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

KAGG RECORD
KAGG:2608.00010

Velocity Interface Contracts: Specifications and Verification for Legged Robot Middleware

Clara Fischer • **Diego Navarro** • **Amelie Laurent**

Robot Dynamics Group, ETH Zurich, Zurich, Switzerland

PUBLISHED	REGISTERED	RECORD TYPE
August 26, 2026	September 2, 2026	Preprint

ABSTRACT

Navigation stacks for legged robots rely on implicit assumptions about the velocity interface exposed by locomotion middleware: how quickly a command takes effect, where it saturates, and whether the response depends on gait mode or command history. These properties are rarely documented, and their violation surfaces as degraded tracking rather than as an explicit fault. This paper introduces velocity interface contracts, a typed declarative specification of command fidelity bounds, latency envelopes, saturation regions, and mode semantics, together with a verification tool that operates in two stages. Offline, the tool checks a calibrated interface model, obtained by planner-conditioned active calibration, against the contract and returns witness commands for each violated clause. Online, it monitors the command and odometry streams with lightweight sequential statistical tests and separates transient violation events from standing violations. The tool was evaluated on a Unitree B2 and an 8-DoF research quadruped over 10 days and 214 h of logged operation, with three known but undocumented B2 firmware behaviors seeded as detection targets. The model-based monitor recorded 11 true-positive detections, against 10 for fixed-threshold monitoring and for a monitor encoding the stack's implicit assumptions, at a false positive rate of 2.8 percent, a 95th-percentile runtime of 1.9 ms at 50 Hz, and a median time to detection of 3.4 s. It flagged the B2 speed limiter 3.4 s into its first engagement, whereas the fixed-threshold monitor required 41 s. The paper also specifies the contract format and proposes an adoption path for legged robot middleware.

INDEX TERMS velocity interface contracts, legged robot middleware, runtime verification, quadruped locomotion, interface calibration, statistical monitoring

KAGG NOTE A velocity command is only as meaningful as the interface behind it; making that interface explicit and checkable turns silent tracking degradation into attributable contract violations.

01 Introduction

Legged robots are increasingly deployed as general-purpose mobile platforms, and most of them expose the same narrow interface to higher-level software: a planar body-velocity command $\mathbf{u} = (v_x, v_y, \omega_z)$ that a vendor-supplied locomotion controller or gait manager converts into joint targets. Model-based controllers [1, 2, 3] and learned locomotion policies [4, 5, 6, 7] alike are trained or tuned to track such commands, and navigation stacks built on ROS and ROS 2 [8, 9, 10] treat the topic that carries them as a clean actuator. The local planner that produces the command, whether a dynamic-window search [11], a sampling-based model-predictive controller [12], or a learned policy [13, 14], usually assumes that the interface responds within a small fraction of its replanning period, reproduces the commanded velocity up to a modest error, and behaves identically regardless of how the robot arrived at its current state.

None of these assumptions is guaranteed. Commercial quadrupeds ship with firmware that silently limits speed under certain conditions, switches gaits with hysteresis, or compensates for payloads by altering the command-to-motion map; research platforms with open-loop gait managers can exhibit response lags that exceed what a planner expects. Because the interface is opaque, these behaviors are discovered indirectly, as path-following error, oscillation near obstacles, or a robot that is slower than its map-based plan predicts. Calibration methods such as GAUGE [15] address part of this problem by actively probing an opaque velocity interface and fitting a model of its response. A calibrated model, however, describes what the interface does; it does not state what the navigation stack requires, and it does not by itself detect when the interface departs from either.

This paper addresses that gap by making the requirements explicit. A *velocity interface contract* is a typed specification, written in a small declarative format, of four properties that navigation software depends on: command fidelity bounds, latency envelopes, saturation regions, and mode semantics, together with a tolerance for slow drift of the operating point. Contracts follow the assume-guarantee tradition of contract-based design [16, 17, 18], and their checking draws on runtime verification [19, 20, 21] and sequential change detection [22, 23, 24]. The resulting tool checks contracts in two places: offline, against a calibrated interface model, and online, on the middleware bus, against the live command and odometry streams.

The contributions of this paper are as follows:

1. A formal definition of velocity interface contracts for legged robots, covering fidelity, latency, saturation, mode semantics, and drift, and a declarative format (version 0.3) in which they can be written and versioned alongside firmware.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

2. An offline verification procedure that checks a calibrated, uncertainty-annotated interface model of the kind produced by GAUGE [15] against a contract, returning a three-valued verdict and witness commands for each clause.
3. A runtime monitor composed of lightweight sequential tests, one per clause type, with an explicit distinction between transient violation events and standing violations, designed to run within one tick of a 50 Hz middleware bus.
4. An evaluation on a Unitree B2 and an 8-DoF research quadruped over 214 h of logged operation, including detection of an undocumented speed limiter in the B2 gait manager, ablations of the monitor components, and an analysis of the recorded failure cases.
5. A proposed adoption path for legged robot middleware, from vendor-published contracts to planner-side consumption of contract parameters.

02 Related Work

2.1 Velocity interfaces in legged locomotion

The velocity command is the de facto boundary between locomotion and navigation. ANYmal [25] and the MIT Cheetah 3 [1] expose body-velocity references to model-based controllers that solve convex or nonlinear motion optimization problems online [1, 2]; perceptive nonlinear MPC [3] extends this to rough terrain while retaining a reference-tracking interface. Learned controllers inherit the same boundary: policies trained with actuator networks [4], teacher-student distillation [5, 6], massively parallel simulation [7], or online adaptation modules [26] are conditioned on a commanded velocity sampled from a training distribution. Margolis et al. show that the command space can be extended with gait parameters to produce a multiplicity of behaviors [27], and that aggressive velocity tracking can be learned for small quadrupeds [28]. Sim-to-real studies of locomotion [29, 30] document that the realized response depends on latency, actuator dynamics, and randomized physical parameters. These works establish that the command-to-motion map is shaped by training ranges, controller structure, and hardware; they do not specify that map to downstream consumers. The present work takes the controller as given and specifies the properties of its interface rather than modifying the controller.

