configuration layer that feeds it rather than a new optimizer.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

2.4 Multi-embodiment locomotion control

Learned locomotion controllers have made legged robots robust on rough terrain [24, 25, 26], enabled by massively parallel simulation [27, 28]. Rapid motor adaptation [29] estimates environment and robot parameters online from proprioception. Several works train a single locomotion controller for several bodies: GenLoco [30] randomizes morphology across quadrupeds, ManyQuadrupeds [31] learns one policy for diverse quadrupeds, and Bohlinger et al. [32] extend multi-embodiment locomotion to robots with different numbers of legs. Learned humanoid locomotion has also been demonstrated in the real world [33]. This line of work shows that low-level control can be shared across bodies. The present paper addresses the layer above it: it assumes each robot has a velocity-tracking controller and asks how a single local planner can command all of them consistently.

2.5 Software abstractions for robot configuration

ROS 2 [34] provides typed messages, parameters, and lifecycle-managed nodes, which give a foundation for run-time configuration but leave the semantics of robot parameters to each package. MoveIt [35] demonstrates how a robot description and a semantic description can make a complex planning framework reusable across manipulators, and behavior trees [36] provide modular task composition used in Nav2. The registry proposed here is in the spirit of these abstractions but targets the specific problem of serving many embodiment descriptors to a navigation planner: it provides schema validation, content addressing, staleness detection, and derived-quantity caching, which are not part of standard parameter servers.

2.6 Simulation and evaluation

Embodied AI simulators such as Habitat [37] and Gibson [38] enable large-scale evaluation, but simulation rankings do not always predict real-world rankings [39]. Standard navigation metrics such as success weighted by path length [40] and success weighted by completion time [41] compare agents on a common task. The plan quality parity metric used here is narrower by design: it asks whether a shared planner reproduces the quality of a morphology-specific reference on the same replanning problem, which isolates the effect of the configuration layer from task-level navigation performance.

03 Problem Formulation

3.1 Fleet, embodiments, and planning problem

Consider a fleet $\mathcal{R} = \{r_1, \dots, r_N\}$ of mobile robots. At replanning step k, robot r_i has planar state $\mathbf{x} = (p_x, p_y, \psi) \in SE(2)$, previous command $\mathbf{u}^-$, local goal $\mathbf{g} \in \mathbb{R}^2$, and a robot-centric local map $\mathcal{E}$ built from its sensors. A local planner returns a control sequence $U = (\mathbf{u}_0, \dots, \mathbf{u}_{H-1})$ over a horizon of H steps of duration Δt, with commands $\mathbf{u} = (v_x, v_y, \omega)$ expressed in the body frame. The first command is passed to an embodiment-specific velocity-tracking controller.

Definition 1 (Embodiment descriptor). An embodiment descriptor is a tuple

$$d = (\mathcal{G},\ \mathcal{K},\ \ell,\ \mathcal{S},\ \mathcal{P},\ \nu) \in \mathcal{D}, \tag{1}$$

where $\mathcal{G}$ is collision geometry, $\mathcal{K}$ is the set of kinematic limits, ℓ is the locomotion class, $\mathcal{S}$ is the sensor rig, $\mathcal{P}$ is the payload state, and ν is version metadata. $\mathcal{D}$ is the set of descriptors admitted by the schema of Section 4.1.

Definition 2 (Configuration handle). A handle is an opaque identifier $h = \eta(d)$ produced by the registry, where η is injective on admitted descriptors with overwhelming probability. A handle refers to exactly one immutable descriptor; changing any field yields a new handle.

Definition 3 (Descriptor-parameterized planner). A planner is a single program π that computes

$$U = \pi\big(\mathbf{x}, \mathbf{u}^-, \mathbf{g}, \mathcal{E};\ h\big), \tag{2}$$

in which all embodiment dependence enters through the handle h. In particular, the program text of π is the same for every robot.

3.2 Quality and reuse criteria

Let $\pi_i^\star$ denote a morphology-specific reference planner for robot r_i that uses the full model of that morphology and an unrestricted computation budget, and let $J_i(U)$ be the evaluation cost defined in Section 5.3. For a set of M replanning problems indexed by m, the plan quality parity of a planner π is

$$\mathrm{PQP}(\pi) = \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}\Big[\mathrm{feas}_m(U_m) \wedge J_m(U_m) \le (1+\epsilon)\, J_m(U_m^\star)\Big], \tag{3}$$

where U_m and $U_m^\star$ are the outputs of π and of the reference for the robot in problem m, feas_m indicates that the plan is collision-free and respects the kinematic limits of that robot, and $\epsilon > 0$ is a fixed relative tolerance shared by all methods.

