Appears in the Proceedings of the 2025 IEEE Int'l Symp. on Workload Characterization (IISWC), October 2025

EntoBench: A Benchmark Suite and Evaluation Framework for Insect-Scale Robotics

Derin Ozturk, Nick Cebry, Angela Cui, Hang Gao, Julie Villamil, E. Farrell Helbling, Christopher Batten
Cornell University, Ithaca, NY

***Abstract*—EntoBench is the first open, MCU-ready benchmark suite and evaluation framework that captures the full insect-scale robot pipeline. Thirty-one kernels, each configurable for float, double, or fixed-point arithmetic, map how milliwatt power budgets and tight memory constraints can reshape algorithmic trade-offs. A synchronized GPIO harness couples logic-analyzer timing with inline current sensing, allowing any Cortex-M0+, M4, M33, or M7 board to report latency, energy, and peak power under cache-on/off condition and varied kernel parameters. More than 400 experiment runs reveal systematic patterns linking architecture features to achievable autonomy, providing a rigorous baseline for future software optimization and hardware-software co-design in this emerging cyber-physical domain. The full framework and benchmark suite are released as open source.**

I. Introduction

Insect-scale robots, typically characterized by lengths under 5cm and masses below 5g, are a rapidly growing area of robotics research. These platforms promise transformative capabilities in fields such as search-and-rescue and environmental monitoring. At these scales, familiar physical intuitions begin to break down: scaling laws introduce new constraints on actuation, sensing, and control, requiring roboticists to employ micro-intuition [28] and look towards biology for inspiration [18] in the robot design process. The effect is an explosion of diversity across demonstrated systems (e.g., flyers [10, 14, 36, 37, 56, 68], crawlers [3, 14, 32, 41, 54, 61, 69], jumpers [2, 8, 39, 61], swimmers [60, 67], gliders [22, 38, 55], and striders [27, 62, 64]) reflecting a wide range of form factors, actuation strategies, and control architectures tailored for operation at the insect scale.

A major trend in recent years is the push toward full autonomy in insect-scale robots, encompassing *sensor*, *actuator*, *power*, and *compute* autonomy. *Sensor autonomy* refers to the robots ability to carry its own sensors to sense its own state and the state of its environment without external infrastructure. *Actuator autonomy* requires onboard actuators and drive circuitry that enable meaningful, taskable locomotion through the right mechanical design and signal generation. *Power autonomy* refers to untethered operation using onboard energy sources such as batteries, solar cells, or other harvesting approaches. Finally, *compute autonomy* entails the robot's ability to carry processors capable of running the full sensing-to-actuation pipeline within the robot's constraints and application requirements. Each of these pillars of autonomy should be viewed as a spectrum, with robots occupying different points which are heavily affected by size, weight, and power constraints. While most demonstrated insect-scale systems currently rely on external position tracking, off-board computation, and tethered power sources, next-generation platforms aim to be self-sufficient: sensing and understanding their environment and internal state, making control decisions in real time, and doing so under tight size, weight, power, and timing constraints. Among these four pillars of autonomy, we argue that *compute autonomy* is the most critical to address first. Processor selection has recently been emphasized for its influence on algorithmic feasibility and efficiency in insect-scale robots [18]. The choice of onboard compute directly determines what sensing and control strategies are feasible and what power budget is sustainable, setting the stage for a virtuous robot-hardware-software co-design loop. Furthermore, optimized compute systems may unlock new capabilities for these robots, beyond enabling operation outside the lab.

The challenge of *compute autonomy* for insect-scale robots likely necessitates a multi-faceted approach, involving custom hardware, innovative hardware-software co-design, and robotics algorithms heavily optimized for extreme resource constraints. Achieving this requires a rigorous and accurate assessment of computational workloads. However, current practices in the insect-scale robotics field often lack the detailed, measurement-driven characterization common in computer systems research. For instance, relying on high-level metrics such as Floating Point Operations (FLOPs) per second, a common practice for assessing both computational efficiency and power efficiency in some recent insect-scale robotics literature [25, 65, 70], can be misleading. Indeed, applying such FLOP counting approaches to workloads representative of those in advanced insect-scale systems can lead to underestimations of actual cycle counts on target microcontroller units (MCUs) by as much as 79.84% and 80.87% for the workload studied in [51], and also fails to address that average power consumption (estimated from FLOPs and the datasheet) is not adequate for comparing efficiency of different workloads, underscoring the critical need for direct measurement and characterization on target hardware.

To enable meaningful progress in tackling compute-autonomy on these constrained systems, we need benchmark suites and evaluation frameworks that reflect the realities of these insect-scale platforms. Existing robotics benchmark suites [5, 6, 9, 44] do not meet these needs for several reasons (see Table I). First, they do not reflect current insect-scale robotics algorithms or pipelines. Second, they assume an abundance of compute resources and software stacks that are impractical for insect-scale deployments. Third, their modularity and extensibility are limited in practice; for example, some

TABLE I
COMPARISON OF ROBOTICS BENCHMARK SUITES

Characteristic	MAV Bench	Robot Perf	RTR Bench	Ro Wild	Ento Bench
Insect Scale	✗	✗	✗	✗	✓
Resource Constrained	✗	✗	✗	✗	✓
Modular & Extensible	✓	✓	✓	✓	✓
Energy & Power Focused	✓	✓	✗	✗	✓
End-to-End	✓	✗	✗	✓	✗

suites simply aggregate open-source projects with disparate build systems, making it difficult to integrate new kernels or target new hardware platforms uniformly. Fourth, they neglect energy as a first-class metric, measuring it only coarsely (e.g., system-level average power) or not at all for individual compute kernels. Lastly, while some suites do not evaluate full end-to-end deployments, we view this as an important future direction. Since such deployments remain rare at the insect scale, we focus on individual kernels for this current work.

In this work, we introduce **EntoBench**, an **MCU-ready, open benchmark suite and evaluation framework** tailored for **insect-scale robotics**. EntoBench is MCU-agnostic, modular and extensible: each kernel is a stand-alone C++ template that can switch between float, double, or fixed-point arithmetic, and users can easily add new workloads or iterate on existing ones. A synchronized GPIO harness aligns logic-analyzer timestamps with inline current sensing, so any ARM Cortex-M board can report latency, energy, and peak power, all without altering application code.

Building on this infrastructure we contribute three things and demonstrate their value through four deployment-driven case studies. First we release the evaluation framework itself, ready to benchmark on STM32-based microcontrollers and adaptable to other commercially available development boards. Second, we provide a curated suite of 30 perception, estimation and control kernels each with template-based parameterization and some with variants. Some of these workloads have not been demonstrated on Cortex-M class hardware before to our knowledge. Third, we deliver a cross-platform characterization over 400 measured datapoints that exposes how caches, floating point units, memory footprints, and numeric formats steer feasibility in ways FLOP tallies cannot capture. We then show how EntoBench can be used in four case studies that (i) prove vision kernels under tight exteroception budgets, (ii) map the tradeoffs in precision versus energy for high-rate inertial filters, (iii) test whether static FLOP counts predict real sensor-fusion cost, (iv) explore how motion-aware minimal solvers reshape the robustness-efficiency balance in visual pose estimation. Finally, we sketch a path from kernel timings to full closed-loop benchmarks for end-to-end system evaluation. Together these elements turn raw measurements into actionable guidance for next-generation insect-scale robots.

II. RELATED WORK

To motivate the need for EntoBench, we review two lines of prior work in benchmarking and evaluation: robotics specific suites, which model realistic pipelines but assume far richer hardware than insect-scale MCUs, and general embedded benchmark suites, which target MCUs yet ignore robotics workloads.

A. Robotics-Oriented Benchmark Suites

A growing number of robotics benchmarking suites target realistic workloads and pipelines, but they assume far more compute, memory, and software infrastructure than is feasible for insect-scale platforms. These suites emphasize system-level evaluation but largely overlook the microcontroller-level constraints relevant to our setting.

MAVBench couples an AirSim-based closed-loop simulator with an application suite that measures perfomance, power, and flight endurance for micro-aerial vehicles running on Jetson-class SoCs [9]. The workloads consider include tasks such as 3D mapping and other computational kernels that are currently not realistic for insect-scale hardware. Although the work links compute to battery life, it evaluates kernels using ROS and reports only board-level power, leaving unanswered questions on how individual algorithms behave in latency and energy on insect-scale hardware.

RobotPerf standardizes ROS 2 computational graphs across CPUs, GPUs, and some FPGAs and publishes throughput, latency, and average power numbers using a vendor-agnostic harness [45]. The emphasis on middleware portability is valuable for larger robots, yet its focus on computational graphs on more powerful hardware, and its reliance on the ROS 2 ecosystem, differs significantly from the realities of deployment on insect-scale hardware.

RTRBench and its sucessor, **RoWild**, are robotics benchmark suites built for computer architecture studies [5, 6]. RTRBench provides 16 real-time kernels and analyzes them in `zsim`, an x86 simulator. RoWild expands the set, executes on real CPUs and GPUs, and places emphasis on 5 different case studies, each resembling a deployment on a different robot. Both ship one representative workload per task and omit direct energy measurement, arguing that energy and power consumption for larger robots is dominated by actuation.

B. General Embedded Benchmark Suites

Beyond robotics-specific suites, several benchmarks target general embedded systems, with some specifically focusing on energy consumption. EntoBench draws inspiration from their methodologies while specializing in the distinct challenges of insect-scale robots.

ULPMark (EEMBC) [24] assesses ultra-low-power MCUs via energy-per-operation scores across CPU and peripheral profiles. While effective for general ULP benchmarking, it lacks robotics-specific workloads and fine-grained task characterization. **Embench** [53] replaces outdated synthetic benchmarks (e.g., Dhrystone) with representative C programs. Embench-IoT focuses on IoT tasks, while Embench-DSP adds floating-point signal processing kernels (e.g., IIR and FIR filtering). **BEEBS** [52] targets energy measurement, using GPIO toggling to delimit benchmarks for external power monitoring—an approach mirrored by EntoBench. However, BEEBS workloads are drawn from generic suites. **MLPerf Tiny** [7] benchmarks TinyML workloads (e.g., keyword spotting, wake word detection) across software and hardware stacks. Its emphasis on system-wide energy measurement is valuable, but its tasks remain in the ML inference domain. Despite their contributions, none of these suites adequately address the computational patterns or constraints of robotics applications, especially those operating at the insect scale.

The insights and methodologies from these general and energy-aware embedded benchmarks have informed the design of EntoBench. However, EntoBench's primary contribution lies in its specialized focus on the unique workload characteristics and evaluation needs of insect-scale robotics, providing a tailored suite that addresses the specific software and hardware co-design challenges in this resource-starved domain.