2.2 Navigation stacks and middleware

ROS [8] and ROS 2 [9] provide the publish-subscribe infrastructure on which most legged navigation stacks run, and Nav2 [10] provides a layered navigation system with planners, controllers, and behavior trees above a velocity topic. Navigation systems for legged robots add terrain representations, including probabilistic elevation maps [31], GPU-accelerated elevation mapping [32], and self-supervised traversability [33], and plan over them with sampling-based planners [34] or learned policies [35, 36, 37]. Local controllers such as the dynamic window approach [11] and MPPI [12] encode velocity and acceleration limits explicitly, but they take the limits as fixed parameters and assume the actuator realizes them. Cross-embodiment navigation models such as GNM and ViNT [13, 14] standardize the action space across robots, and configuration-driven local planners such as CeRLP [38] and AgniNav [39] parameterize planning by embodiment descriptors. These efforts make the navigation side more portable, which increases the importance of knowing what the velocity interface on each body actually delivers. A velocity interface contract can be read as the missing interface half of such an embodiment descriptor: it states, in checkable form, the response that a configuration-driven planner presumes.

2.3 Calibration and identification of opaque interfaces

GAUGE [15] is the work most closely related to this paper. It treats the quadruped velocity controller as an opaque interface and calibrates it actively: probe commands are selected conditioned on the planner that will later use the interface, so that calibration effort concentrates on the region of command space the planner actually visits, and a model of command-to-motion response is fitted online from the realized motion. GAUGE answers the question of what the interface does, in the parts of command space that matter to a given planner. The present work is complementary and depends on it. It assumes a calibrated model in the GAUGE style as input, adds a separate specification of what the interface is required to do, and provides the machinery to compare the two offline and to detect, online, when the live interface leaves the calibrated regime. The contract is deliberately independent of any particular calibration method, but the offline checker exploits two features that planner-conditioned calibration provides: parameter uncertainty, which is needed for a sound three-valued verdict, and coverage of the planner's operating region, which bounds where fidelity clauses must hold. Classical system identification [40] supplies the model structures and the uncertainty quantification on which both steps rest.

2.4 Contracts and runtime verification

Design by contract [16] attaches preconditions, postconditions, and invariants to software components; its extension to cyber-physical systems [18, 17] formalizes assume-guarantee contracts over continuous behavior and supports refinement and composition. Velocity interface contracts are assume-guarantee contracts in this sense: the assumption is that commands stay within a declared nominal region and the current mode, and the guarantee bounds fidelity, latency, and saturation behavior. Runtime verification [19] checks properties on executions rather than on models. Signal temporal logic monitoring [20] and its quantitative robustness semantics [21] are the natural formal basis for real-valued signals, and the offline robustness margins used here follow that semantics. ROSRV [41] brings runtime monitoring to ROS by interposing on message traffic to enforce safety and security properties. The present monitor differs from these approaches in that the observed signals are noisy realizations of a dynamical response rather than exact values, so purely logical monitors either alarm constantly or require margins large enough to hide the behaviors of interest; it therefore replaces Boolean predicates with sequential statistical tests whose alarms carry known false-alarm properties.

2.5 Change detection, fault diagnosis, and safety filters

The online tests are drawn from the change detection literature. Page's cumulative sum scheme [22] and Wald's sequential probability ratio test [23] detect shifts in the mean of a residual with bounded detection delay for a given false-alarm rate, and Basseville and Nikiforov [24] develop these tests for residuals of dynamical models. Model-based fault diagnosis [42] uses the same residual-generation principle to isolate actuator and sensor faults. In this vocabulary, an undocumented interface be-

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

havior is a fault relative to the contract rather than relative to the hardware's own design, which changes what counts as a nominal model. Safety filters based on control barrier functions [43] and the broader safe-learning literature [44] constrain actions so that a safe set remains invariant; their guarantees presume that the commanded action is realized by the plant within a known model. Contract monitoring is a precondition for such guarantees rather than a substitute for them: it detects when the assumed actuator model no longer holds.

03 Problem Formulation

3.1 Velocity interfaces

Let $k \in \mathbb{N}$ index ticks of the middleware bus with period Δt. The navigation stack issues a command $\mathbf{u}_k \in \mathcal{U} \subset \mathbb{R}^3$, with components (v_x, v_y, ω_z) expressed in the body frame, and the state estimator reports a realized body velocity $\mathbf{y}_k \in \mathbb{R}^3$ in the same frame. A velocity interface is the causal, possibly stateful map

$$\mathbf{y}_k = \mathcal{I}\left(\mathbf{u}_{0:k},\, \boldsymbol{\xi}_{0:k}\right), \tag{1}$$

where $\boldsymbol{\xi}_k$ collects exogenous factors not visible to the navigation stack, such as terrain, payload, battery state, and estimator noise. The interface is opaque: its internal structure, including any gait manager modes $m_k \in \mathcal{M}$ and their switching logic, is not exposed.