Shared module reuse measures how much of the planning stack is common to all morphologies:

$$\mathrm{SMR} = \frac{\sum_{q \in \mathcal{M}_\cap} \mathrm{LOC}(q)}{\sum_{q \in \mathcal{M}} \mathrm{LOC}(q)}, \tag{4}$$

where $\mathcal{M}$ is the set of source modules in the planning stack, from sensor adapters to the command interface, $\mathcal{M}_\cap \subseteq \mathcal{M}$ contains the modules that are linked and executed for all four morphologies, and LOC counts logical source lines.

3.3 Requirements

The fleet abstraction must satisfy three requirements. (R1) *Binary invariance*: the planner executable is bit-identical across morphologies. (R2) *Latency*: resolving a handle into planning-ready quantities must fit a configuration load budget of 5 ms, and a full replanning cycle must complete within the replanning period. (R3) *Integrity*: an invalid or stale descriptor must never be used silently; it must either be rejected at registration or detected at run time and replaced by a conservative fallback.

04 Method

The system consists of three components, summarized in the architecture note below: the configuration schema, the registry service, and the handle-based planner. Robots register descriptors with the registry, receive handles, and pass the handle together with their state and local map to the shared planner. The planner resolves the handle into a set of derived quantities through an in-process cache backed by the registry.

ARCHITECTURE SUMMARY

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

Robot → register(d) → registry: validate $V(d)$, compute $h = \eta(d)$, compute $\Phi(d)$, store. Robot → plan(h, x, u, g, E) → planner: resolve $\Phi(h)$ from cache or registry, check epoch and sensor-rig consistency, run MPPI, return U.

4.1 Configuration schema

The schema is intentionally compact: it contains only quantities that change the outcome of local planning, and it is typed so that each field has a unit, a range, and a frame.

Collision geometry. $\mathcal{G} = (F, h_{\text{lo}}, h_{\text{hi}})$, where $F \subset \mathbb{R}^2$ is a simple polygon describing the body footprint in the body frame, and $[h_{\text{lo}}, h_{\text{hi}}]$ is the collision-relevant height interval above the local ground, that is, the vertical extent of the body that can contact obstacles. The four-parameter envelope of AgniNav is the special case

$$F = [-l_r,\ l_f] \times [-w,\ w], \tag{5}$$

with rear length l_r, front length l_f, and half width w. Polygonal footprints are needed for the humanoid, whose arms and torso form a non-rectangular outline, and for rovers with protruding sensor masts.

Kinematic limits. $\mathcal{K} = (v_{\max}, v_{\text{lat}}, \omega_{\max}, a_{\max}, \alpha_{\max}, \tau, \theta_{\max}, s_{\max})$ contains maximum forward speed, maximum lateral speed, maximum yaw rate, maximum linear and angular acceleration, command-to-motion latency, maximum traversable slope, and maximum step height. Schema ranges restrict these to physically plausible values; for example, $0 < v_{\max} \le 5\,\text{m/s}$ and $0 < \tau \le 0.2\,\text{s}$.

Locomotion class. $\ell \in \{\text{QUAD}, \text{BIPED}, \text{SKID}\}$ selects one of three motion models that are compiled into the shared binary (Section 4.4). The class is an enumeration rather than free-form code, so adding a robot does not add code unless it introduces a genuinely new class of motion.

Sensor rig. $\mathcal{S} = \{s_j\}_{j=1}^{n_s}$, where each sensor is

$$s_j = \big(\kappa_j,\ T^b_{s_j},\ \Omega_j,\ r_j^{\max},\ f_j,\ \phi_j\big), \tag{6}$$

with modality κ_j (for example depth camera or LiDAR), extrinsic transform $T^b_{s_j} \in SE(3)$ from sensor to body frame, field of view Ω_j, maximum range $r_j^{\max}$ (up to roughly 100 m for LiDAR), nominal rate f_j, and a frame fingerprint ϕ_j that the sensor driver also attaches to its data stream.

Payload. $\mathcal{P} = (m_p, \bar{m}_p, \boldsymbol{\beta})$ contains the current payload mass, the rated maximum payload, and derating coefficients $\boldsymbol{\beta} = (\beta_v, \beta_\omega, \beta_a) \in [0,1)^3$ supplied by the platform integrator.

Version metadata. $\nu = (\text{schema version},\ \text{robot id},\ \text{epoch}\, e)$, where the epoch is a monotonically increasing counter per robot.