III. Compute Autonomy for Insect-Scale Robots

While general-purpose robots benefit from powerful processors, modular software stacks, and generous energy budgets, insect-scale robots have severe size, weight and power (SWaP) constraints that impose strict limits on sensing, actuation, compute, and power. Research on these systems has primarily focused on mechanism design and locomotion strategies, with simple point solutions for proprioceptive sensing and control, and open-loop power strategies. Due to the constraints these robots face, the insect-scale robot computational pipeline generally looks different from that of larger robots, focusing on the classical feedback control loop (see Figure 1) and not considering computational stages such as semantic scene understanding, localization, mapping, collision avoidance, or planning. Onboard MCUs need to (1) interface with sensors, (2) compute robot pose, (3) update control parameters, and (4) generate actuation signals. In addition to tightening SWaP constraints, as we scale down robots, mass drops faster than structural stiffness, which causes the natural resonant frequencies of the robot body to increase. For example, in [32], when scaling down a crawling inset-scale robot from 45.1mm and 1.41g to 22.5mm and 0.32g, the optimal stride frequency tripled. In other words, insect-scale robots have faster dynamics and require faster control loop and and state estimation update rates, consequently requiring faster computation and clock speeds [30, 36, 71].

Onboard sensors need to sense the robot's state (*proprioception*) and external environment (*exteroception*). Sensors for insect-scale robots include inertial measurement units (IMUs [26, 33, 65]), time-of-flight range finders (ToF [23, 34, 65, 70]), low-resolution optic flow vision sensors [23, 65], or high resolution cameras [51]. These sensors have varying communication interfaces (I2C, SPI, analog), and feedback rates (10Hz-2kHz generally, reaching upwards of 50 MHz for cameras), which are important metrics for any

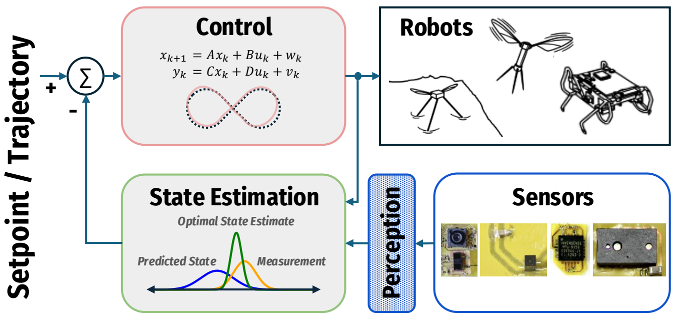


Fig. 1. The Insect-Scale Robot Pipeline – Extreme size, weight, and power constraints have generally restricted insect-scale robots to considering computation only for high-rate feedback control.

onboard compute system. Estimation algorithms (e.g., linear interpolation, Kalman Filtering, Extended Kalman Filtering) then compute pose estimates from raw sensor data. These estimation algorithms require varying levels of compute (e.g., matrix inversion, floating point math, non-linear optimization). Insect-scale controllers range between model-free PID control, linear-time invariant controllers, nonlinear adaptive controllers, and model predictive controllers, depending on the application and the required feedback rate, accuracy and precision of the robot state [11, 12, 19, 21, 23, 25, 26, 29, 33]. Finally, these control parameters need to map to motor control signals, which either vary PWM or PFM signals [36, 46]. An onboard MCU needs to have sufficient inputs and outputs, numerical precision, and loop frequency to meet the requirements of this pipeline. Runtime is also of critical importance, and onboard batteries can be limited to 1.8 MJ/kg [69].

Commercial off-the-shelf MCUs that meet the SWaP requirements of these robots are limited to 8- or 32-bit Atmel AVR chips or ARM Cortex-M cores. These existing systems operate without an operating system, sometimes with no hardware floating-point unit, and under strict power and memory budgets. We focus our work on 32-bit ARM Cortex-M cores, as 8-bit cores are not practical for robotics workloads. Available SRAM ranges anywhere from ˜32KB to ˜2MB, limiting onboard sensor data. Additionally, power consumption scales with clock frequency squared, and these microcontrollers have flexible, internal clock generation mechanisms that can scale compute power to fit requirements during active periods and low-power sleep modes for inactive periods. Mapping the sensing, estimation, and control algorithms onto Cortex-M processors demands budgeting for heterogeneous memories (flash, SRAM, tightly coupled memory), optional instruction and data caches, and the presence, or absence, of FPUs and DSP extensions.

Executing the entire insect-scale pipeline within the severe SWaP constraints of these robots remains a formidable challenge, and consequently most published prototypes still rely on off-board processors and tethers for computation. This gap underscores the importance of a rigorous and reproducible evaluation framework that targets suitable insect-scale hardware.

IV. EntoBench

To rigorously evaluate algorithms under the constraints of insect-scale robotics, we introduce EntoBench, a modular framework and benchmark suite designed for real-time execution on microcontrollers. EntoBench emphasizes realistic workloads, hardware-awareness, and configurability, enabling better system-level design and exploration, and sets the stage for future work in hardware-software-robot co-design. This section begins by first outlining the 5 design goals that guide the creation of EntoBench, then describing the structure of the framework and it key components, and finally presents the curated benchmark suite organized by pipeline stage.

A. Design Goals

EntoBench is designed around five core principles that address the unique requirements of insect-scale robotics evaluation:

Representative Pipeline Coverage. EntoBench targets the main pipeline stages of current insect-scale autonomy—perception, state estimation, and control—omitting mapping and planning, whose footprints exceed on-chip limits and have not been explored at this scale. Kernels are drawn from published work or the immediate research frontier.

Suitable for Resource-Constrained Platforms. Benchmarks must assume no external memory, and limited SRAM/FLASH. They must avoid dynamic allocation, virtual functions and heavy libraries; template metaprogramming should handle compile-time variants to take advantage of known parameters and operating conditions.

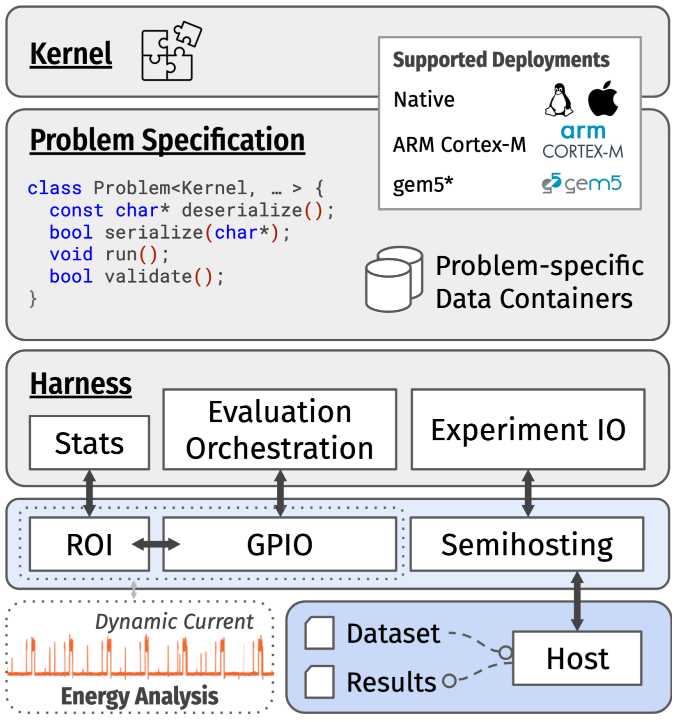
Modular and Extensible Design. Each kernel should be implemented as a standalone component with minimal dependencies, exercised through a common harness, and include unit tests. Users should be able to swap precision modes, test new kernels, or compose pipelines with minimal changes.

Treat Energy and Peak Power as First-Class Metrics. The framework must report both cumulative energy and instantaneous peak power, because on insect-scale robots energy per kernel sets mission endurance during untethered operation, while bursts in current consumption can brown-out custom made power electronics.

Forward-Looking and Evolvable. The framework must be designed to evolve alongside the fast-moving landscape of insect-scale robotics. As new sensors, actuators, algorithms, and platforms emerge, the suite's modular structure should allow for workloads to be updated, replaced, or expanded, mirroring Embench's call for sustainable, relevant benchmarking in dynamic domains [53].

B. Evaluation Framework

The EntoBench evaluation framework, illustrated in Figure 2, keeps benchmark code and measurement infrastructure strictly separate, enabling easy reuse and extension. Compile time templates are used thoughout the codebase for specialization and inlining. Table II shows the different parameters EntoBench supports across different categories. At the heart of EntoBench



```
class Problem<Kernel, … > {
  const char* deserialize();
  bool serialize(char*);
  void run();
  bool validate();
}
```

Fig. 2. Overview of the EntoBench Framework

are the following components, which work in concert to enable rigorous benchmark definition, execution, and measurement:

Kernels. Each workload is a stand-alone C++ template parameterized by algorithm variant, scalar type (float, double, fixed-point), and dimension, where applicable. Kernels depend only on Eigen and sometimes various specialized utility functions that are useful for workloads that are from the same general category. We support a custom fixed-point type with full Eigen integration, enabling easy deployment on microcontrollers without FPUs.

Problem Specifications. Object-oriented problem specifications that employ the *curiously recurring template pattern* wrap kernels into fully evaluable tasks via the `EntoProblem` base class interface. This interface defines how inputs are synthesized or loaded, how the kernel is invoked `solve()`, and how results are validated `validate()`. It also encodes metadata like dataset needs, such as `RequiresDataset` (useful for microbenchmarking) and `SavesResults` (to reduce interaction with the host computer), and integrates with `ExperimentIO` for file I/O. This abstraction allows kernels to be reused across simple synthetic tasks or realistic pipelines with recorded sensor data.

Harness. A generic driving harness instantiates problems, toggles region of interest GPIOs (`start_roi`, `end_roi`) and orchestrates benchmarks based on user parametrizable cache warm up and repetitions parameters. Our limited support in the microarchitectural simulator `gem5` supports setting regions of interest with conditionally compiled code using the same toolchain for MCU builds and deployed on `gem5`'s provided ARM processor model.

TABLE II
PARAMETER CATEGORIES AND EXAMPLE PARAMETERS

Category	Parameters
Harness	Repetitions, Cache Control Verbosity, Data I/O Configuration
General	Scalar Precision, Data Types, Tolerances
Perception	Image Dimensions, Feature Counts Detection Thresholds, Window Sizes
State Estimation	Model Dimensions, Sensor Configuration, Filter Types, RANSAC Configuration
Control	Prediction Horizon, System Size, Convergence Criteria

MCU Abstraction Layer. We provide a lightweight hardware abstraction layer for system clock configuration, on-chip cache management, cycle counting, and GPIO pin toggling. These GPIOs are critical for EntoBench's energy and timing instrumentation, allowing external logic analyzers and power monitors to track precise execution intervals.