3.2 Calibrated interface models

Following the calibration setting of GAUGE [15], the interface is approximated by a mode-indexed model with a static map, a dead time, and first-order dynamics per axis:

$$\begin{aligned} \mathbf{x}_{k+1} &= A_m \mathbf{x}_k + (I - A_m)\, g_m(\mathbf{u}_{k-d_m}), \\ \mathbf{y}_k &= \mathbf{x}_k + \boldsymbol{\eta}_k, \qquad \boldsymbol{\eta}_k \sim \mathcal{N}(\mathbf{0}, \Sigma_m), \end{aligned} \tag{2}$$

where $\mathbf{x}_k \in \mathbb{R}^3$ is the filtered body velocity, $A_m = \mathrm{diag}(a_{m,1}, a_{m,2}, a_{m,3})$ with $0 \le a_{m,i} < 1$ is the per-axis discrete pole in mode m, $d_m \in \mathbb{N}$ is the dead time in ticks, Σ_m is the measurement noise covariance, and $g_m : \mathcal{U} \to \mathbb{R}^3$ is the steady-state map. The steady-state map is modeled as an affine gain followed by projection onto a feasible set,

$$g_m(\mathbf{u}) = \Pi_{\mathcal{F}_m}(K_m \mathbf{u} + \mathbf{c}_m), \tag{3}$$

with gain matrix $K_m \in \mathbb{R}^{3\times 3}$, offset $\mathbf{c}_m \in \mathbb{R}^3$, and convex feasible set $\mathcal{F}_m$; $\Pi_{\mathcal{F}}$ denotes Euclidean projection. The per-axis time constant and the effective response time, defined as the time to reach $1 - e^{-1}$ of a step, are

$$\tau_{m,i} = -\frac{\Delta t}{\ln a_{m,i}}, \qquad T_{m,i} = d_m \Delta t + \tau_{m,i}. \tag{4}$$

The calibration returns a parameter estimate $\hat{\theta}$, collecting $(K_m, \mathbf{c}_m, \mathcal{F}_m, A_m, d_m, \Sigma_m)$ for all identified modes, together with a covariance P, which defines the confidence set

$$\Theta_\alpha = \left\{ \theta : (\theta - \hat{\theta})^\top P^{-1} (\theta - \hat{\theta}) \le \chi^2_{p,\,1-\alpha} \right\}, \tag{5}$$

where p is the number of continuous parameters and $\chi^2_{p,1-\alpha}$ is the $(1-\alpha)$ quantile of the chi-squared distribution. Dead times and the set of modes are treated as discrete and are enumerated rather than included in Θ_α.

3.3 Velocity interface contracts

Definition 1 (velocity interface contract). A velocity interface contract is a tuple

$$\mathcal{C} = \langle \mathcal{U}_{\text{nom}},\, \Phi_F,\, \Phi_L,\, \Phi_S,\, \Phi_M,\, \Phi_D \rangle, \tag{6}$$

where $\mathcal{U}_{\text{nom}} \subseteq \mathcal{U}$ is the nominal command region (the assumption), and the remaining components are clauses (the guarantee), defined as follows.

Fidelity Φ_F bounds the steady-state tracking error per axis with absolute and relative tolerances $\varepsilon_i^{\text{abs}}, \varepsilon_i^{\text{rel}} \ge 0$:

$$\begin{aligned} &|[g_m(\mathbf{u}) - \mathbf{u}]_i| \le \varepsilon_i(\mathbf{u}) := \varepsilon_i^{\text{abs}} + \varepsilon_i^{\text{rel}} |u_i| \\ &\forall\, \mathbf{u} \in \mathcal{U}_{\text{nom}} \setminus \textstyle\bigcup_j \mathcal{S}_j,\ m \in \mathcal{M},\ i \in \{1,2,3\}, \end{aligned} \tag{7}$$

Latency Φ_L bounds dead time and effective response time by $d^{\max}$ and $\tau^{\max}$: $d_m \Delta t \le d^{\max}$ and $T_{m,i} \le \tau^{\max}$ for all modes and axes.

Saturation Φ_S is a finite list of declared regions $\mathcal{S}_j = \{\mathbf{u} : H_j \mathbf{u} \le \mathbf{h}_j\}$, each with a declared behavior $\beta_j \in \{\mathsf{clip}, \mathsf{scale}, \mathsf{hold}, \mathsf{reject}\}$. Outside the declared regions the interface must not saturate, which is expressed through the saturation defect

$$\begin{aligned} &s_m(\mathbf{u}) = \|g_m(\mathbf{u}) - (K_m \mathbf{u} + \mathbf{c}_m)\|_\infty \le \varepsilon_S \\ &\forall\, \mathbf{u} \in \mathcal{U}_{\text{nom}} \setminus \textstyle\bigcup_j \mathcal{S}_j. \end{aligned} \tag{8}$$

Inside $\mathcal{S}_j$, the realized response must match β_j; for example, clip requires $g_m(\mathbf{u}) = \Pi_{\mathcal{F}}(\mathbf{u})$ up to the fidelity tolerance for a declared box $\mathcal{F}$.

Mode semantics Φ_M is a declared automaton $\mathcal{A}$ with mode set $\mathcal{M}$, initial mode m_0, a transition function $\delta(m, \mathbf{u})$ guarded by command predicates, and declared hysteresis widths $\mathbf{w}$, together with a statefulness flag stating whether the response may depend on command history beyond the current mode. An undeclared mode, an undeclared transition, or a hysteresis width outside its declared value violates Φ_M.

Drift tolerance Φ_D bounds slow changes of the operating point, such as those caused by payload or battery state, by a magnitude $\boldsymbol{\mu}^{\max}$ and a rate $r^{\max}$. Drift within these bounds is absorbed; drift outside them is a violation.

Table 1 summarizes the clause types, the model quantity each constrains, and the corresponding runtime evidence.

3.4 Verification problems

Two problems are posed. *Offline verification*: given $\mathcal{C}$ and a calibrated model $(\hat{\theta}, P)$, decide for each clause whether it holds for all $\theta \in \Theta_\alpha$, fails for all $\theta \in \Theta_\alpha$, or neither, and in case of failure return a witness command. *Runtime monitoring*: given the streams $\mathbf{u}_{0:k}$ and $\mathbf{y}_{0:k}$, raise an alarm attributed to a clause as soon as the evidence that the live interface violates it exceeds a threshold chosen for a target false-alarm rate, and classify each violation as a transient event or a standing condition.