4.2 Derived planning quantities

The planner never reads raw descriptor fields during optimization. Instead, the registry computes a set of derived quantities $\Phi(d)$ once per descriptor, which keeps the inner loop identical for every morphology and moves all embodiment-specific geometry processing out of the replanning path.

Payload-derated limits. Effective limits decrease linearly with payload fraction,

$$\bar{v} = v_{\max}\Big(1 - \beta_v \frac{m_p}{\bar{m}_p}\Big), \quad \bar{\omega} = \omega_{\max}\Big(1 - \beta_\omega \frac{m_p}{\bar{m}_p}\Big), \quad \bar{a} = a_{\max}\Big(1 - \beta_a \frac{m_p}{\bar{m}_p}\Big), \tag{7}$$

and the lateral limit $\bar{v}_{\text{lat}}$ is derated with β_v. A linear model is a conservative first-order approximation; it is chosen because integrators can estimate its coefficients from a few speed tests at different loads.

Stopping distance. The distance needed to stop from speed v is

$$d_{\text{stop}}(v) = v\,\tau + \frac{v^2}{2\bar{a}}, \tag{8}$$

which combines latency and deceleration and is used as a speed-dependent clearance requirement.

Circle cover. The footprint is approximated by n_c circles $\{(\mathbf{c}_j, \rho_j)\}$ in the body frame such that

$$F \subseteq \bigcup_{j=1}^{n_c} \mathcal{B}(\mathbf{c}_j, \rho_j) \subseteq F \oplus \mathcal{B}(\mathbf{0}, \varepsilon_c), \tag{9}$$

where $\mathcal{B}(\mathbf{c}, \rho)$ is a disk, $\oplus$ is the Minkowski sum, and ε_c bounds the over-approximation. The cover is computed greedily along the medial axis of F until the outer bound holds. Circle covers make collision checking a sequence of distance-field lookups, whose cost scales with n_c rather than with polygon complexity.

Height band and coverage masks. The obstacle height band is $B = [\,s_{\max},\ h_{\text{hi}} + \delta_h\,]$, with a small vertical margin δ_h, and each sensor yields a static coverage mask C_j over the robot-centric grid computed from $T^b_{s_j}$, Ω_j, and $r_j^{\max}$. Both depend only on the descriptor and are therefore cached.

4.3 Sensor-rig abstraction

All sensors are mapped into a common robot-centric 2.5D elevation grid $\mathcal{E}$ with cell height $z(\mathbf{q})$ and local ground estimate $z_g(\mathbf{q})$ at cell $\mathbf{q}$, in the manner of GPU elevation mapping [17]. A point $\mathbf{p}^s$ measured by sensor s_j is transformed into the body frame by

$$\mathbf{p}^b = T^b_{s_j}\,\mathbf{p}^s, \tag{10}$$

and fused into $\mathcal{E}$ with a per-modality noise model. Given the descriptor, a cell is an obstacle if

$$o_d(\mathbf{q}) = \mathbf{1}\Big[\, z(\mathbf{q}) - z_g(\mathbf{q}) \in B \ \ \vee \ \ \sigma(\mathbf{q}) > \theta_{\max}\Big], \tag{11}$$

where $\sigma(\mathbf{q})$ is the local slope estimated from a plane fit over neighboring cells. Protrusions lower than $s_{\max}$ are treated as steppable, and overhangs above $h_{\text{hi}} + \delta_h$ are ignored, so the same map yields different obstacle sets for a compact quadruped and a humanoid. A Euclidean distance transform of o_d gives the clearance field $D_d(\mathbf{q})$. Cells outside $\bigcup_j C_j$, or cells with no recent measurement, are marked unknown.

This abstraction is what allows sensor rigs to differ freely. A robot with a single forward depth camera and a robot with a LiDAR and two side cameras both produce the same map type, and differences in coverage appear only as different unknown regions, which the planner penalizes explicitly.

4.4 Handle-based sampling planner

Motion models. The planner propagates the state with

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta t \begin{bmatrix} R(\psi_t)\,[\,v_{x,t},\ v_{y,t}\,]^\top \\ \omega_t \end{bmatrix}, \tag{12}$$

where $R(\psi)$ is the planar rotation matrix. The locomotion class constrains the lateral channel: QUAD allows $|v_y| \le \bar{v}_{\text{lat}}$; BIPED allows a smaller lateral speed and adds a turn-in-place penalty that reflects stepping constraints; and SKID enforces the nonholonomic constraint $v_y = 0$.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

Admissible controls. The admissible set at time t is

$$\mathcal{U}_d(\mathbf{u}_{t-1}) = \Big\{\mathbf{u} : |v_x| \le \bar{v},\ |v_y| \le \bar{v}_{\text{lat}},\ |\omega| \le \bar{\omega},\ |\mathbf{u} - \mathbf{u}_{t-1}| \le \mathbf{a}_d\,\Delta t\Big\}, \tag{13}$$

with $\mathbf{a}_d = (\bar{a}, \bar{a}, \alpha_{\max})$ and componentwise inequalities. Sampled controls are projected onto $\mathcal{U}_d$ before rollout.