Energy Measurement Setup. We combine a Saleae Logic 2 analyzer and a STLINK-V3PWR current probe to measure dynamic power during execution. A dedicated GPIO trigger pin initiates the STLINK-V3PWR recording, while a separate latency pin marks ROI boundaries. Sampling at 100 kHz with 50 nA resolution, this setup enables synchronized and repeatable energy and peak power measurements. A Python script is used to synchronize both data logs to yield latency, energy, and peak-power per run.

Build System. EntoBench has a modular CMake build system that targets (1) native Linux/Mac, (2) Cortex-M via the ARM GCC Embedded Toolchain and OpenOCD, and (3) gem5 (limited support, no validated MCU models). New board support is handled via thin CMake tool-chain files. Semihosting plays the key role of moving data between host and MCU. All benchmarks can be configured via JSON files that our build system uses for build-time parameters such as `Reps`, `Verbosity`, and `TotalRuns`.

C. Benchmark Suite

EntoBench ships 31 microcontroller-ready kernels spanning perception, state estimation, and control, each lifted from published insect-scale work or its immediate frontier. Table III lists the kernels in current suite with full static metrics that are expanded in Section V. Table IV accompanies it listing dynamic metrics for the same kernels.

Perception. Three corner detectors with descriptors, `fastbrief` [35, 57], `orb` [58], and SIFT [43], cover the invariance and compute spectrum. Motion tracking via optical flow is represented by iterative Lucas Kanade [4], (`lkof`), image interpolation [63] (`iiof`), and brute force block matching [1] (`bbof`). Together these provide building block towards visual(-inertial) odometry, place recognition, and SLAM.

State Estimation. High-rate attitude filters (`mahony`, `madgwick`, `fourati`) are included and support float, double, and fixed point formats. Sensor fusion is covered by two recently proposed EKFs for insect-scale flapping wing aerial vehicles: a 4-state RoboFly filter [65] that fuses asynchronous time-of-flight (ToF), optical flow, and IMU data, and a 10-state RoboBee EKF that fuses ToF and IMU data. Our generic EKF wrapper supports synchronous or asynchronous updates, implementing the *sequential update* and *truncated update* logic presented in [65], and allows users to specialize measurement and dynamics updates for their platform.

Geometric pose estimation spans absolute and relative pose solvers, ranging from minimal solvers—`p3p` [20], `up2p` for absolute pose [40], and `5pt` [49], `up2pt` [13], `u3pt` [20] for relative pose—to linear solvers—`dlt` [31], `homography` [31, 51], and `up3pt` [13]. `up2p`, and `u3pt` require knowledge of the gravity direction, easily provided by an IMU, and `up2pt`, `up3pt` require knowledge of gravity direction and a planar motion constraint. All plug into a compile-time configurable `lo-ransac` wrapper [15] with optional linear or nonlinear local refinement, and optional final bundle adjustment. Together these can be used to quantify how structural and sensor priors as well as robust estimation, shape energy and latency, providing more building blocks towards visual(-inertial) odometry alongside our perception kernels.

Control. Kernels range from a sparse 4×4 LQR (`fly-lqr`) and it's 10 time-step horizon TinyMPC successor (`fly-tiny-mpc`), to an OSQP-based ADMM MPC [17] (`bee-mpc`), an SE(3) geometric controller [42, 46] `bee-geom`, and a sliding mode adaptive controller `bee-smac` [11, 12]. Benchmarks focus on high-level reference computation only; actuator mapping (e.g., piezoelectric actuator pulses for flapping wing) is left out. TinyMPC and LQR are generic where dynamics and sensor matrices can be easily changed.

Future Extensions. Planned near-term expansions include lightweight factor graph optimization [50], CNN-based monocular depth estimation and object recognition [72], and higher-level navigation kernels [73]. EntoBench will grow iteratively, welcoming community-driven contributions, with versioned releases that will keep the suite reproducible and well documented.

V. WORKLOAD CHARACTERIZATION

Every EntoBench kernel is profiled twice, once with caches disabled and once with caches enabled, on STM32G474 (M4, 128 KB SRAM), STM32U575 (M33, 1 MB SRAM), and STM32H7A3 (M7, 1.4MB SRAM) MCUs. Problem sizes are chosen so that the M4's SRAM is sufficient, making reported results directly comparable across MCUs. Notably for perception tasks, feature detection is performed with 160x160 images and optical flow performed with 80x80 images. The single exception is the SIFT detector, which exceeds the memory of the M4 and M33 and is therefore reported only on the M7. Architectural details for the three boards appear in Table V; full latency, energy, and peak-power numbers with caches on and off are collected in Table IV. Before going

TABLE III
BENCHMARK SUITE STATIC METRICS – FLASH SIZE AND STATIC INSTRUCTION MIX BREAKDOWN

					M4 Instr. Mix				**M33 Instr. Mix**				**M7 Instr. Mix**			
Stage	**Kernel**	**Category**	**Dataset**	**Flash**	F	I	M	B	F	I	M	B	F	I	M	B
P	fastbrief	Feat. Extr.	midd-stereo	5088	113	598	282	200	112	600	269	198	112	618	282	205
P	orb	Feat. Extr.	midd-stereo	11616	671	1267	464	506	677	1236	465	474	463	984	395	312
P	sift	Feat. Extr.	midd-stereo	76268	-	-	-	-	-	-	-	-	2219	4346	2167	1188
P	lkof	Opt. Flow	midd-flow	45844	516	6557	4646	3141	518	6529	4578	3135	518	6529	4578	3135
P	iiof	Opt. Flow	midd-flow	1280	57	175	130	59	57	147	115	57	57	147	115	57
P	bbof	Opt. Flow	midd-flow	2936	296	336	216	98	303	312	234	86	277	288	204	77
S	mahony	Att. Est.	bee-synth	1560	195	112	96	74	196	108	98	74	192	110	96	72
S	madgwick	Att. Est.	bee-synth	1424	168	114	99	62	171	110	101	62	168	114	99	60
S	fourati	Att. Est.	bee-synth	3088	543	138	133	71	588	134	136	70	533	138	132	69
S	fly-ekf (sync)	Kalman Filt.	fly-synth	29072	3650	2632	1381	986	3820	2628	1412	976	3488	2295	1272	772
S	fly-ekf (seq)	Kalman Filt.	fly-synth	29072	3650	2632	1381	986	3820	2628	1412	976	3488	2295	1272	772
S	fly-ekf (trunc)	Kalman Filt.	fly-synth	29072	3650	2632	1381	986	3820	2628	1412	976	3488	2296	1272	772
S	bee-ceekf	Kalman Filt.	bee-hil	38432	4351	3818	2132	1200	4598	3828	2049	1194	4110	3494	2118	986
S	p3p	Abs. Pose	abs-synth	12132	1446	974	387	541	1424	968	392	524	1310	311	230	190
S	up2p	Abs. Pose	up-abs-synth	3720	630	179	147	73	630	180	144	73	629	180	146	73
S	dlt	Abs. Pose	abs-synth	18908	2404	1655	1203	668	2429	1637	1230	659	2393	1644	1199	617
S	absgoldstd	Abs. Pose	abs-synth	31332	4088	2420	1342	1035	4185	2392	1275	1005	3881	1667	1290	779
S	up2pt	Rel. Pose	str-rel-synth	3160	420	217	149	131	418	218	147	130	420	215	148	131
S	up3pt	Rel. Pose	str-rel-synth	8500	559	1166	848	374	562	1164	858	373	569	1164	839	376
S	u3pt	Rel. Pose	upr-rel-synth	9388	1136	673	327	378	1137	667	330	360	866	365	247	195
S	5pt	Rel. Pose	rel-synth	73932	5550	8259	5235	3779	5892	8240	5208	3814	5543	8090	5219	3740
S	8pt	Rel. Pose	rel-synth	30364	3860	2475	1902	1033	4027	2474	1832	1009	3818	2471	1892	981
S	relgoldstd	Rel. Pose	rel-synth	87464	5465	10198	7165	4594	5547	10144	7392	4599	5241	9860	7085	4369
S	homography	Abs./Rel. Pose	homog-synth	4976	824	202	222	117	821	204	224	117	826	180	213	108
S	abs-lo-ransac	Robust Pose	rob-abs-synth	119132	7795	14217	9462	6328	7973	14122	9477	6302	7453	13433	9281	5826
S	rel-lo-ransac	Robust Pose	rob-rel-synth	164660	12597	18919	12682	7902	12997	18849	12808	7874	11937	18281	12812	7533
C	fly-tiny-mpc	Opt. Ctrl.	fly-traj	64676	3390	7683	5583	3590	3550	7683	5769	3586	3396	7819	6024	3565
C	fly-lqr	Opt. Ctrl.	fly-traj	1484	125	175	111	58	112	169	127	60	126	175	110	58
C	bee-mpc	Opt. Ctrl.	bee-synth	80760	2277	9782	6749	4649	2333	9777	6725	4676	2316	9529	6605	4551
C	bee-geom	Geom. Ctrl.	bee-synth	21952	3333	1064	1252	384	3348	1031	1324	373	3119	768	1184	199
C	bee-smac	Adapt. Ctrl.	bee-traj	33840	3423	3268	1385	1496	3476	3233	1400	1463	4005	1561	875	745