04 Method

4.1 System architecture

The navigation stack considered here has the usual layered structure. A global planner computes a path on a terrain map; a local controller, for example a dynamic-window or MPPI controller [11, 12], produces $\mathbf{u}_k$ at its own rate; and the vendor locomotion stack converts $\mathbf{u}_k$ into joint commands. The verification tool is a passive subscriber on the middleware bus. It reads the command topic and the state-estimator odometry,

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

Table 1: Clause types of a velocity interface contract (format v0.3).

Clause	Model quantity	Offline check	Runtime evidence
Fidelity Φ_F	Steady-state map g_m on $\mathcal{U}_{\text{nom}}$	Grid robustness margin with Lipschitz correction	Two-sided CUSUM on normalized residual
Latency Φ_L	Dead time d_m, response time $T_{m,i}$	Interval bound over Θ_α	Windowed response-time estimate with binomial exceedance test
Saturation Φ_S	Feasible set $\mathcal{F}_m$, defect s_m	Witness search outside declared regions	One-sided sequential test on projected shortfall
Mode Φ_M	Mode set, switching logic, hysteresis	Up-down ramp simulation through guards	Shadow automaton versus likelihood-inferred mode
Drift Φ_D	Slow change of $\mathbf{c}_m$	Not checked offline	EWMA offset with Page-Hinkley test

runs the runtime monitor at 50 Hz ($\Delta t = 20$ ms), and publishes clause-attributed alarms and standing-violation reports on a diagnostics channel. It never modifies commands. This choice keeps the tool outside the control loop, so that a monitor fault cannot destabilize locomotion, and it allows the same tool to run on platforms whose locomotion stack cannot be modified. The offline checker runs before deployment and after each firmware or calibration update, consuming the contract file and the calibrated model.

4.2 Contract format

Contracts are written in a small YAML-based declarative format, currently at version 0.3. A contract names the interface and the frame, declares the command components and units, and lists one section per clause type. The format is typed: each field has a declared unit and a declared shape, the parser rejects contracts whose saturation regions are not polyhedra in command space or whose mode guards reference undeclared quantities, and every contract carries the firmware version it was written for. Figure 1 shows an illustrative excerpt. The format deliberately excludes arbitrary expressions; guards are restricted to linear inequalities and norm bounds on the command, so that the offline checker can enumerate guard boundaries exactly.

```
contract: research_quadruped/cmd_vel
format: 0.3
firmware: gait-manager <version>
frame: base_link
command: [vx, vy, wz]      # m/s, m/s, rad/s
nominal_region:
  box: {vx: [-0.5, 1.0], vy: [-0.3, 0.3],
        wz: [-1.0, 1.0]}
fidelity:
  abs: [0.05, 0.05, 0.10]
  rel: [0.10, 0.10, 0.10]
latency:
  dead_time_max: 0.04      # s
  response_time_max: 0.08  # s
saturation:
  - region: {vx: [1.0, inf]}
    behavior: clip
modes:
  states: [stand, walk]
  initial: stand
  transitions:
    - {from: stand, to: walk,
       guard: "norm(u) > 0.05"}
    - {from: walk, to: stand,
       guard: "norm(u) < 0.02"}
  stateful: false
drift:
  max_offset: [0.10, 0.10, 0.15]
  max_rate: 0.02           # per minute
```

Figure 1: Illustrative excerpt of a contract in format v0.3. Apart from the 80 ms response-time bound used for the research quadruped in Section 5, the numeric values are placeholders and are not the deployed contract parameters.

4.3 Offline verification against a calibrated model

For each clause, the offline checker computes a robustness margin $\rho(\theta)$ that is positive when the clause holds for parameter θ and negative when it fails, in the spirit of quantitative temporal-logic semantics [21]. The three-valued verdict is

$$\text{verdict} = \begin{cases} \text{satisfied} & \text{if } \min_{\theta\in\Theta_\alpha} \rho(\theta) > 0, \\ \text{violated} & \text{if } \max_{\theta\in\Theta_\alpha} \rho(\theta) < 0, \\ \text{inconclusive} & \text{otherwise.} \end{cases} \tag{9}$$

An inconclusive verdict indicates that the calibration is not precise enough to decide the clause and is reported as a request for additional probing, which a planner-conditioned calibrator such as GAUGE [15] can target at the relevant command region.

Fidelity. The margin $\rho_F(\theta)$ is the minimum of $\varepsilon_i(\mathbf{u}) - |[g_m(\mathbf{u}) - \mathbf{u}]_i|$ over modes m, axes i, and commands $\mathbf{u} \in \mathcal{U}_{\text{nom}} \setminus \bigcup_j \mathcal{S}_j$. It is evaluated on a grid $\mathcal{G}$ with covering radius h in the infinity norm, including the vertices of $\mathcal{U}_{\text{nom}}$ and points on each guard boundary. If g_m is L_g-Lipschitz in the infinity norm, the integrand is Lipschitz with constant $L_g + 1 + \max_i \varepsilon_i^{\text{rel}}$, which gives the sound lower bound

$$\rho_F(\theta) \geq \min_{\mathbf{u}\in\mathcal{G},m,i} \left(\varepsilon_i(\mathbf{u}) - |[g_m(\mathbf{u}) - \mathbf{u}]_i|\right) - \left(L_g + 1 + \max_i \varepsilon_i^{\text{rel}}\right) h. \tag{10}$$

For the affine-then-projection map of the calibrated model, Euclidean projection onto a convex set is nonexpansive in the Euclidean norm, and the Euclidean and infinity norms on $\mathbb{R}^3$ differ by at most a factor $\sqrt{3}$, so $L_g \leq \sqrt{3}\,\|K_m\|_\infty$; the checker uses this constant. The minimum over Θ_α is approximated by evaluating the margin at $\hat{\theta}$ and at the endpoints of the principal axes of the ellipsoid, and the witness command is the grid point that attains the minimum.