Cost. For a rollout $X = (\mathbf{x}_0, \dots, \mathbf{x}_H)$ generated by U, the planning cost is

$$J(U) = \sum_{t=0}^{H-1} \Big[w_g \,\|\mathbf{p}_t - \mathbf{g}\| + w_o \sum_{j=1}^{n_c} \varphi\big(\chi_{j,t}\big) + w_n\, \mathbf{1}[\text{unk}_t] + w_u \,\|\mathbf{u}_t - \mathbf{u}_{t-1}\|^2 \Big] + w_H \,\|\mathbf{p}_H - \mathbf{g}\|, \tag{14}$$

where $\mathbf{p}_t$ is the position component of $\mathbf{x}_t$, $\chi_{j,t} = D_d\big(\mathbf{x}_t \oplus \mathbf{c}_j\big) - \rho_j$ is the clearance of circle j at time t (with $\mathbf{x}_t \oplus \mathbf{c}_j$ denoting the circle center transformed to the map frame), $\varphi(\chi) = \exp(-\chi/\lambda_o)$ is a barrier with length scale λ_o, unk_t indicates that the footprint overlaps unknown cells, and the w terms are weights. The BIPED turn-in-place penalty is added to the stage cost when $\ell = \text{BIPED}$. A rollout is infeasible, and its cost set to infinity, if

$$\min_j \chi_{j,t} < \delta_s + d_{\text{stop}}(|v_{x,t}|)\,\gamma_t \tag{15}$$

for any t, where δ_s is a static safety margin and $\gamma_t \in [0, 1]$ is the cosine between the heading and the direction to the nearest obstacle, so that the stopping-distance requirement applies only toward obstacles ahead of the robot.

Optimization. The planner uses MPPI [14]. Given a nominal sequence $\bar{U}$, it samples K perturbed sequences $U^{(k)} = \text{proj}_{\mathcal{U}_d}(\bar{U} + \mathcal{N}(0, \Sigma_d))$, evaluates their costs $J^{(k)}$, and updates

$$\bar{U} \leftarrow \sum_{k=1}^{K} w^{(k)} U^{(k)}, \qquad w^{(k)} = \frac{\exp\big(-(J^{(k)} - J_{\min})/\lambda\big)}{\sum_{k'} \exp\big(-(J^{(k')} - J_{\min})/\lambda\big)}, \tag{16}$$

where $J_{\min} = \min_k J^{(k)}$ and λ is the temperature. The sampling covariance Σ_d is scaled by the derated limits so that the exploration noise is proportionate to each robot's control authority. The first command of $\bar{U}$ is sent to the tracking controller and $\bar{U}$ is shifted by one step for warm starting.

Timing. The planner runs at 10 Hz, so the replanning period is 100 ms, while each robot's tracking controller runs at 50 Hz (a 20 ms control period). The horizon $H\Delta t$ is set from the descriptor so that it covers at least the stopping distance at $\bar{v}$, which keeps the look-ahead proportional to each robot's speed.

Why one binary suffices. Equations (11)–(16) contain no morphology-specific branches other than the enumerated locomotion class. Every quantity that differs between robots, namely $\bar{v}$, $\bar{v}_{\text{lat}}$, $\bar{\omega}$, $\bar{a}$, d_{stop}, the circle cover, B, the coverage masks, Σ_d, and H, is an element of $\Phi(d)$. The planner API therefore exposes a single entry point, plan(h, x, u, g, E), and the binary does not change when a robot is added.

4.5 Registry service, validation, and caching

Handles. On registration, the registry serializes the descriptor canonically (sorted keys, fixed units, fixed numeric precision) and computes

$$h = \eta(d) = \text{trunc}_{128}\big(\text{SHA-256}(\text{canon}(d))\big). \tag{17}$$

Content addressing makes handles immutable and globally comparable: two robots with identical configurations share a handle and its cached quantities, and any field change, including a payload update, yields a new handle and an incremented epoch.

