A Rust-Native Embeddable Reactive Numerical Language for Heterogeneous Computing in Robotics

www.mech-lang.org
mech-lang/mech

Corey Montella, Aung Kant Ko, Steven McPhillimey

Motivation

Rust's memory safety and performance make it a strong foundation for robotics. Still, numerical kernels can require substantial code for SIMD layouts and GPU execution. This code can obscure the equations, and each specialized version must be verified and maintained.

Mech is an open-source, Rust-native embeddable numerical language. For robotics, its reactive execution handles sensor updates and retains accepted state after rejected turns. The EKF results below show SIMD throughput comparable to Rust.

Embedded numerical kernels are one use of Mech, which also has first-class state machines and pattern matching. **Scan the QR at the bottom for a demo.**

Embeddable

Mech runs standalone or embeds in Rust, from a single numerical function to a reactive subsystem. Rust hosts can compile Mech source or load a dylib, and browser applications can embed it through WebAssembly.

JIT compilation from Rust

```rust
use mech::kernel::{Backend, Kernel};
let source = include_str!("ekf.mec");
let kernel = mech::mech!(source)
    .input("bearing", [-0.55; 4])
    .export("state")
    .compile(Backend::Jit)?;
let mut ekf = kernel.start()?;
ekf.turn([("bearing", [-0.54; 4])])?;
let state = ekf.state("state")?;
```

Compile Mech source from a file or inline macro, then submit updates as they arrive. Calls run synchronously.

Checked dylibs†

Artifact	Bytes	M/s ± MAD	Peak RSS (MiB)
Mech scalar	33,544	14.671 ± 0.008	2.69 ± 0.00
Rust scalar	50,016	21.121 ± 0.005	2.69 ± 0.00
Mech SIMD-4	33,864	34.863 ± 0.010	2.69 ± 0.00

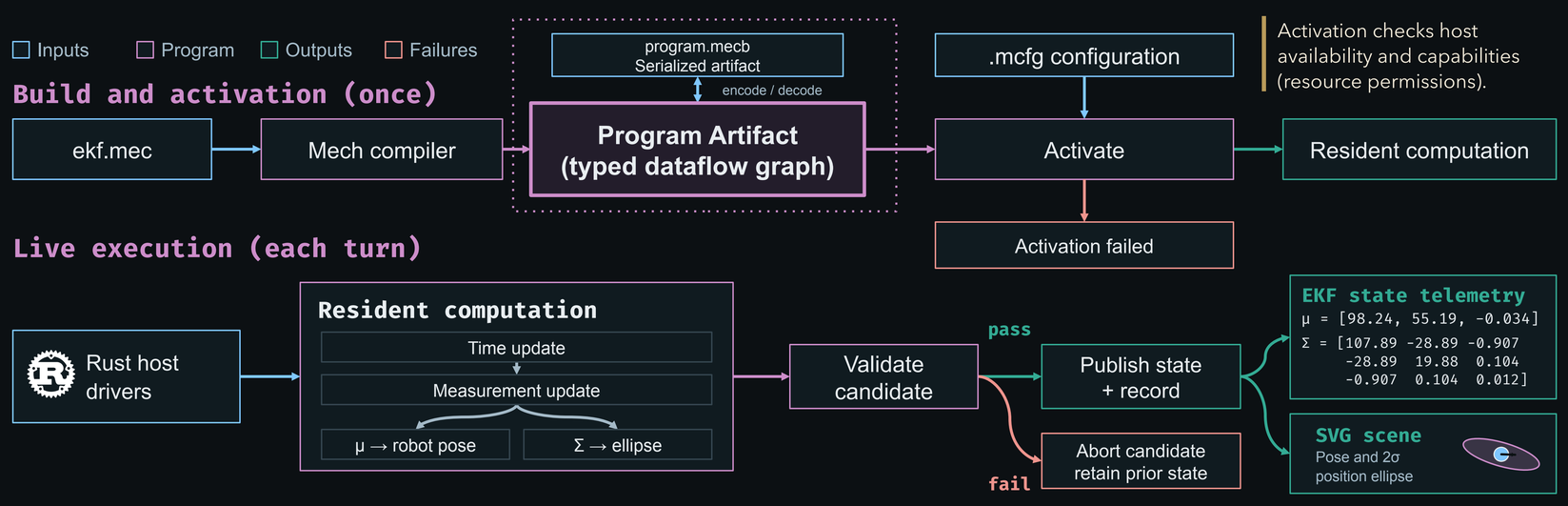
Load an AOT library

```rust
let kernel = unsafe { Kernel::load_bundle("ekf.bundle")? };
let mut ekf = kernel.start()?;
ekf.turn([("bearing", [-0.54; 4])])?;
```

The bundle stores the dylib and metadata for named inputs, exported state and initial values.