Latency. Because $T_{m,i}$ is monotone in $a_{m,i}$, the extreme values over the ellipsoid are attained at $\hat{a}_{m,i} \pm \sqrt{\chi^2_{p,1-\alpha} P_{a_{m,i}}}$, where $P_{a_{m,i}}$ is the marginal variance, which gives the interval bound $\rho_L = \tau^{\max} - \max_{\theta\in\Theta_\alpha} T_{m,i}$ in closed form, evaluated separately for every mode, axis, and enumerated dead time.

Saturation. The checker searches for witnesses $\mathbf{u} \in \mathcal{U}_{\text{nom}} \setminus \bigcup_j \mathcal{S}_j$ with $s_m(\mathbf{u}) > \varepsilon_S$. Since s_m is zero on the interior of $\{\mathbf{u} : K_m\mathbf{u} + \mathbf{c}_m \in \mathcal{F}_m\}$, the search reduces to intersecting the preimage of the boundary of $\mathcal{F}_m$ with the undeclared part of the nominal region, which for polyhedral $\mathcal{F}_m$ is a set of linear feasibility problems. For declared regions, the checker evalu-

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

ates whether the model behavior matches the declared β_j on the vertices and on a grid of the region.

Mode semantics. The checker simulates the calibrated model along up-down ramps through each declared guard, at the command rates the planner is configured to produce, and records the switch points. A recorded hysteresis width that differs from the declared width by more than its tolerance, a mode identified in calibration that is absent from the contract, or a switch that occurs away from any declared guard yields a violation with the ramp as witness.

4.4 Runtime monitoring

Online, the monitor propagates the calibrated model in the mode predicted by the contract automaton to obtain a one-step prediction $\hat{\mathbf{y}}_k$ and forms the normalized residual

$$\mathbf{e}_k = \Sigma_m^{-1/2}\left(\mathbf{y}_k - \hat{\mathbf{y}}_k - \boldsymbol{\mu}_k\right), \tag{11}$$

where $\boldsymbol{\mu}_k$ is the current drift estimate described below. Under the contract and the calibrated model, $\mathbf{e}_k$ is approximately zero-mean with identity covariance. Each clause has its own test statistic and alarm rule, summarized in Table 2.

Fidelity: two-sided CUSUM. For each axis i, Page's cumulative sum [22] accumulates evidence of a mean shift of at least δ_F standard deviations:

$$\begin{aligned} S^+_{k,i} &= \max\Big(0,\; S^+_{k-1,i} + e_{k,i} - \kappa\Big), \\ S^-_{k,i} &= \max\Big(0,\; S^-_{k-1,i} - e_{k,i} - \kappa\Big), \end{aligned} \tag{12}$$

with reference value $\kappa = \delta_F/2$, and an alarm is raised when $\max(S^+_{k,i}, S^-_{k,i}) > h_F$. The residual alarm is attributed to Φ_F only if the corresponding steady-state error also exceeds $\varepsilon_i(\mathbf{u}_k)$; otherwise it indicates departure from the calibrated model while the contract still holds, and it is logged as a recalibration request rather than as a violation. The threshold h_F is set for a target in-control average run length using the standard approximations for Gaussian residuals [24].

Saturation: projected shortfall test. Saturation shows up as a shortfall of the realized velocity along the commanded direction. For commands in the undeclared part of the nominal region with $\|\mathbf{u}_k\| > u_{\min}$, the monitor computes

$$\begin{aligned} q_k &= \frac{\mathbf{u}_k^\top\left(\hat{\mathbf{y}}^{\mathrm{lin}}_k - \mathbf{y}_k\right)}{\|\mathbf{u}_k\|}, \\ \Lambda_k &= \max\left(0,\; \Lambda_{k-1} + \frac{\delta_S}{\sigma_q^2}\left(q_k - \varepsilon_S - \frac{\delta_S}{2}\right)\right), \end{aligned} \tag{13}$$

where $\hat{\mathbf{y}}^{\mathrm{lin}}_k$ is the model prediction with the projection in Eq. (3) removed, σ_q^2 is the residual variance along the command direction, and δ_S is the smallest shortfall of interest beyond ε_S. Λ_k is the log-likelihood ratio of Wald's test [23] for a Gaussian mean shift, in its repeated (CUSUM) form, and an alarm is raised when $\Lambda_k > \ln((1-\beta)/\alpha_S)$ for nominal error probabilities α_S and β. Because the test is directional, it distinguishes a speed limiter, which produces a persistent positive shortfall at high commanded speed, from symmetric tracking noise. Inside a declared region $\mathcal{S}_j$, the same statistic is computed against the declared behavior β_j instead of against the linear prediction.

Latency: windowed response-time test. Over a sliding window of W ticks with sufficient command excitation, the monitor re-estimates the response time on each axis i with the model structure of Eq. (2). For each candidate dead time $\ell \in \{0, \dots, L\}$, it forms the deviations $\tilde{y}_{j,i}(\ell) = y_{j,i} - [g_m(\mathbf{u}_{j-\ell})]_i$ and fits the pole by least squares,

$$\begin{aligned} \hat{a}_{k,i}(\ell) &= \frac{\sum_{j=k-W+1}^{k-1} \tilde{y}_{j+1,i}(\ell)\,\tilde{y}_{j,i}(\ell)}{\sum_{j=k-W+1}^{k-1} \tilde{y}_{j,i}(\ell)^2}, \\ \hat{T}_{k,i} &= \hat{\ell}_{k,i}\,\Delta t - \frac{\Delta t}{\ln \hat{a}_{k,i}(\hat{\ell}_{k,i})}, \end{aligned} \tag{14}$$

where $\hat{\ell}_{k,i}$ is the candidate with the smallest one-step prediction error and $\hat{a}$ is clipped to $[0, 1)$. Let $z^L_k = \mathbb{1}[\hat{T}_{k,i} > \tau^{\max}]$. Under the contract, exceedances occur with a probability at most p_0, determined from calibration data so that it also absorbs the estimator's bias and variance; an alarm is raised when the number of exceedances in the last N_L evaluated windows exceeds the $(1-\alpha_L)$ quantile of the binomial distribution $\mathrm{Bin}(N_L, p_0)$. Because measurement noise in $\tilde{y}_{j,i}$ biases $\hat{a}$ toward zero, the estimate tends to understate the response time in poorly excited windows; skipping such windows and calibrating p_0 empirically keep the test usable, at the cost of some sensitivity.