Schema validation. A descriptor is admitted only if

$$V(d) = \bigwedge_i c_i(d) = \text{true}, \tag{18}$$

where the constraints c_i fall into three groups. *Static* constraints check types, units, and ranges. *Cross-field* constraints check consistency between fields, for example $h_{\text{lo}} < h_{\text{hi}}$, $m_p \le \bar{m}_p$, $\bar{v} > 0$ after derating, $v_{\text{lat}} = 0$ when $\ell = \text{SKID}$, a simple polygon F, and sensor mounts that lie on or within the body envelope dilated by a mount tolerance. *Run-time* constraints are evaluated by the planner at every cycle: the epoch of the bound handle must equal the latest epoch for the robot, and the frame fingerprints ϕ_j arriving with sensor data must match those in $\mathcal{S}$. A run-time violation marks the descriptor stale. The planner then switches to a conservative fallback that halves $\bar{v}$, doubles δ_s, and treats cells observed only by mismatched sensors as unknown until a valid descriptor is registered.

Derived-quantity cache. $\Phi(d)$ is computed once at registration and stored under h. Each planner process keeps an in-process cache keyed by handle in front of the registry, so a cache hit resolves a handle without a network round trip. Because handles are immutable, the cache requires no invalidation protocol: entries are never updated, only superseded by new handles.

4.6 Design choices

Several design choices deserve justification. *Typed schema rather than free-form parameters*: typed fields with units and ranges allow cross-field validation, which is impossible when parameters are opaque key-value pairs. *Enumerated locomotion classes rather than plugins*: classes keep the binary fixed and make the set of motion models auditable, at the cost of requiring a code change for a genuinely new motion class. *Content-addressed handles rather than robot identifiers*: a robot identifier names a robot, not a configuration, so it cannot distinguish a robot before and after a sensor swap. *Precomputed derived quantities*: circle covers, coverage masks, and derated limits are costly to recompute and unchanged between descriptor updates, so computing them once keeps the replanning path short and identical across robots. *Conservative fallback rather than halting*: a stale descriptor is usually a small discrepancy, and continued slow operation is safer in a fleet than abrupt stops in shared corridors.

05 Experiments

5.1 Setup and platform

The evaluation uses a simulated heterogeneous fleet of four morphologies with distinct sensor rigs: a Go2-class compact quadruped, a B2-class larger quadruped, a 1.4 m-tall humanoid,

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

Table 1: Simulated fleet. All four morphologies are served by the same planner binary.

Morphology	Class ℓ	Distinguishing descriptor content
Go2-class quadruped	QUAD	Compact footprint, lateral motion, own sensor rig
B2-class quadruped	QUAD	Longer body, higher payload, own sensor rig
Humanoid (1.4 m tall)	BIPED	Tall height band, polygonal footprint, own sensor rig
4WD rover	SKID	No lateral motion ($v_y = 0$), own sensor rig

and a four-wheel-drive rover (Table 1). The test set contains 15 mixed-fleet navigation scenarios, each run with 5 seeds, and each scenario contains 40 replanning events. The protocol therefore comprises $15 \times 5 = 75$ scenario-seed episodes and $75 \times 40 = 3{,}000$ scheduled replanning events per method. The planner binary is bit-identical across morphologies, satisfying (R1). The configuration registry service runs on a 4-core virtual machine.

5.2 Baselines

Two baselines are compared with the proposed configuration registry. The *per-embodiment planner* maintains four separate code paths, one per morphology, each with compiled-in parameters, its own sensor preprocessing, and its own motion model, while sharing common libraries such as map storage and math utilities. The *shared planner with ad hoc configuration flags* uses one code path controlled by boolean and enumerated flags and per-robot parameter files, which is a common pattern in parameter-server-based stacks; it has no typed schema, no validation, no content-addressed handles, and no derived-quantity cache. All methods use the same MPPI optimizer family and the same evaluation cost, so differences reflect how embodiment information reaches the planner.

5.3 Metrics

Four metrics are reported. *Shared module reuse* is SMR in (4). *Configuration load latency* is the time from a handle (or, for the baselines, a robot identifier) to planning-ready quantities. *Plan quality parity* is PQP in (3), with the evaluation cost J in (14) computed under each robot's reference model and a reference planner per morphology. *Replan p95 latency* is the 95th-percentile wall-clock time of one complete replanning cycle.

5.4 Main results

Table 2 reports the main comparison.

Plan quality parity. The configuration registry reaches a parity of 96.4%, compared with 81.2% for the per-embodiment planner and 88.2% for the flag-based shared planner, an improvement of 15.2 and 8.2 percentage points, respectively. Equivalently, the fraction of replans that fail the parity criterion falls from 18.8% to 3.6%. That separate, hand-specialized code paths achieve the lowest parity may appear surprising. A plausible explanation, consistent with the ablation below, is that the per-embodiment paths encode sensor handling and derived constants separately for each robot, so inconsistencies between a path and the robot it serves are not caught, whereas the registry derives all such quantities from one validated descriptor. The data do not decompose parity by morphology, apart from the rover slope case discussed in Section 5.6, so this explanation remains a hypothesis.