TABLE IV
BENCHMARK SUITE DYNAMIC METRICS – LATENCY, ENERGY, AND PEAK POWER WITH AND WITHOUT CACHES

		Latency (μs)						**Energy (μJ)**						**$P_{\max}$ (mW)**					
		M4		**M33**		**M7**		**M4**		**M33**		**M7**		**M4**		**M33**		**M7**	
Stage	**Kernel**	C	NC	C	NC	C	NC	C	NC	C	NC	C	NC	C	NC	C	NC	C	NC
P	fastbrief	26K	26K	24K	46K	11K	34K	3K	3K	702	1K	2K	4K	132	131	35	35	161	128
P	orb	54K	54K	42K	81K	20K	55K	7K	7K	1K	2K	3K	6K	146	142	35	37	174	128
P	sift	-	-	-	-	2M	4M	-	-	-	-	207K	483K	-	-	-	-	216	130
P	lkof	18K	19K	13K	22K	6K	17K	2K	2K	424	699	852	2K	202	200	39	41	200	126
P	iiof	2K	2K	2K	3K	840	2K	199	196	45	73	94	198	108	106	27	30	122	120
P	bbof	468	469	381	590	191	244	52	98	11	9	26	55	115	106	32	38	160	128
S	mahony	2	2	6	9	2	3	0.2	0.2	0.2	0.3	0.2	0.3	106	104	33	37	121	123
S	madgwick	2	2	3	3	0.9	1	0.2	0.2	3	7	0.1	0.2	100	96	30	32	108	118
S	fourati	7	8	6	8	3	4	0.7	0.8	0.1	0.2	0.3	0.4	118	116	31	32	123	122
S	fly-ekf (sync)	40	45	34	39	14	26	5	5	1	1	2	3	136	133	36	36	147	127
S	fly-ekf (seq)	57	67	43	57	18	32	7	8	1	2	2	4	132	128	36	37	149	128
S	fly-ekf (trunc)	43	48	29	46	15	38	5	5	0.9	1	2	4	131	128	35	36	161	125
S	bee-ceekf	4K	4K	4K	7K	2K	4K	529	532	109	200	204	450	139	137	38	36	150	127
S	p3p	129	136	116	198	11	18	14	15	3	6	1	2	120	118	31	33	131	120
S	up2p	12	13	10	12	4	7	1	2	0.2	0.4	0.5	0.7	127	125	34	34	132	126
S	dlt	408	441	301	432	140	301	49	51	9	13	17	34	148	143	37	38	148	127
S	absgoldstd	4K	4K	3K	5K	2K	4K	491	517	93	160	217	405	189	187	41	40	189	153
S	up2pt	14	15	12	15	5	8	1	2	0.3	0.4	0.6	0.9	115	113	30	32	129	122
S	up3pt	25	26	19	28	9	19	3	3	0.5	0.9	1	2	128	124	35	36	152	127
S	u3pt	16	17	14	19	7	11	2	2	0.3	0.5	0.7	1	113	113	31	33	134	124
S	5pt	682	737	605	860	267	498	84	88	18	27	32	56	145	138	42	40	158	126
S	8pt	228	250	173	244	79	157	26	28	5	7	10	18	136	133	37	36	148	128
S	relgoldstd	3K	3K	3K	4K	1K	3K	357	369	72	111	160	294	161	158	44	40	201	145
S	homography	37	40	31	47	14	33	4	5	4	7	2	4	131	130	37	35	166	127
S	abs-lo-ransac	5K	5K	22K	34K	5K	9K	624	630	578	988	550	982	135	132	35	36	154	132
S	rel-lo-ransac	42K	44K	39K	54K	19K	37K	5K	5K	1K	2K	2K	4K	144	140	41	41	189	134
C	fly-tiny-mpc	168	176	151	228	68	181	20	21	5	7	9	21	139	134	39	37	170	127
C	fly-lqr	1	1	1	2	0.5	1	0.2	0.1	0.0	0.1	0.1	0.2	122	120	35	35	165	121
C	bee-mpc	8K	8K	7K	13K	3K	10K	1K	1K	213	399	477	2K	151	147	37	36	183	128
C	bee-geom	73	81	68	83	26	75	8	9	2	2	4	9	120	117	36	35	167	124
C	bee-smac	611	641	619	998	45	87	69	71	17	28	6	10	125	121	38	35	163	122

Notes for TABLE III and TABLE IV: Flash size (B) is reported for the M4 build. For `sift`, flash corresponds to the M7 build. There are very minor differences in Flash size, if any, between the same program compiled for a different Cortex- M processor. Instruction mix reported is number of instructions for **F**loat, **I**nteger, **M**emory, **B**ranch. Dynamic metrics reported use same datasets listed with static metrics.

through each category of kernels, we describe some notable high level trends.

Memory Placement. The use of caching drastically improves the performance for the M7 and generally improves the performance of the M33. For the M7 this is due to memory placement, where the vendor linker script places stack in AXI-SRAM for the M7. Caches can also drastically increase the peak power, especially on the M7 with the largest difference of 86 mW encountered during SIFT, signaling a drastic energy vs peak power tradeoff. The M4 sees slight increases in peak power when using its small, loosely-coupled cache added by the manufacturer that sits between FLASH and the processor. Optimizing memory placement of both instructions and data is considered out of scope for this work due to the fact that different microcontrollers models can have drastically different memory layouts.

Importance of Process Node. Modern silicon processes make the M33, a more recent Cortex-M offering, the most energy efficient MCU in our tests. However, few M33 products are offered in wafer-level chip-scale packages in a footprint suitable for insect-scale robots. These processors are an in between of the M4 and M7, providing a very similar pipeline structure as the M4 but with I- and D- cache support as part of the ARMv8-M Mainline architecture. These cores should become the new workhorse for general insect-scale deployments, with M7 reserved for robots that can afford larger payloads and power sources.

Feature Detection. `fastbrief` and `orb` are integer only except for gaussian blurring in both, and rotated feature and descriptor computation in `orb`. There efficiency is carried over even into Cortex-M as they are primarily a masked pixel-wise threshold test. `sift` is very expensive, especially memory-wise, building four Difference-of-Gaussians (DoG) and using 128-byte descriptors, and barely fits the M7 even with incremental pyramid and DoG building as well as re-computing blurred images to save space.

Optical Flow. Lucas Kanade (`lkof`) is the most computationally expensive due to pyramid creation and computation of spatial and temporal gradients. All optical flow kernels scale as the patch size centered on the tracked feature increases. Block matching (`bbof`) represents the other end of the spectrum, where matching using sum-of-absolute differences is especially efficient in the M7's superscalar pipeline.

Attitude Estimation. Attitude filters, Mahony, Madgwick, and Fourati, all require less than 2k cycles. They are primarily made up of vector operations, and the choice of which one should come down to the complexity of parameter tuning for the specific platform. Mahony and Madgwick in this chart are for IMU architectures, where no magnetometor is present, whereas Fourati may only be used with a MARG architecture, where it is present. Upgrading to a MARG architecture only results in a slight increase in latency.

Extended Kalman Filters. The four-state `fly-ekf` and ten-state `bee-ceekf` are dominated by matrix and vector operations. Constant Jacobians make the RoboFly variant markedly faster. Despite using sparse dynamics and sensor measurement matrices, it is impossible to realize their potential computation benefits in a generic EKF framework. Eigen's sparse matrix class resulted in slower computation due to control flow and dynamic memory allocation overhead.

TABLE V
CONSIDERED CORTEX-M ARCHITECTURES FOR SEC. V

MCU	Key Features
Cortex-M4	3-stage pipeline (ARMv7E-M), up to $\sim$200 MHz, optional SP FPU, widely available even in ultra-compact packaging (e.g., WLCSP).
Cortex-M33	3-stage pipeline (ARMv8-M), up to $\sim$200 MHz, optional SP FPU, optional coprocessor interface, less commonly available in ultra-compact packaging (e.g., WLCSP).
Cortex-M7	6-stage superscalar pipeline with branch prediction (ARMv7E-M), up to $\sim$600 MHz, optional SP or DP FPU, optional I/D caches, optional tightly coupled memory (TCM), widely available even in ultra-compact packaging, typically larger than M4/M33

Pose Estimation. Motion- and/or gravity- aware solvers (`up2p`, `up2pt`, `up3pt`, and `up3pt`) are orders of magnitude cheaper than linear baselines such as `dlt` or the `8-pt` algorithm, which pay the full cost of an SVD. LO-RANSAC workloads couple `p3p` and `5-pt` solvers with nonlinear local optimization and polishing using a 25% inlier ratio, exposing how iteration count amplifies per solver cost. We expand on this in Case Study #4.

Control. The sparsity of the LQR implementation can't be taken advantage of, due to the same issues mentioned with our EKFs. TinyMPC involves dense and iterative matrix-vector products at start-up, which could be moved completely offline. We note that this start-up computation can exceed available stack space on the M4 if the horizon length is too long, motivating compile-time generation. OSQP MPC is the only control kernel with iterative optimization, which can be seen by the instruction breakdowns.

VI. Case Studies

This section demonstrates how EntoBench enables quantitative evaluation of representative kernels across perception, state estimation, and control—the core stages of the insect-scale robotics pipelines. Each case study explores how algorithm, scalar type, dataset and/or architecture interact to shape latency, energy, and memory demands under real-world constraints.

We highlight how EntoBench enables consistent evaluation across diverse tasks and platforms, with task-specific correctness metrics incorporated where applicable. The following subsections describe the setup, research questions, and key findings for each domain.

A. Case Study #1: High-Resolution Exteroception Under Tight Energy Budgets

Motivation. Insect-scale robots face a pronounced exteroception gap: the onboard sensors they can power and process deliver far less environmental richness than typical robotics autonomy stacks demand. Even when microcameras like the

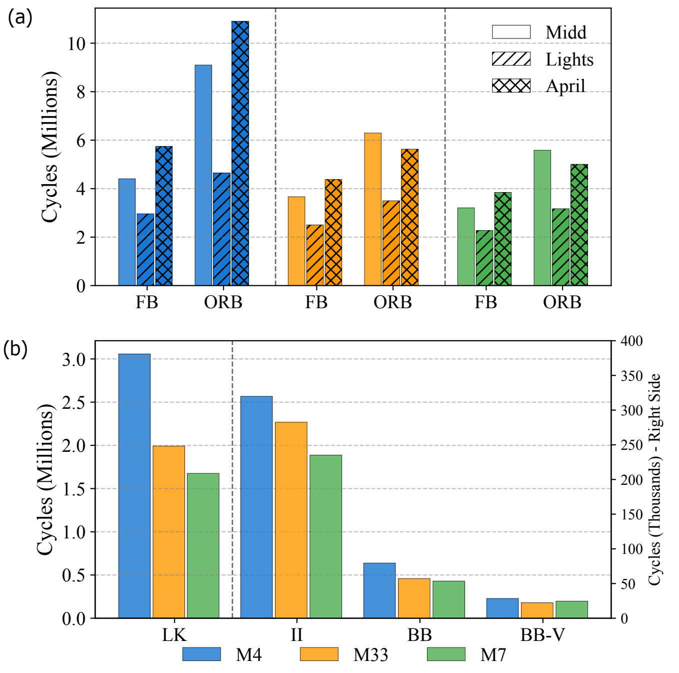


Fig. 3. Cycle Counts for Feature Detection and Optical Flow – (a) Computational complexity for feature detection algorithms across three datasets and (b) comparison of four optical flow kernels of varying complexity.

NanEyeC are integrated, designers often mitigate computational cost through aggressive trade-offs, processing only fixed-size regions [70], using extremely low resolution sensors [23], downsampling in software [70], or sparsifying their pixel stream by, for example, using reduced exposure to isolate bright features [51]. These simplifications are critical to meet energy and real-time constraints, but their cost in perception quality is not well understood.