Mode semantics: shadow automaton. The monitor runs the declared automaton $\mathcal{A}$ on the command stream to obtain the contract mode $m^{\mathcal{C}}_k$, and in parallel infers the most likely realized mode from the mode-indexed calibrated models,

$$\hat{m}_k = \arg\max_{m \in \mathcal{M}} \sum_{j=k-W_M+1}^{k} \log \mathcal{N}\Big(\mathbf{y}_j;\, \hat{\mathbf{y}}^{(m)}_j,\, \Sigma_m\Big). \tag{15}$$

A violation of Φ_M is raised when $\hat{m}_k \neq m^{\mathcal{C}}_k$ for N_M consecutive ticks, or when no mode model explains the window better than a likelihood floor, which indicates an undeclared mode. Hysteresis that the contract does not declare appears as a run of ticks after a guard crossing in which the inferred mode lags the shadow mode.

Drift: EWMA with Page-Hinkley confirmation. Slow changes of the operating point are tracked with an exponentially weighted moving average of the unnormalized residual, $\boldsymbol{\mu}_k = (1-\lambda)\boldsymbol{\mu}_{k-1} + \lambda(\mathbf{y}_k - \hat{\mathbf{y}}_k)$, with forgetting factor $\lambda \in (0, 1)$. A Page-Hinkley test [24] on $\|\boldsymbol{\mu}_k\|$ confirms that a drift has occurred. A confirmed drift with $|\mu_{k,i}| \le \mu_i^{\max}$ and rate below $r^{\max}$ is absorbed into the operating point: $\boldsymbol{\mu}_k$ enters the residual in Eq. (11), so that the other tests are re-centered. A drift outside these bounds is reported as a violation of Φ_D. Re-centering is what prevents a legitimate change, such as a new payload, from producing repeated fidelity alarms; its cost is that alarms raised before the drift is confirmed are false positives, a trade-off examined in Section 5.

4.5 Events and standing violations

A violation that is present for most of the operating time is qualitatively different from one that appears intermittently. Repeating alarms for the former floods the diagnostics channel and trains operators to ignore it. For each clause, the monitor therefore maintains a long-window violation fraction

$$\bar{z}_k = \frac{1}{W_s}\sum_{j=k-W_s+1}^{k} z_j, \tag{16}$$

Table 2: Runtime tests used by the monitor, one per clause type.

Clause	Statistic	Alarm rule	Output
Fidelity	Two-sided CUSUM $S^{\pm}_{k,i}$ on $\mathbf{e}_k$	$\max(S^+, S^-) > h_F$ and error exceeds ε_i	Event, or recalibration request
Saturation	Projected shortfall q_k, log-likelihood ratio Λ_k	$\Lambda_k > \ln((1-\beta)/\alpha_S)$	Event with command witness
Latency	Windowed estimate $\hat{T}_{k,i}$	Binomial exceedance count over N_L windows	Event or standing violation
Mode	Shadow mode $m^{\mathcal{C}}_k$ versus inferred $\hat{m}_k$	Mismatch for N_M ticks or likelihood floor	Event with guard crossing
Drift	EWMA offset $\boldsymbol{\mu}_k$, Page-Hinkley statistic	Confirmed drift outside Φ_D	Absorb, or violation

where z_j is the clause's per-tick or per-window violation indicator. When $\bar{z}_k \geq \rho_s$, the clause is promoted to a *standing violation*: it is reported once, together with summary statistics of the violating quantity, and event alarms for that clause are suppressed until $\bar{z}_k$ falls below a lower release threshold. Standing violations indicate that the contract and the platform disagree by design, and their resolution is a contract revision or a platform change rather than an operational response.

4.6 Computational cost and design choices

With a three-dimensional command, the per-tick cost of the residual, CUSUM, shortfall, drift, and shadow-automaton updates is constant and small. The response-time estimate and mode likelihoods require sums over windows, which are maintained incrementally; the search over $L + 1$ candidate dead times in the response-time estimate is the dominant term. The design favors tests with closed-form updates over learned anomaly detectors for three reasons: alarms must be attributable to a specific clause, thresholds must be interpretable in terms of false-alarm rates, and the monitor must fit within a fraction of the 20 ms tick on the robot's onboard computer without a GPU. The choice of a first-order-plus-dead-time model per axis is a compromise; it is the simplest structure in which latency and saturation clauses are well defined, and the residual tests detect when it is inadequate.

05 Experiments

5.1 Platforms and protocol

The study used two platforms: a Unitree B2 quadruped and a research quadruped with 8 degrees of freedom and an open-loop gait manager. Both ran in the laboratory for 10 days of continuous operation, yielding 214 h of logged operation. Contracts were written in format v0.3, and the verification tool ran at 50 Hz on the middleware bus. For detection validation, three known but undocumented behaviors of the B2 firmware were seeded as ground-truth targets: a speed limiter, gait hysteresis, and payload compensation. These behaviors are absent from the vendor interface documentation and, by construction, absent from the B2 contract, so any engagement of them is a contract violation that the monitor should detect.

5.2 Baselines and metrics

Two baselines were compared with the model-based monitor. *No contract (implicit assumptions)* encodes the assumptions a navigation stack makes when no contract exists, namely unit gain, negligible lag, and no saturation or mode dependence within the planner's velocity limits, and checks the live interface against them. *Fixed-threshold monitoring (no model)* raises an alarm when the per-axis difference between commanded and realized velocity exceeds a fixed threshold for a fixed duration, without an interface model or statistical tests.