Reactive

Mech models robot feedback loops over continuous sensor streams from Rust host drivers, through direct interfaces or ROS. Each update triggers a turn that computes and validates candidate state. Rejection notifies the caller and preserves prior state.

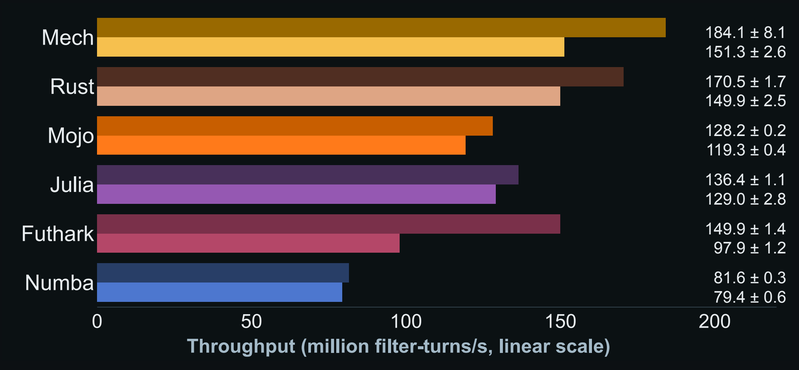


Heterogeneous

Mech's typed graph targets CPU/GPU and WebAssembly/WGPU. The right chart runs the same f32 EKF on five backends without source changes. Integrity checks add cost but can be disabled to increase throughput. The left chart compares a similar workload across languages used for numerical computing.

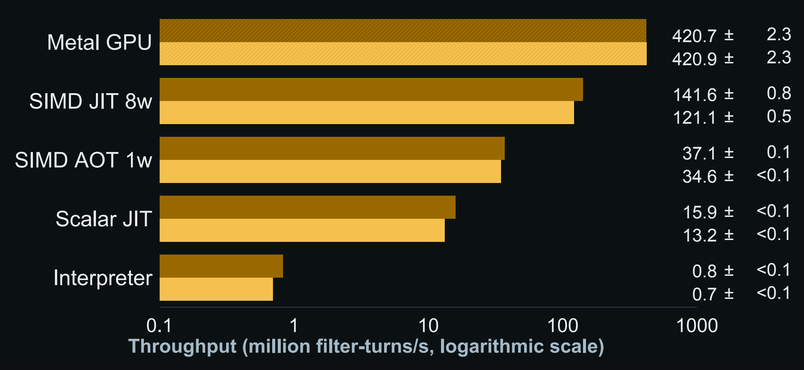
Mech and Rust show similar throughput‡

Median ± MAD



Backend choice changes EKF throughput

Median ± MAD



SIMD JIT and AOT share the numerical compilation pipeline.
1w/8w: one/eight workers. Worker scaling depends on host support.
Performance specialists may further optimize these implementations.

Numerical

Mech offers a concrete syntax for typed matrix equations, illustrated by the bearing-only EKF below.

The kernel is shorter than both the textbook-style Rust implementation and the SIMD Rust implementation. SIMD Rust adds vector and batch support, while Mech reuses one numerical kernel across backends.