Code reuse. Shared module reuse rises from 83.2% for the per-embodiment planner and 83.8% for the flag-based planner to 87.2%. The per-embodiment baseline already shares most of its code through common libraries, so the 4.0 percentage-point gain reflects the removal of the forked planning, preprocessing, and motion-model code. Reuse does not reach 100% for the proposed system because sensor drivers and the command translation into each robot's tracking controller remain morphology-specific; these modules lie outside the planner binary.

Latency. Configuration load latency is similar for all methods (1.8 to 2.0 ms), which indicates that the registry's validation and hashing do not add load-time overhead on the cached path and that all three methods meet the 5 ms configuration load budget in steady state. The 95th-percentile replanning latency is 34 ms for the proposed system, 35 ms for the flag-based planner, and 38 ms for the per-embodiment planner, a reduction of 4 ms (about 10.5%) relative to the latter. All three remain well within the 100 ms replanning period. Because derived quantities are precomputed, the replanning path of the proposed planner contains only map fusion, distance transform, and MPPI rollouts.

5.5 Ablation

Table 3 removes one component of the registry at a time and reports plan quality parity.

Removing the derived-quantity cache lowers parity by 6.3 percentage points to 90.1%. Without the cache, circle covers, coverage masks, and derated limits must be recomputed inside the replanning cycle, which leaves less of the cycle for MPPI sampling; the drop is consistent with reduced optimization effort under a fixed period. Removing schema validation lowers parity by 9.3 points to 87.1%, which is close to the flag-based baseline (88.2%); this suggests that much of the advantage over flag-based configuration comes from rejecting inconsistent descriptors rather than from the handle mechanism itself. Removing the sensor-rig abstraction, so that each robot's primary range sensor is consumed under a fixed default mounting assumption without extrinsic-driven fusion or coverage masks, produces the largest drop, 12.7 points to 83.7%, close to the per-embodiment planner (81.2%). Sensor heterogeneity, rather than body geometry alone, therefore appears to be the dominant obstacle to sharing a planner across this fleet.

5.6 Failure cases

Three failure cases were recorded.

Rover on steep slopes. For the rover morphology, plan quality parity dips to 88% in steep-slope scenarios, 8.4 percentage points below the fleet-wide 96.4%. The descriptor encodes the rover's nonholonomic constraint ($v_y = 0$) and a maximum slope $\theta_{\max}$, but the differential-drive constraint is not fully captured: on slopes, skid-steer turning depends on traction and track slip, so yaw-rate authority varies with terrain inclination and the constant $\bar{\omega}$ in (7) overstates the turning capacity. The planner then generates turns that the reference model cannot execute, and those replans fail the feasibility condition in (3). A slope-dependent yaw-rate limit in the schema is the natural remedy.

Registry cold start under concurrent load. When 40 configuration loads arrived concurrently at a cold registry, the 99th-percentile configuration load latency was 11 ms, above the 5 ms budget of (R2) by a factor of 2.2. The cause is that every request

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

Table 2: Main comparison on the simulated four-morphology fleet (15 scenarios × 5 seeds, 40 replans per scenario). Higher is better for reuse and parity; lower is better for latencies.

Method	Shared module reuse (%)	Config load latency (ms)	Plan quality parity (%)	Replan p95 latency (ms)
Per-embodiment planner (4 separate code paths)	83.2	1.9	81.2	38
Shared planner, ad-hoc config flags	83.8	2.0	88.2	35
Ours	**87.2**	**1.8**	**96.4**	**34**

Table 3: Ablation of the configuration registry (plan quality parity).

Variant	Plan quality parity (%)
Ours (configuration registry)	96.4
w/o derived-quantity cache	90.1
w/o schema validation	87.1
w/o sensor-rig abstraction	83.7

on a cold start computes or fetches $\Phi(d)$ from the 4-core registry at the same moment. The issue was fixed by the in-process cache described in Section 4.5, which serves repeated handles locally; because handles are immutable, cached entries never need invalidation. The steady-state load latencies in Table 2 reflect the cached path.

Stale descriptor after a mid-run sensor swap. In one scenario with mixed lighting and a sensor swap during the run, the robot continued to reference a descriptor whose sensor rig no longer matched the hardware. The run-time validation layer detected the frame-fingerprint mismatch in 41 ms, less than half of one 100 ms replanning period, and switched the planner to the conservative fallback. This case illustrates why run-time constraints are part of $V(d)$: a descriptor that was valid at registration can become wrong later, and only a check against live sensor metadata reveals it.