Four metrics are reported. Contract violation detection is the number of true-positive detections, that is, alarms matched to ground-truth violation episodes. The false positive rate is the percentage of contract-satisfying evaluation windows in which an alarm was raised. Monitor runtime is the 95th percentile of per-tick processing time. Time to detection is the median, over detected episodes, of the time from episode onset to the first alarm.

5.3 Main results

Table 3 reports the main comparison. The model-based monitor recorded 11 true-positive detections, compared with 10 for each baseline, and had the lowest false positive rate, 2.8% against 3.0% for both baselines. Its 95th-percentile runtime of 1.9 ms corresponds to 9.5% of the 20 ms tick at 50 Hz; the baselines required 2.2 ms and 2.0 ms. Its median time to detection was 3.4 s, which is 0.2 s shorter than the no-contract monitor (3.6 s) and 0.5 s shorter than the fixed-threshold monitor (3.9 s).

These differences are modest and should be read with care. The campaign is a single deployment, the counts are small, and the evaluation records no repeated trials from which confidence intervals could be computed; a difference of one detection, or of 0.2 percentage points in false positive rate, is not by itself evidence of a systematic advantage. The aggregate medians also hide the case that most clearly separates the methods, discussed in Section 5.5: on the first engagement of the B2 speed limiter, the model-based monitor detected the violation after 3.4 s and the fixed-threshold monitor after 41 s, roughly twelve times longer. What the results do support is that a monitor with an explicit contract and a calibrated model is at least as sensitive as the two simpler alternatives, does not raise more false alarms, and fits well within the real-time budget of the bus.

5.4 Ablation

Table 4 removes one component of the monitor at a time. Removing the saturation region check or the mode semantics each reduced true-positive detections from 11 to 9, and removing the statistical drift test reduced them to 10. The saturation and mode components therefore account for the largest share of detections in this campaign, which is consistent with two of the three seeded behaviors, the speed limiter and gait hysteresis, being saturation and mode phenomena. The drift test contributes one detection; its main role, discussed below, is to absorb legitimate operating-point changes.

5.5 Failure cases and recorded behaviors

Speed limiter on the B2. The undocumented speed limiter in the B2 gait manager was caught 3.4 s into its first engagement. The fixed-threshold monitor needed 41 s to flag the same engagement, and it never detected the gait hysteresis. The difference follows from the structure of the tests. A speed limiter produces a shortfall that grows with commanded speed but can remain

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

Table 3: Main comparison over 214 h of logged operation on both platforms.

Method	Contract violation detection (true positives)	False positive rate (%)	Monitor runtime p95 (ms)	Time-to-detection p50 (s)
No contract (implicit assumptions)	10	3.0	2.2	3.6
Fixed-threshold monitoring (no model)	10	3.0	2.0	3.9
Ours	**11**	**2.8**	**1.9**	**3.4**

Table 4: Ablation of monitor components (true-positive detections).

Variant	Contract violation detection (true positives)
Ours (model-based monitor)	11
w/o saturation region check	9
w/o mode semantics	9
w/o statistical drift test	10

below a fixed per-axis threshold at moderate speeds, so a fixed-threshold monitor must wait for a large command to cross its threshold for its full duration; the directional shortfall test accumulates evidence from every tick in which the command lies in the declared-unsaturated region. Gait hysteresis, in turn, produces no persistent tracking error at all, only a delayed mode change after a guard crossing, which a monitor without mode semantics cannot represent.

Payload compensation on the B2. The payload compensation behavior produced 6 false positives in the first 24 h of operation, before the drift test learned the new operating point. This is the expected cost of the design described in Section 4.4: until the Page-Hinkley test confirms a drift, the shifted residual is indistinguishable from a fidelity violation, and the CUSUM raises alarms. After the drift was absorbed, the remaining tests were re-centered. A faster drift estimator would reduce these false positives at the price of absorbing genuine slow violations more readily; the contract's drift bounds $\boldsymbol{\mu}^{\max}$ and $r^{\max}$ are the parameters that make this trade-off explicit.

Standing latency violation on the research quadruped. The research quadruped's open-loop gait manager violated the latency contract continuously, with a measured lag of 120 to 160 ms against the 80 ms bound, that is, 40 to 80 ms above the bound, or 1.5 to 2 times the bound. Because the violation fraction $\bar{z}_k$ remained above the promotion threshold, the tool reported it as a standing violation rather than as a stream of events. This is the intended behavior: the violation is not an operational anomaly but a mismatch between what the navigation stack assumes and what the platform delivers by design, and it calls for either a revised contract with a larger $\tau^{\max}$ and corresponding latency compensation in the local controller, or a change to the gait manager.

06 Toward Adoption in Legged Robot Middleware

The contract format is intended to be adopted incrementally. The following path is proposed, with each stage useful on its own.

1. *Declaration.* Platform vendors and integrators publish a contract file alongside each firmware release, versioned with it. Even without verification, a declared nominal region, latency bound, and list of saturation regions and modes replaces implicit assumptions with reviewable ones.
2. *Offline conformance.* After each firmware update, a calibration in the style of GAUGE [15] is run and the offline checker is applied. A violated or inconclusive clause blocks deployment of the update or of the contract, analogous to a failing test in continuous integration.
3. *Runtime monitoring.* The monitor runs as a node on the middleware bus; in ROS 2 [9], the contract can be distributed as a parameter of the velocity driver and published on a topic with transient-local durability, and violations can be reported through the standard diagnostics interface so that existing tooling displays them.
4. *Planner consumption.* Navigation frameworks such as Nav2 [10] and configuration-driven planners [38, 39] read velocity limits, latency bounds, and saturation regions from the contract rather than from separately maintained configuration, so that the planner's model of the actuator and the verified contract cannot diverge.