Mech compiles this syntax to a typed dataflow artifact that can run in a nalgebra-based interpreter or compile for different host architectures without source changes or backend-specific annotations.

```
Extended Kalman Filter
═══════════════════════
+> math/*, logic/all
 Δt := 0.1
 m := [140 12]'
 Q := [0.01 0       -- +> import library
       0    0.0025] -- ~  mutable
 R := 0.25          -- := define
 I := [1 0 0        -- =  assign
       0 1 0        -- ** matrix multiply
       0 0 1]       -- '  transpose
 ~μ := [55 25 0.4]' -- *  broadcast multiply
 ~Σ := [100 0   0   -- x! integrity constraint
        0   100 0
        0   0   0.15]

Time Update
────────────
 θ := μ[3]
 d := v * Δt
 G := [1 0 (0 - d * sin(θ))
       0 1 d * cos(θ)
       0 0 1]
 V := [cos(θ) * Δt 0
       sin(θ) * Δt 0
       0          Δt]

 μ̄ := μ + [d * cos(θ)
           d * sin(θ)
           ω * Δt]
 Σ̄ := G ** Σ ** G' + V ** Q ** V'

Measurement Update
──────────────────
 δ := m - μ̄[1..=2]
 δx := δ[1]
 δy := δ[2]
 q := δx ^ 2 + δy ^ 2

 ẑ := atan2(δy, δx) - μ̄[3]
 r := z - ẑ
 v := atan2(sin(r), cos(r))

 H := [δy / q, (0 - δx) / q, -1]
 S := H ** Σ̄ ** H' + R
 K := (Σ̄ ** H') / S[1]
 A := I - K ** H

 μ₊ := μ̄ + K * v
 Σ₊ := A ** Σ̄ ** A' + (K ** K') * R

 μ = μ₊
 Σ = Σ₊
 variance! := all(Σ₊[[1 5 9]] > 0)
```

Normalized◆ source characters

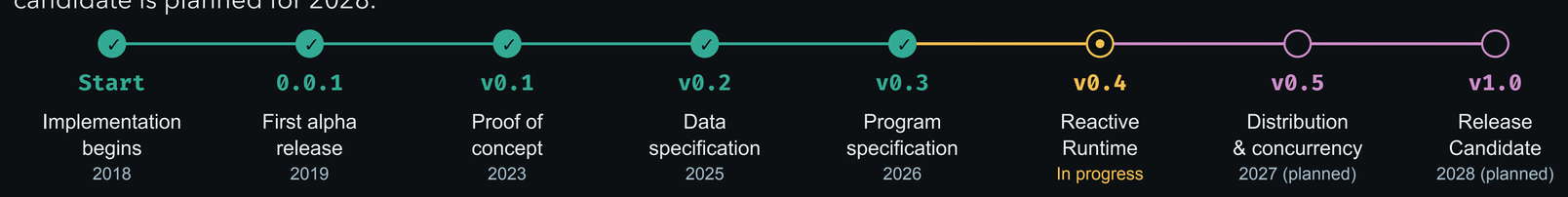
Application source	Mech	Rust textbook	Rust SIMD
Setup	340	486	561
EKF + candidate update	356	1,024	1,119
Validation	383	263	396
Matrix helpers	0	251	310
SIMD + batch support	0	331	2,857
Total	1,079	2,355	5,243

The EKF and candidate update alone use about three times as many normalized characters in Rust (1,024 or 1,119) as in Mech (356), before counting helpers or batching.

SIMD and batch support adds 2,857 characters, over half the Rust SIMD source. Mech handles these operations in reusable compiler and runtime components.

Status and Future Work

Mech v0.4-beta needs further validation and hardening. Over the next year, we plan to expand research-robot deployments and use feedback from users to guide work on additional hosts, an expanded standard library and language features. A v1.0 release candidate is planned for 2028.



Experience this poster as a live reactive program: CPU/WGPU in WebAssembly.
Contributions and bug reports are welcome on GitHub: mech-lang/mech.

mech-lang.org/iros-r4r-2026/

‡ Benchmark setup. Apple M1 (4P + 4E, 8-core GPU, 8 GB), macOS 15.6.1. f32, 500,000 filters × 40 turns, n=10. CPU: 8 workers, fused turns. Mech backends: per-turn publication. Rust 1.96.0-nightly ec818fda3/LLVM 22.1.0. Cranelift 0.131.3. Julia 1.12.6. Mojo 1.1.0/MAX 26.6.0. Futhark 0.27.1/ISPC 1.31.0/LLVM 22.1.8. Python 3.12.14/NumPy 2.2.6/Numba 0.61.2. Clang 17.0.0. Mech v0.4 development (package 0.3.6). Backend pairs f4b69052c, CPU eedc1c75b, dylibs dab98af1d. iros-workshop-benchmarks, benchmarks/iros-2026.

† Dylibs: minimal native-ABI hosts, 1 worker, 10,000 filters × 200 turns, n=7. Whole-process peak RSS. Timing excludes compilation, loading and warmup.

◆ Programmer-chosen names count as one character per occurrence. Comments and nonliteral whitespace are removed. General library, compiler and runtime internals are excluded.

LEHIGH UNIVERSITY