Meanwhile, advances in wafer-level imaging and microcontroller architecture design (e.g., ARM Helium SIMD extensions in Cortex-M55/M85, RISC-V Scalable Vector) promise to widen the feasible design space for perception-rich autonomy. This case study quantifies how algorithmic complexity and input data characteristics jointly shape energy and latency footprints, setting a baseline for evaluating future SIMD-enabled microcontrollers.

Methodology. We evaluate two perception tasks, feature detection (`fastbrief`, `orb`) and optical flow, `lkof`, `iiof`, `bbof`, across two realistic datasets collected with an insect-scale appropriate camera, the NanEyeC: (1) a highly textured surface, and (2) a sparse, LED-illuminated dataset mimicking constrained lighting conditions from [51]. We measure latency, cycles, and energy consumption on Cortex-M4, M33, and M7.

Results. Figure 3 clearly shows that input data characteristics and algorithm choice strongly affect energy and latency. `orb` feature detection is 1.5–2.5x more costly in cycles and energy than `fastbrief` across all MCUs and datasets. All algorithms run faster and use less energy on the sparse lights dataset compared to the more textured Middlebury and April datasets.

TABLE VI
ENERGY AND PEAK POWER FOR PERCEPTION KERNELS ON CORTEX-M4, M33, AND M7

		Energy			P_{max}		
Kernel	Data	M4	M33	M7	M4	M33	M7
fastbrief	midd	3167	702	1532	132	35	161
	lights	2086	471	1031	127	33	163
	april	4193	853	1873	133	35	173
orb	midd	6574	1218	2564	146	35	174
	lights	3208	679	1422	132	33	169
	april	7950	1110	2358	147	35	174
lkof	midd	2296	424	852	202	39	200
bbof	midd	52	11	26	115	32	160
bbof-vec	midd	19	4	11	128	38	170
iiof	midd	199	45	94	108	27	122

Energy reported in microjoules (μJ) and peak power in milliwatts (mW). Results shown for representative datasets: *Midd*, *Lights*, and *April*. All measurements taken with cache enabled.

Parameter tuning could be very effective for both feature detectors if input data characteristics are known ahead of time for specific applications.

For optical flow, `lkof` (LK) is an order of magnitude more demanding than block-based `bbof` (BB) or `iiof` (II) methods. Vectorized block-based flow `bbof-v` (BB-V) further reduces cycles by almost 4x compared to standard BB, showing the dominance of the sum of absolute differences (SAD) computation that is easily vectorizable with 32-bit, 4-lane `USADA8` instructions.

It is important to note that block-based matching algorithms utilizing SAD computations are particularly well suited for the limited DSP vector instruction set provided by Cortex-M devices (M4, M7, M33). The 4x improvement in `BB-V` demonstrates how algorithms that align well with available SIMD instructions can achieve significant performance gains, while more complex feature detection algorithms like `fastbrief` face greater vectorization challenges due to their irregular memory access patterns and conditional operations that cannot be easily implemented without significant overhead.

Actionable Insights. Using EntoBench, we highlight the data dependence perception algorithms have in terms of energy and latency while also showing the promise of vectorization. System designers must benchmark with real input data while tuning algorithm parameters for maximal reliability and lowest energy cost. Vectorization is promising, but is not straightforward using the very limited 32-bit SIMD support available on Cortex-M4, M33, and M7. New vector instruction sets for microcontrollers may be critical in future designs.

B. Case Study #2: The Precision-Energy Frontier in High-Rate Proprioceptive Estimation

Motivation. Case Study #1 highlighted *exteroception* challenges, emphasizing how sensor input quality and algorithmic choices jointly influence computational feasibility. However, insect-scale robots must also rapidly integrate high-rate inertial data (*proprioception*) hundreds to thousands of times per

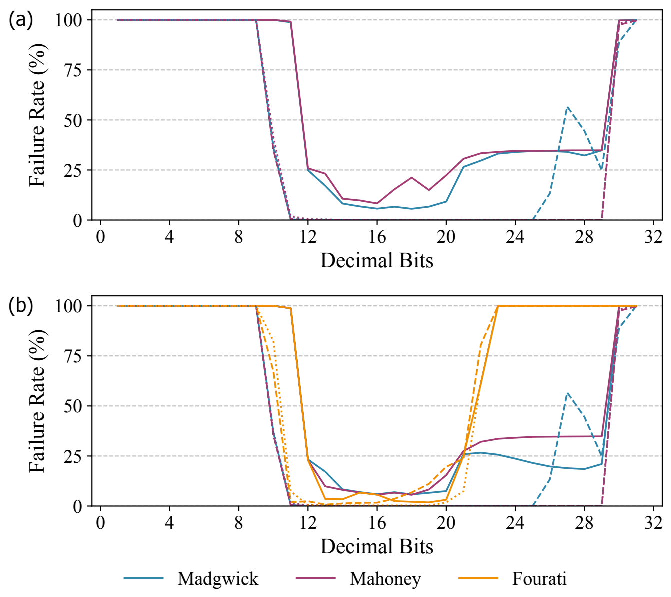


Fig. 4. Attitude Estimation Fixed Point Failure Rate Analysis – Solid, dashed, and dotted line represent failure rates on a simulated IMU dataset from a RoboBee motion capture, a straight line trajectory of a water strider, and an active steering trajectory from a water strider. (a) Portrays the failure rate of fixed point arithmetic for the Mahony and Madgwick IMU based attitude estimators. (b) Portrays the failure rate for Mahony, Madgwick, and Fourati MARG based attitude estimators.

TABLE VII
LATENCY, ENERGY, AND PEAK POWER FOR ATTITUDE FILTERS ON CORTEX-M0+, M4, AND M33

		Latency			Energy			P_{max}		
Filter	Format	M0+	M4	M33	M0+	M4	M33	M0+	M4	M33
mahony (I)	f32	369	2	2	5K	205	68	15	106	34
	q7.24	218	8	6	3K	832	198	15	120	33
madgwick (I)	f32	430	2	2	6K	193	60	15	100	30
	q7.24	399	15	12	6K	1588	396	15	117	33
mahony (M)	f32	642	4	4	9K	338	132	15	105	33
	q7.24	3.0K	11	8	43K	1213	280	15	120	35
madgwick (M)	f32	832	4	4	11K	332	136	15	104	34
	q7.24	492	18	15	7K	1936	510	15	119	34
fourati (M)	f32	1.39K	7	6	19K	679	180	15	118	30
	q7.24	701	20	17	10K	2159	578	15	134	34

Latency reported in microseconds (μs), energy in nanojoules (nJ), and peak power in milliwatts (mW). Filters evaluated in both floating point (f32) and fixed point (q7.24) formats. (I) denotes inertial-only filters, (M) denotes magnetometer-inclusive filters.

second. Attitude estimation algorithms are lightweight enough to run on minimal MCUs (e.g., Cortex-M0+) without floating point hardware, reducing power, PCB complexity, and cost, making them an attractive option during the MCU selection phase. Yet without hardware FPUs, designers must either pay the price of software emulated floating point (`float`, `double`) or resort to fixed-point arithmetic, introducing complexity due to context-dependent numeric precision tuning and increased risk of numeric instability.

Methodology. We use EntoBench to systematically explore the precision-energy tradeoff of three widely used attitude filters, Mahony, Madgwick, and Fourati, on Cortex-M0+, M4, and M33, to show the cost of moving from M0+ to the next tier Cortex-M offerings that include an FPU and benefits of newer process technology. We evaluate on three datasets, the first a synthetically generated dataset using real motion capture data of RoboBee in a hover scenario, the second of the water strider GammaBot [27] performing striding in a straight line, and the last of GammaBot performing a steering maneuver. We evaluate Mahony and Madgwick in two contexts: one when only accelerometer and gyroscope data are present, denoted IMU or I, and one when magnetometer data is also present, denoted MARG or M. We vary fixed-point format, $q_{M.N}$, through the full range of possible values and track the failure rate, counting events due to fixed-point overflow, quaternion norm drift, early exits from near-zero divisors, and large attitude errors exceeding 2.5° where ground truth is available. We select one of the fixed points formats and compare across M0+, M4, and M33.

Results. First, Figure 4 shows the complexity of choosing a proper fixed-point format, even for relatively simple attitude estimators. Dynamic range is needed given that the input sensor data across accelerometer, gyroscope, and magnetometer can vary. The three datasets we test on exhibit different distributions of these sensor values, dependent on the maneuver performed, where gyroscope data can be the hardest to account for as its reports readings in rad/s, an unbounded unit. As for computational performance, Table VI-B shows that despite the lower power draw of M0+ it is limited heavily by its slower clock speed and basic 2-stage pipeline, with energy consumption larger than both M4 and M33 despite its lower power draw. This highlights a key embedded design principle: minimizing energy often requires racing to idle on more capable microcontrollers, rather than relying solely on low-power hardware. The M33's superior energy efficiency stems primarily from its newer process node, as discussed earlier, demonstrating generational improvements beyond architectural enhancements. Last, we note that fixed point is slower on the M4 and M33 due to the need to shift back every multiply.

Actionable Insights. EntoBench highlights significant practical challenges introduced by fixed-point arithmetic, even for lightweight algorithms like attitude estimation. Optimal numeric precision selection strongly depends on platform dynamics, sensor characteristics, maneuver profiles, and update rates, emphasizing the continued importance of hardware FPUs for robustness. Future automated tools for more in depth fixed point sensitivity analysis and analytical methods for converting floating point algorithms to fixed point representations with provable guarantees on accuracy and failure rate could dramatically streamline moving to fixed point representations. Making it easier to utilize the most area, energy and weight efficient microcontrollers available for select tasks. The M33's superior performance reinforces how newer process technology nodes can provide significant energy benefits, making it the optimal choice when packaging constraints permit, even beyond its architectural improvements over older MCU generations.

C. Case Study #3: Is FLOP Counting a Good Model for Latency or Energy Consumption?

Motivation. While Case Study #2 showed that numeric precision governs energy and latency in *proprioceptive* kernels, full sensor fusion is essential for enabling closed-loop control in the lab without relying on external motion capture systems. Recent insect-scale robots increasingly rely on Extended Kalman Filters to fuse inertial and exteroceptive cues, yet many works justify feasibility using only static FLOP counts and datasheet-based power estimates. These estimates implicitly assume idealized computation, often neglecting the effects of memory access, control flow, or algorithmic structure. As robots push toward fully onboard sensing and control, we must re-examine whether FLOP counting meaningfully predicts real performance on microcontrollers.

Methodology. We use EntoBench to benchmark sensor fusion and control algorithms deployed on flapping-wing MAVs: (i) the `fly-ekf` [65], which fuses IMU, optical flow, and time-of-flight data using two asynchronous update strategies, sequential update (S) and truncated update (T); (ii) the `bee-ceekf` [47], which integrates ToF and IMU sensors; (iii) `fly-lqr` [19] for linear quadratic regulation; and (iv) `fly-tiny-mpc`, which employs TinyMPC [48] with a 10-step horizon using the LQR infinite horizon formulation from [19].