These stages suggest three conformance levels for a platform: declared, offline-verified, and runtime-monitored. The format itself is at version 0.3, and stabilizing it would require agreement on units, frames, and mode vocabularies across vendors, which is beyond what a single laboratory study can establish.

07 Limitations

The evaluation covers two platforms in a laboratory setting. The detection targets on the B2 were three known behaviors, seeded for validation; the study therefore measures the detection of behaviors whose existence was known in advance, not the rate at which previously unknown behaviors would be discovered in the field. The reported counts are from a single campaign without repeated trials, so the differences in Table 3 cannot be assigned confidence intervals, and the total number of ground-truth episodes is not reported, which precludes a recall estimate. The offline checker is described and used in the workflow but is not separately quantified here. The interface model is first-order per axis with a dead time and a static map; platforms with strongly coupled or higher-order dynamics may produce residual alarms that reflect model inadequacy rather than contract violation, and the recalibration-request path mitigates but does not eliminate this. The drift test trades early false positives for robustness to legitimate operating-point changes, as the payload case shows. Finally, the monitor is passive; it detects violations but does not enforce contracts or adapt the planner.

08 Conclusion

Velocity interfaces on legged robots carry assumptions about fidelity, latency, saturation, and mode behavior that navigation stacks depend on and that middleware rarely documents. This paper defined velocity interface contracts that state these assumptions in a typed declarative format, an offline checker that compares a calibrated interface model against a contract with a three-valued verdict, and a runtime monitor built from sequential statistical tests that distinguishes events from standing vio-

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

lations. On a Unitree B2 and an 8-DoF research quadruped over 214 h of operation, the monitor recorded 11 true-positive detections at a 2.8% false positive rate and a 95th-percentile runtime of 1.9 ms, caught the B2 speed limiter 3.4 s into its first engagement where a fixed-threshold monitor needed 41 s, and reported the research quadruped's 120 to 160 ms lag against an 80 ms bound as a standing violation. The margins over simpler monitors in aggregate metrics are small, and broader evaluation is needed; the clearer benefit is that violations become attributable to specific, reviewable clauses. A staged adoption path, from declared contracts to planner consumption of verified parameters, offers a route by which legged robot middleware could make the velocity interface an explicit, checkable boundary.

References

[1] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim. Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[2] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter. Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots. IEEE Robotics and Automation Letters, 3(3):2261-2268, 2018.

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

[4] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning Agile and Dynamic Motor Skills for Legged Robots. arXiv:1901.08652, 2019.

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

[6] 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.

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

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An Open-Source Robot Operating System. ICRA Workshop on Open Source Software, 2009.

[9] 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.

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

[11] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to Collision Avoidance. IEEE Robotics and Automation Magazine, 4(1):23-33, 1997.

[12] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information Theoretic MPC for Model-Based Reinforcement Learning. IEEE International Conference on Robotics and Automation (ICRA), 2017.

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

[14] 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.

[15] T. Zang, H. Huang, S. Cheng, and W. Zhang. GAUGE: Planner-Conditioned Active Calibration of Opaque Quadruped Velocity Interfaces. arXiv:2607.02841, 2026.

[16] B. Meyer. Applying Design by Contract. IEEE Computer, 25(10):40-51, 1992.

[17] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen. Contracts for System Design. Foundations and Trends in Electronic Design Automation, 12(2-3):124-400, 2018.

[18] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. European Journal of Control, 18(3):217-238, 2012.

[19] M. Leucker and C. Schallhart. A Brief Account of Runtime Verification. Journal of Logic and Algebraic Programming, 78(5):293-303, 2009.

[20] O. Maler and D. Nickovic. Monitoring Temporal Properties of Continuous Signals. International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), 2004.

[21] A. Donzé and O. Maler. Robust Satisfaction of Temporal Logic over Real-Valued Signals. International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), 2010.

[22] E. S. Page. Continuous Inspection Schemes. Biometrika, 41(1/2):100-115, 1954.

[23] A. Wald. Sequential Tests of Statistical Hypotheses. Annals of Mathematical Statistics, 16(2):117-186, 1945.

[24] M. Basseville and I. V. Nikiforov. Detection of Abrupt Changes: Theory and Application. Prentice Hall, 1993.

[25] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger. ANYmal - A Highly Mobile and Dynamic Quadrupedal Robot. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016.

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

[27] G. B. Margolis and P. Agrawal. Walk These Ways: Tuning Robot Control for Generalization with Multiplicity of Behavior. arXiv:2212.03238, 2022.

[28] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid Locomotion via Reinforcement Learning. arXiv:2205.02824, 2022.

[29] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. arXiv:1804.10332, 2018.

[30] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. arXiv:1710.06537, 2017.

[31] P. Fankhauser, M. Bloesch, and M. Hutter. Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization. IEEE Robotics and Automation Letters, 3(4):3019-3026, 2018.

[32] 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.

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

[34] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter. Navigation Planning for Legged Robots in Challenging Terrain. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016.

[35] D. Hoeller, L. Wellhausen, F. Farshidian, and M. Hutter. Learning a State Representation and Navigation in Cluttered and Dynamic Environments. arXiv:2103.04351, 2021.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010

[36] Z. Fu, A. Kumar, A. Agarwal, H. Qi, J. Malik, and D. Pathak. Coupling Vision and Proprioception for Navigation of Legged Robots. arXiv:2112.02094, 2021.

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

[38] 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.

[39] 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.

[40] L. Ljung. System Identification: Theory for the User. 2nd edition, Prentice Hall, 1999.

[41] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu. ROSRV: Runtime Verification for Robots. International Conference on Runtime Verification (RV), 2014.

[42] R. Isermann. Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance. Springer, 2006.

[43] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control Barrier Function Based Quadratic Programs for Safety Critical Systems. arXiv:1609.06408, 2016.

[44] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. arXiv:2108.06266, 2021.

KAGGLEYEs ACADEMY • E-PRINT REGISTRY • KAGG:2608.00010