06 Limitations

The study has several limitations. First, all results come from simulation; the fleet has not been evaluated on physical robots, and simulation results may not predict real-world behavior [39]. Second, the fleet contains four morphologies, which is enough to exercise the abstraction but not to show how it scales to tens of distinct configurations or to articulated bodies whose footprint changes during motion. Third, the descriptor is static between registrations. The rover failure shows that some embodiment properties, such as turning authority on slopes, depend on terrain and cannot be represented by constants; a terrain-conditioned descriptor or online estimation in the spirit of rapid motor adaptation [29] would be needed. Fourth, the plan quality parity metric compares against morphology-specific reference planners and a fixed tolerance, so it measures fidelity to a reference rather than task-level navigation success, and it is not reported per morphology except for the rover slope case. Fifth, the payload derating model in (7) is linear and relies on integrator-supplied coefficients. Finally, the registry was evaluated on a single 4-core virtual machine, and the cold-start result indicates that its behavior under fleet-scale concurrency depends on local caching; replication and failover were not studied.

07 Conclusion

This paper extended configuration-driven local planning from individual robots to heterogeneous fleets. A typed configuration schema, a registry that validates descriptors, issues content-addressed handles, and caches derived quantities, and a planner interface that accepts handles together allowed one bit-identical planning binary to serve a Go2-class quadruped, a B2-class quadruped, a 1.4 m humanoid, and a four-wheel-drive rover in simulation. Relative to four separate per-embodiment code paths, the abstraction raised shared module reuse from 83.2% to 87.2%, reduced configuration load latency from 1.9 ms to 1.8 ms and 95th-percentile replanning latency from 38 ms to 34 ms, and raised plan quality parity from 81.2% to 96.4%. The ablation indicates that the sensor-rig abstraction and schema validation contribute most of this gain. The recorded failures point to the next steps: terrain-dependent descriptor fields for differential-drive platforms, registry designs that remain within budget on cold starts, and validation against physical fleets.

References

[1] T. Zang, S. Cheng, H. Huang, S. Wang, and W. Zhang. AgniNav: Configuration-Driven Cross-Embodiment Local Planning for Robot Navigation. arXiv:2606.10903, 2026.

[2] H. Xi, M. Tan, X. Zhang, S. Cheng, S. Wang, Y. Gu, X. Shen, and W. Zhang. CeRLP: A Cross-embodiment Robot Local Planning Framework for Visual Navigation. arXiv:2603.19602, 2026.

[3] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, and S. Levine. GNM: A General Navigation Model to Drive Any Robot. arXiv:2210.03370, 2022.

[4] D. Shah, A. Sridhar, N. Dashora, K. Stachowicz, K. Black, N. Hirose, and S. Levine. ViNT: A Foundation Model for Visual Navigation. arXiv:2306.14846, 2023.

[5] A. Sridhar, D. Shah, C. Glossop, and S. Levine. NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration. arXiv:2310.07896, 2023.

[6] A. Eftekhar et al. The One RING: a Robotic Indoor Navigation Generalist. arXiv:2412.14401, 2024.

[7] J. Zhang et al. Embodied Navigation Foundation Model. arXiv:2509.12129, 2025.

[8] J. Zhang, Y. Du, X. Guo, S. Sun, X. Liu, Y. Sun, G. Lu, W. Sui, and J. Li. EA-Nav: Learning Safe Visual Navigation Policies with Embodiment Awareness. arXiv:2607.19880, 2026.

[9] J. Yang, C. Glossop, A. Bhorkar, D. Shah, Q. Vuong, C. Finn, D. Sadigh, and S. Levine. Pushing the Limits of Cross-Embodiment Learning for Manipulation and Navigation. arXiv:2402.19432, 2024.

[10] R. Doshi, H. Walke, O. Mees, S. Dasari, and S. Levine. Scaling Cross-Embodied Learning: One Policy for Manipulation, Navigation, Locomotion and Aviation. arXiv:2408.11812, 2024.

[11] Open X-Embodiment Collaboration. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. arXiv:2310.08864, 2023.

[12] Octo Model Team. Octo: An Open-Source Generalist Robot Policy. arXiv:2405.12213, 2024.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008

[13] G. Kahn, P. Abbeel, and S. Levine. BADGR: An Autonomous Self-Supervised Learning-Based Navigation System. arXiv:2002.05700, 2020.