Results. Table VIII clearly shows that static FLOP counts alone fail to predict real-world latency, energy, and power for embedded sensor fusion. Practical implementations incur substantial overhead from memory access, conditionals, and matrix sparsity—none of which are captured in FLOP-based estimates. This discrepancy can mislead system designers, especially when provisioning energy or scheduling high-rate estimation loops.

This was also apparent in the `fly-lqr` kernel, where the 4×4 sparse LQR gain cannot be fully taken advantage of to yield the performance estimated in the supplemental material provided. While bespoke hand-tuned implementations can approach FLOP-based estimates by exploiting sparsity and avoiding general-purpose library overhead, this approach becomes impractical as algorithms grow in complexity, limiting the scalability of FLOP-based design methodologies. For example, TinyMPC, which is the logical next step from `fly-lqr` as mentioned in the supplemental material of [19], shows a 17-33x gap between FLOP-estimated and measured energy consumption, demonstrating that more complex optimal control algorithms require significant implementation effort to meet FLOP-based performance predictions.

Actionable Insights. Static FLOP tallies and arithmetic operation counting are safe if careful considerations are made with the implementation on target hardware; real deployment must measure or model memory traffic and control flow. Designers should (i) augment FLOP tallies with a simulated trace to derive an instruction-weighted energy model and (ii)

TABLE VIII
CYCLES AND ENERGY PER UPDATE ON CORTEX-M4, M33, AND M7

		Cycles			Est. Energy			Meas. Energy		
Kernel	FLOPs	M4	M33	M7	M4	M33	M7	M4	M33	M7
fly-ekf (S)	2696	10k	6.4k	5.1k	1.5	0.17	1.1	7	1	2
fly-ekf (T)	1036	7k	4.4k	4.2k	0.6	0.07	0.4	5	0.9	2
bee-ceekf	1063	744k	559k	424k	0.6	0.07	0.4	529	109	204
fly-lqr	30	230	160	136	0.02	0.002	0.01	0.15	0.03	0.07
fly-tiny-mpc	1000	29k	22k	19k	0.6	0.06	0.4	20	4.9	9.3

Estimated energy based on FLOPs (μJ) and nominal per-cycle current consumption from MCU datasheets. Measured energy is per update (μJ). `fly-tiny-mpc` cycle estimate is for a 10-step horizon.

push for MCU-friendly sparse-matrix libraries or compiler passes that exploit fixed sparsity without resorting to hand-tuned code.

D. Case Study #4: Minimal Solver and Robust Estimation Trade-offs for Relative Pose Estimation

Motivation. While high-rate inertial pipelines investigated in Case Studies #2 and #3 deliver the responsiveness required for aggressive control, long-horizon autonomy demands a complementary mechanism for drift correction, typically achieved through geometric visual cues. Relative pose estimation, especially from monocular image pairs, provides this correction. Many minimal solvers exploit structural priors, e.g., known gravity or planar motion, mirroring the design choices of insect-scale robots like GammaBot, which operates on water surfaces with constrained motion and known upright orientation. Upright pose estimation solvers (e.g., `u3pt`, `up2pt`, `up3pt`) exploit the assumption that the camera orientation relative to gravity is known, reducing the degrees of freedom in the pose estimation problem, enabling more efficient computation. `up2pt` and `up3pt` also assume planar motion along the ground plane, similar to a vehicle, or a water strider inspired insect-scale robot such as one seen in [27].

Designers of insect-scale systems often incorporate such priors to reduce computational burden—examples include reduced-state EKFs with constant Jacobians, as seen in RoboFly. These assumptions are baked into estimator and controller design, yet visual pose estimation solvers have not received the same level of principled evaluation. Moreover, real-world use demands robustness to outliers and sensor noise, making it important to benchmark not only standalone solvers, but also their behavior when embedded in robust estimation frameworks. In this case study, we use EntoBench to evaluate both aspects, quantifying how structural priors and robust estimation strategies trade off accuracy, latency, and energy—offering a first-of-its-kind template for system-level visual estimation analysis at the insect scale.

Methodology. Using EntoBench, we evaluate our minimal relative pose solvers and their use in a LO-RANSAC framework. We generate 1000 synthetic problems for all kernels tested, as commonly done in pose estimation literature, allowing for fair comparison across approaches when controlling parameters

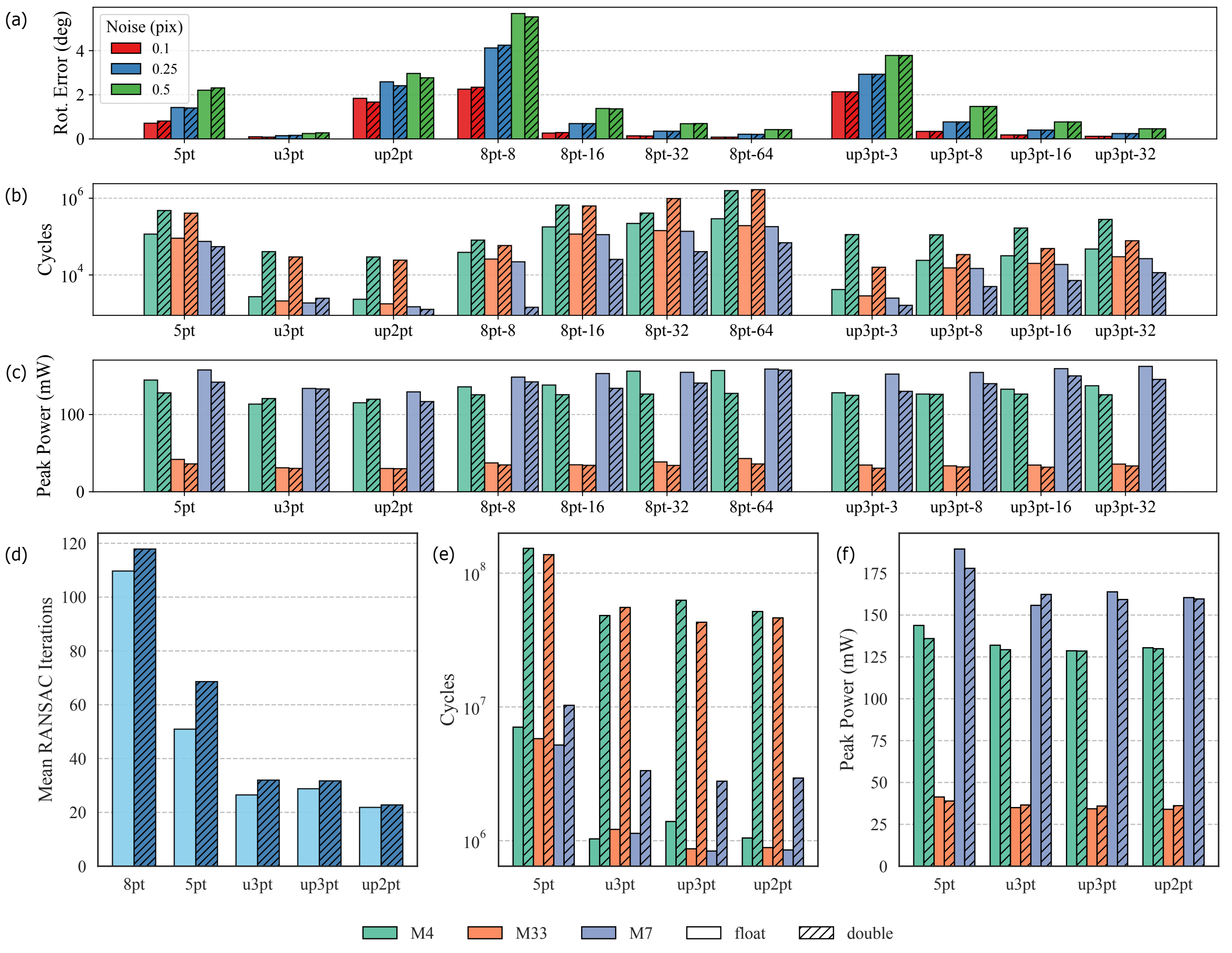


Fig. 5. Combined Relative Pose Estimation and LO-RANSAC Analysis – All plots compare float (solid) and double (diagonal hatched) precision. Plots (a) Rotation error in degrees for minimal and linear relative pose solvers showing accuracy degradation with increasing noise. Linear solvers (e.g., `8pt-N`, representing `8-pt` using N correspondences) see improved robustness to noise with increasing feature correspondences. (b-c) Computational cycle counts and peak power consumption in mW for relative pose solvers at 0.1 pixel noise across different MCU architectures. (d) LO-RANSAC average RANSAC iterations until convergence for different inner loop solvers, demonstrating the importance of minimal solvers. (e-f) Computational cycle counts and peak power consumption for LO-RANSAC using different minimal solvers across different MCU architectures. The 8-pt method is excluded due to its computational overhead making it impractical for insect-scale systems.

such as pixel noise. When testing our robust estimators, the outlier ratio was set to 25% and 0.5 pixels standard deviation noise was used.

Results. As shown in Figure 5, the use of double is not worth the cost in latency on MCUs without a double precision FPU, nor does it provide consistently better accuracy when using well-conditioned data. The `8pt` method is not as accurate unless overdetermined even when using proper normalization. Linear solvers (`8pt`, and `up3pt`) scale linearly in cycles as N increases due to their reliance on SVD for the overdetermined case. Minimal solvers with known motion or sensor priors are reliable and fast, but further testing needs to be done when known gravity direction is more noisy and the planar motion constraint is loosened.

Robust estimation amplifies the gaps between these minimal solvers. The `8-pt` method was not worth it in practice, requiring theoretically many RANSAC samples and failing to find pose estimates with a minimal sample. The `5-pt` method, although only requiring around 2x the number of iterations as the upright solvers, is much slower in practice, due to the strenuous Gröbner basis approach it performs and the fact that it can return up to 10 solutions, which all must be validated each iteration.

Actionable Insights. Leveraging motion or gravity priors in robust pose estimation can lower the gap and brings visual-inertial odometry and SLAM in sight for insect-scale deployment. The benefit of gravity priors alone justify the use of coupled camera-IMU sensor suites despite increased mass, power, and integration costs. We also note that robust pose estimation may not always show up in real-time visual odometry and SLAM, but also in structure from motion, where the robot may intermittently take pictures, detect features, and estimate pose, while slowly optimizing the trajectory over time.

E. Beyond Kernels: Toward Closed-Loop Benchmarks and End-to-End Evaluation

The kernels profiled so far feed a low-level controller that keeps an insect-scale robot on some trajectory. Meeting high update rates is necessary but what ultimately matters when closing the loop is task-level performance such as disturbance rejection, agility, and energy spent per mission. Designers can choose anything from hand-tuned PID to linear MPC, which might all fit on the same MCU yet yield very different trajectories and power draw. Estimator assumptions matter too: the EKFs in Case Study #3 rely on flat ground and hover assumptions, so drift is likely to appear more aggressively in sloped or textured terrains and while performing more agile maneuvers. Kernel-level timing therefore tells only part of the story.

Data and Simulation Gap. Large robots enjoy rich datasets and mature simulators [16, 59, 66] with open-source robot models. Insect-scale robots lack both: data logs are hard to acquire without access to a real robot, and no open simulator has mature and high fidelity models that capture the small-scale physics of these platforms. Without them, designers must guess how kernel trade-offs propagate to mission-level success.

Long-term Roadmap. We plan to extend EntoBench with an open insect-scale simulator that plugs into the current evaluation harness. Controllers will run end-to-end while the framework logs both the compute cost we have presented in this work, as well as task-level metrics (e.g., path error, completion rate). This will answer questions kernel timing alone cannot: How coarse can the dynamics model be before energy savings hurt success? Is co-scheduling enough when all three stages share one processor, or is heterogeneous compute required? Building and validating such a simulation will require close collaboration with insect-scale roboticists, but mapping the path now clarifies why today's kernel results matter and where the systems community can contribute next.

VII. Conclusion

EntoBench delivers the first rigorous workload characterization for insect-scale robots. Its 30-kernel suite mirrors their perception-estimation-control pipeline and pairs it with a low-cost harness for Cortex-M MCUs, providing a systematic and reproducible process for evaluating latency, energy, and peak-power across kernels and MCUs. This process may reveal non-intuitive architecture-algorithm trade-offs and serves as a basis for software optimization and guiding hardware-software co-design.

Four case studies show the range of insights this enables: (1) scene texture, algorithm choice, and available SIMD width jointly bound vision energy; (2) fixed-point arithmetic pays off only when area or integration constraints dominate or if provable guarantees can be made on numerical stability; (3) static FLOP or cycle counts break down when estimating runtime and energy consumption; (4) motion aware minimal solvers make robust visual odometry viable on insect-scale hardware.

EntoBench is open-source, modular and intended to grow. Planned long-term additions include a lightweight insect-scale dynamics simulator, curated insect-scale robot datasets, and microarchitectural simulator support, enabling true closed-loop valuation of autonomy stacks on real hardware. We invite the systems and robotics community to contribute kernels, platforms, and data so that EntoBench can evolve into the standard yardstick for compute autonomy at the insect scale.

Acknowledgments

This work was supported in part by NSF CSSI Award #2311890 and a Cornell College of Engineering SPROUT award.

References

[1] "An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications," in *Int'l Conf. on Robotics and Automation (ICRA)*.

[2] C. A. Aubin, R. H. Heisser, O. Peretz, J. Timko, J. Lo, E. F. Helbling, S. Sobhani, A. D. Gat, and R. F. Shepherd, "Powerful, soft combustion actuators for insect-scale robots," *Science*, vol. 381, Sep 2023.

[3] A. T. Baisch and R. Wood, "Design and fabrication of the harvard ambulatory micro-robot," *Int'l Symp. on Robotics Research (ISRR)*, Aug 2011.

[4] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, Feb 2004.

[5] M. Bakhshalipour and P. B. Gibbons, "Agents of autonomy: A systematic study of robotics on modern hardware," *Measurements and Analysis of Computing Systems (MACS)*, Dec 2023.

[6] M. Bakhshalipour, M. Likhachev, and P. B. Gibbons, "Rtrbench: A benchmark suite for real-time robotics," *Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, May 2022.

[7] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J. sun Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, "Benchmarking tinyml systems: Challenges and direction," *Computing Research Repository (CoRR)*, vol. arXiv:2003.04821, Aug 2020.

[8] S. Bergbreiter and K. S. Pister, "Design of an autonomous jumping microrobot," *Int'l Conf. on Robotics and Automation (ICRA)*, Apr 2007.

[9] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. J. Reddi, "Mavbench: Micro aerial vehicle benchmarking," *Int'l Symp. on Microarchitecture (MICRO)*, Oct 2018.

[10] Y. Chen, E. F. Helbling, N. Gravish, K. Ma, and R. J. Wood, "Hybrid aerial and acquatic locomotion in an at-scale robotic insect," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, Aug 2015.

[11] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Adaptive control for takeoff, hovering, and landing of a robotic fly," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, Nov 2013.

[12] ——, "Adaptive control of a millimeter-scale flapping-wing robot," *Bioinspiration and Biomimetics*, vol. 9, no. 2, p. 025004, May 2014.

[13] S. Choi and J.-H. Kim, "Fast and reliable minimal relative pose estimation under planar motion," *Image and Vision Computing*, vol. 69, pp. 103–112, Jan 2018.

[14] Y. M. Chukewad, J. James, A. Singh, and S. Fuller, "Robofly: An insect-sized robot with simplified fabrication that is capable of flight, ground, and water surface locomotion," *IEEE Transactions on Robotics*, pp. 2025–2040, May 2021.

[15] O. Chum, J. Matas, and J. Kittler, "Locally Optimized RANSAC," in *Pattern Recognition*, G. Goos, J. Hartmanis, J. Van Leeuwen, B. Michaelis, and G. Krell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 2781, pp. 236–243.

[16] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[17] A. De, R. McGill, and R. J. Wood, "An efficient, modular controller for flapping flight composing model-based and model-free components," *Int'l Journal of Robotics Research (IJRR)*, vol. 41, no. 4, Apr 2022.

[18] G. C. H. E. de Croon, J. J. G. Dupeyroux, S. B. Fuller, and J. A. R. Marshall, "Insect-inspired ai for autonomous robots," *Science Robotics*, vol. 7, no. 67, Jun 2022.

[19] D. Dhingra, K. Kaheman, and S. B. Fuller, "Modeling and LQR control of insect sized flapping wing robot," *npj Robotics*, vol. 3, no. 1, p. 6, Mar 2025.

[20] Y. Ding, J. Yang, V. Larsson, C. Olsson, and K. Åström, "Revisiting the P3P Problem," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vancouver, BC, Canada: IEEE, Jun 2023, pp. 4872–4880.

[21] N. Doshi, K. Jayaram, S. Castellanos, S. Kuindersma, and R. Wood, "Effective locomotion at multiple stride frequencies using proprioceptive feedback on a legged microrobot," *Bioinspiration and Biomimetics*, vol. 14, no. 5, p. 056001, Jul 2019.

[22] D. S. Drew, N. O. Lambert, C. B. Schindler, , and K. S. J. Pister, "Toward controlled flight of the ionocraft: A flying microrobot using electrohydrodynamic thrust with onboard sensing and no moving parts," *IEEE Robotics and Automation Letters (RAL)*, vol. 3, no. 4, pp. 2807–2813, Oct 2018.

[23] P.-E. J. Duhamel, C. O. Perez-Arancibia, G. L. Barrows, and R. J. Wood, "Biologically Inspired Optical-Flow Sensing for Altitude Control of Flapping-Wing Microrobots," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 2, pp. 556–568, Apr 2013.

[24] EEMBC, "ULPMark® Benchmark," Embedded Microprocessor Benchmark Consortium (EEMBC), Benchmark Specification, 2019.

[25] S. Fuller, Z. Yu, and Y. P. Talwekar, "A gyroscope-free visual-inertial flight control and wind sensing system for 10-mg robots," *Science Robotics*, vol. 7, no. 72, Nov 2022.

[26] S. B. Fuller, E. F. Helbling, P. Chirarattananon, and R. J. Wood, "Using a mems gyroscope to stabilize the attitude of a fly-sized hovering robot," *Int'l Micro Air Vehicle Conf.*, Aug 2014.

[27] H. Gao, S. Jung, and E. F. Helbling, "High-speed interfacial flight of an insect-scale robot," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2024.

[28] A. Ghosh, "Scaling laws," *Mechanics Over Micro and Nano Scales*, May 2011.

[29] B. Goldberg, N. Doshi, K. Jayaram, and R. J. Wood, "Gait studies for a quadrupedal microrobot reveal contrasting running templates in two frequency regimes," *Bioinspiration and Biomimetics*, vol. 12, no. 4, Jun 2017.

[30] B. Goldberg, R. Zufferey, N. Doshi, E. F. Helbling, G. Whittredge, M. Kovac, and R. Wood, "Power and control autonomy for high-speed locomotion with an insect-scale legged robot," *IEEE Robotics and Automation Letters (RAL)*, vol. 3, no. 2, pp. 987–993, Apr 2018.

[31] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, UK ; New York: Cambridge University Press, 2003.

[32] K. Hayaram, J. Shum, S. Castellanos, E. F. Helbling, and R. Wood, "Scaling down an insect-size microrobot, hamr-vi into hamr-jr," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2020.

[33] E. F. Helbling, S. B. Fuller, and R. J. Wood, "Pitch and yaw control of a robotic insect using an onboard magnetometer," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2014.

[34] ——, "Altitude Estimation and Control of an Insect-Scale Robot with an Onboard Proximity Sensor," in *Robotics Research*, A. Bicchi and W. Burgard, Eds. Cham: Springer International Publishing, 2018, vol. 2, pp. 57–69.

[35] D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and Generic Corner Detection Based on the Accelerated Segment Test," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 6312, pp. 183–196.

[36] N. T. Jafferis, E. F. Helbling, M. Karpelson, and R. Wood, "Untethered flight of an insect-sized flapping-wing microscale aerial vehicle," *Nature*, vol. 570, pp. 491–495, Jun 2019.

[37] J. James, V. Iyer, Y. Chukewad, S. Gollakota, and S. B. Fuller, "Liftoff of a 190 mg laser-powered aerial vehicle: The lightest wireless robot to fly," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2018.

[38] K. Johnson, V. Arroyos, A. Ferran, R. Villanueva, D. Yin, T. Elberier, A. Aliseda, S. Fuller, V. Iyer, and S. Gollakota, "Solar-powered shape-changing origami microfliers," *Science Robotics*, vol. 8, no. 82, Sep 2023.

[39] M. Kovac, M. Fuchs, A. Guignard, J.-C. Zufferey, and D. Floreano, "A miniature 7g jumping robot," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2008.

[40] Z. Kukelova, M. Bujnak, and T. Pajdla, "Closed-Form Solutions to Minimal Absolute Pose Problems with Known Vertical Direction," in *Computer Vision – ACCV 2010*, R. Kimmel, R. Klette, and A. Sugimoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6493, pp. 216–229.