[14] G. Williams, A. Aldrich, and E. Theodorou. Model Predictive Path Integral Control using Covariance Variable Importance Sampling. arXiv:1509.01149, 2015.

[15] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao. EGO-Planner: An ESDF-free Gradient-based Local Planner for Quadrotors. arXiv:2008.08835, 2020.

[16] S. Macenski, F. Martín, R. White, and J. Ginés Clavero. The Marathon 2: A Navigation System. arXiv:2003.00368, 2020.

[17] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, and M. Hutter. Elevation Mapping for Locomotion and Navigation using GPU. arXiv:2204.12876, 2022.

[18] M. Frey, M. Mattamala, N. Chebrolu, C. Cadena, M. Fallon, and M. Hutter. Fast Traversability Estimation for Wild Visual Navigation. arXiv:2305.08510, 2023.

[19] D. D. Fan, K. Otsu, Y. Kubo, A. Dixit, J. Burdick, and A. Agha-Mohammadi. STEP: Stochastic Traversability Evaluation and Planning for Risk-Aware Off-road Navigation. arXiv:2103.02828, 2021.

[20] R. Grandia, F. Jenelten, S. Yang, F. Farshidian, and M. Hutter. Perceptive Locomotion through Nonlinear Model Predictive Control. arXiv:2208.08373, 2022.

[21] N. Rudin, D. Hoeller, M. Bjelonic, and M. Hutter. Advanced Skills by Learning Locomotion and Local Navigation End-to-End. arXiv:2209.12827, 2022.

[22] D. Hoeller, N. Rudin, D. Sako, and M. Hutter. ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots. arXiv:2306.14874, 2023.

[23] J. Hu, S. Yuan, G. C. R. Bethala, A. Tzes, and Y. Fang. Learning Adaptive Safety Margins for Visual Navigation. arXiv:2607.18200, 2026.

[24] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning Quadrupedal Locomotion over Challenging Terrain. arXiv:2010.11251, 2020.

[25] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning Robust Perceptive Locomotion for Quadrupedal Robots in the Wild. arXiv:2201.08117, 2022.

[26] A. Agarwal, A. Kumar, J. Malik, and D. Pathak. Legged Locomotion in Challenging Terrains using Egocentric Vision. arXiv:2211.07638, 2022.

[27] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. arXiv:2109.11978, 2021.

[28] V. Makoviychuk et al. Isaac Gym: High Performance GPU-Based Physics Simulation for Robot Learning. arXiv:2108.10470, 2021.

[29] A. Kumar, Z. Fu, D. Pathak, and J. Malik. RMA: Rapid Motor Adaptation for Legged Robots. arXiv:2107.04034, 2021.

[30] G. Feng, H. Zhang, Z. Li, X. B. Peng, B. Basireddy, L. Yue, Z. Song, L. Yang, Y. Liu, K. Sreenath, and S. Levine. GenLoco: Generalized Locomotion Controllers for Quadrupedal Robots. arXiv:2209.05309, 2022.

[31] M. Shafiee, G. Bellegarda, and A. Ijspeert. ManyQuadrupeds: Learning a Single Locomotion Policy for Diverse Quadruped Robots. arXiv:2310.10486, 2023.

[32] N. Bohlinger, G. Czechmanowski, M. Krupka, P. Kicki, K. Walas, J. Peters, and D. Tateo. One Policy to Run Them All: an End-to-end Learning Approach to Multi-Embodiment Locomotion. arXiv:2409.06366, 2024.

[33] I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath. Real-World Humanoid Locomotion with Reinforcement Learning. arXiv:2303.03381, 2023.

[34] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot Operating System 2: Design, Architecture, and Uses in the Wild. arXiv:2211.07752, 2022.

[35] D. Coleman, I. Sucan, S. Chitta, and N. Correll. Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. arXiv:1404.3785, 2014.

[36] M. Colledanchise and P. Ögren. Behavior Trees in Robotics and AI: An Introduction. arXiv:1709.00084, 2017.

[37] M. Savva et al. Habitat: A Platform for Embodied AI Research. arXiv:1904.01201, 2019.

[38] F. Xia, A. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese. Gibson Env: Real-World Perception for Embodied Agents. arXiv:1808.10654, 2018.

[39] A. Kadian et al. Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance? arXiv:1912.06321, 2019.

[40] P. Anderson et al. On Evaluation of Embodied Navigation Agents. arXiv:1807.06757, 2018.

[41] N. Yokoyama, S. Ha, and D. Batra. Success Weighted by Completion Time: A Dynamics-Aware Evaluation Criteria for Embodied Navigation. arXiv:2103.08022, 2021.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2607.00008