[41] Y. Lai, C. Zang, G. Luo, S. Xu, R. Bo, J. Zhao, Y. Yang, T. Jin, Y. Lan, Y. Wang, L. Wen, W. Pang, and Y. Zhang, "An agile multimodal microrobot with architected passively morphing wheels," *Science Advances*, Dec 2024.

[42] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *IEEE Conf. on Decision and Control (CDC)*. Atlanta, GA: IEEE, Dec 2010, pp. 5420–5425.

[43] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov 2004.

[44] V. Mayoral-Vilches, J. Jabbour, Y.-S. Hsiao, Z. Wan, M. Crespo-Alvarez, M. Stewart, J. M. Reina-Muñoz, P. Nagras, G. Vikhe, M. Bakhshalipour, M. Pinzger, S. Rass, S. Panigrahi, G. Corradi, N. Roy, P. B. Gibbons, S. M. Neuman, B. Plancher, and V. J. Reddi, "Robotperf: An open-source, vendor-agnostic, benchmarking suite for evaluating robotics computing system performance," *Computing Research Repository (CoRR)*, vol. arXiv:2309.09212v2, Jan 2024.

[45] V. Mayoral-Vilches, J. Jabbour, Y.-S. Hsiao, Z. Wan, A. Martínez-Fariña, M. Crespo-Álvarez, M. Stewart, J. M. Reina-Muñoz, P. Nagras, G. Vikhe, M. Bakhshalipour, M. Pinzger, S. Rass, S. Panigrahi, G. Corradi, N. Roy, P. B. Gibbons, S. M. Neuman, B. Plancher, and V. J. Reddi, "RobotPerf: An Open-Source, Vendor-Agnostic, Benchmarking Suite for Evaluating Robotics Computing System Performance," 2023.

[46] R. McGill, N. seung Patrick Hyun, and R. J. Wood, "Modeling and control of flapping-wing micro-aerial vehicles with harmonic sinusoids," *IEEE Robotics and Automation Letters (RAL)*, vol. 7, no. 2, Dec 2021.

[47] A. Naveen, J. Morris, C. Chan, D. Mhrous, E. F. Helbling, N.-S. P. Hyun, G. Hills, and R. J. Wood, "Hardware-in-the-Loop for Characterization of Embedded State Estimation for Flying Microrobots," 2024.

[48] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "Tinympc: Model-predictive control on resource-constrained microcontrollers," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2024.

[49] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, Jun 2004.

[50] E. Olson, "AXLE: Computationally-efficient trajectory smoothing using factor graph chains," in *Int'l Conf. on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, May 2021, pp. 7443–7448.

[51] D. Ozturk, Z. Wang, and E. F. Helbling, "Absolute pose estimation for a mm-scale vision system," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, Oct 2024.

[52] J. Pallister, S. Hollis, and J. Bennett, "BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms," Sep 2013.

[53] D. Patterson, J. Bennett, M. Bennett, H. Chelin, D. Harris, J. Hellar, W. Jones, K. Moron, P. Savini, R. Shepherd, R. Simar, Z. Susskind, and S. Wallentowitz, "Embench IOT 2.0 and DSP 1.0: Modern Embedded Computing Benchmarks," *Computer*, vol. 58, no. 5, pp. 37–47, May 2025.

[54] R. S. Pierre and S. Bergbreiter, "Gait exploration of sub-2 g robots using magnetic actuation," *IEEE Robotics and Automation Letters (RAL)*, vol. 2, no. 1, pp. 34–40, Jan 2017.

[55] H. K. H. Prasad, Y. M. C. Ravi Sankar Vaddi, E. Dedic, I. Novosselov, and S. B. Fuller, "A laser-microfabricated electrohydrodynamic thruster for centimeter-scale aerial robots," *PLOS ONE*, vol. 15, p. e0231362, Apr 2020.

[56] Z. Ren, S. Kim, X. Ji, W. Zhu, F. Niroui, J. Kong, and Y. Chen, "A high-lift micro-aerial-robot powered by low-voltage and long-endurance dielectric elastomer actuators," *Advanced Materials*, vol. 34, p. 2106757, Nov 2021.

[57] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 3951, pp. 430–443.
[58] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Int'l Conf. on Computer Vision (ICCV)*. Barcelona, Spain: IEEE, Nov 2011, pp. 2564–2571.
[59] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*, M. Hutter and R. Siegwart, Eds. Cham: Springer International Publishing, 2018, vol. 5, pp. 621–635.
[60] B. H. Shin, K.-M. Lee, and S.-Y. Lee, "A miniaturized tadpole robot using an electromagnetic oscillatory actuator," *Journal of Bionic Engineering*, Mar 2015.
[61] S. Singh, Z. Temel, and R. S. Pierre, "Multi-modal jumping and crawling in an autonomous springtail-inspired microrobot," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2024.
[62] Y. S. Song and M. Sitti, "Stride: A highly maneuverable and non-tethered water strider robot," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2007.
[63] M. V. Srinivasan, "An image-interpolation technique for the computation of optic flow and egomotion," *Biological Cybernetics*, vol. 71, no. 5, pp. 401–415, Sep 1994.
[64] S. H. Suhr, Y. S. Song, S. J. Lee, and M. Sitti, "Biologically inspired miniature water strider robot," *Robotics: Science and Systems (RSS)*, 2005.
[65] Y. P. Talwekar, A. Adie, V. Iyer, and S. B. Fuller, "Towards sensor autonomy in sub-gram flying insect robots: A lightweight and power efficient avionics system," *Int'l Conf. on Robotics and Automation (ICRA)*, May 2022.
[66] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Int'l Conference on Intelligent Robots and Systems (IROS)*. Vilamoura-Algarve, Portugal: IEEE, Oct 2012, pp. 5026–5033.
[67] C. K. Trygstad, E. K. Blankenship, and N. O. Perez-Arancibia, "A new 10-mg sma-based fast bimorph actuator for microrobotics," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, Oct 2024.
[68] R. J. Wood, "The first takeoff of a biologically inspired at-scale robotics insect," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 341–347, Apr 2008.
[69] X. Yang, L. Chang, and N. O. Pérez-Arancibia, "An 88-milligram insect-scale autonomous crawling robot driven by a catalytic artificial muscle," *Science Robotics*, vol. 5, no. 45, p. eaba0015, Aug 2020.
[70] Z. Yu, J. Tran, C. Li, A. Weber, Y. P. Talwekar, and S. Fuller, "Tinysense: A lighter weight and more power-efficient avionics system for flying insect-scale robots," *Computing Research Repository (CoRR)*, vol. arXiv:2501.03416, Jan 2025.
[71] X. Zhang, M. Lok, T. Tong, S. K. Lee, B. Reagen, S. Chaput, P.-E. J. Duhamel, R. J. Wood, D. Brooks, and G.-Y. Wei, "A fully integrated battery-powered system-on-chip in 40-nm cmos for closed-loop control of insect-scale pico-aerial vehicle," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 52, no. 9, pp. 2374–2387, Sep 2017.
[72] H. Zheng, S. Rajadnya, and A. Zakhor, "Monocular depth estimation for drone obstacle avoidance in indoor environments," *Int'l Conf. on Intelligent Robots and Systems (IROS)*, Oct 2024.
[73] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, and F. Gao, "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm5954, May 2022.

Appendix

A. Abstract

EntoBench reproduces the paper's latency, peak power, and energy measurements on STM32 Nucleo boards. This appendix gives hardware/software requirements, Ubuntu 24.04 setup (toolchain, Python environment, GUI tools), and a step-by-step workflow (build/flash, acquire logic/current traces, run analysis scripts) to regenerate the Workload Characterization results.

B. Artifact check-list (meta-information)

- **Algorithm: Insect-Scale Robotics Kernels**
- **Program: C/C++, bare-metal kernels, benchmarking harness, energy measurement scripts**
- **Compilation: cross-compilation (`arm-none-eabi`), Native (x86, ARM64)**
- **Binary: ARM Cortex-M ELF Firmware**
- **Data set: robot data, simulated data**
- **Run-time environment: Bare-metal ARM Cortex-M Microcontroller**
- **Hardware: ARM Cortex-M, STM32-Nucleo**
- **Execution: Manual + scripted; `make` targets; GUI for measurements**
- **Metrics: Latency, Cycle Counts, Peak Power, Energy**
- **Output: Measurement Logs, Terminal Output**
- **Experiments: kernel benchmarks, workload characterization**
- **How much disk space required (approximately)?: 10 GB**
- **How much time is needed to prepare workflow (approximately)?: 0.5-1 hr**
- **How much time is needed to complete experiments (approximately)?: 0.5-1 hr for 5 experiments**
- **Publicly available?: Yes**
- **Workflow automation framework used?: None (shell + make)**
- **Archived (provide DOI)?: https://doi.org/10.5281/zenodo.16931570**

C. Description

EntoBench is organized as a modular benchmarking framework for insect-scale robotics workloads. Kernels are compiled into standalone bare-metal benchmarks and executed on STM32 Nucleo boards. Measurement scripts and harnesses automate flashing, cycle counting, and collection of current traces.

The experiment workflow is designed to reproduce the results in the Workload Characterization section of the paper. In practice, this means (1) building and flashing benchmark kernels to supported microcontrollers, (2) collecting logic analyzer and current measurements with GUI tools, and (3) running Python analysis scripts to derive latency, peak power, and energy.

1) How to access: Users may access the artifact using the following DOI: https://doi.org/10.5281/zenodo.16931570. They may also access the code at https://github.com/cornell-brg/ento-bench.

2) Hardware dependencies: To use EntoBench users ideally have access to the following hardware:

- STLINK-V3PWR for Current Measurements, Flashing, and Semihosting
- Saleae Logic Pro Analyzer
- One or more of the following STMicroelectronics STM32 Nucleo Development Boards: NUCLEO-STM32G474RE, NUCLEO-STM32U575ZIQ, NUCLEO-STM32H7A3ZIQ

It should be possible to follow this processing using an X-NUCLEO-LPM01A current probe, commonly seen in TinyML performance benchmarking papers, but we do not provide official support and instructions for this setup. Another programmer may be used, such as an STLINK-V3SET, ST-LINK/V2, etc., such that the STLINK-V3PWR is used solely for current measurements. However, this artifact appendix assumes the user has the hardware listed above.

3) Software dependencies: To use EntoBench, users must manually download two user applications, `STM32CubeMonPwr` and `Saleae Logic 2`, in addition the cross-compilation toolchain used (`GNU ARM Embedded Toolchain`), several software packages that may be installed via the command line for build support, and setting up a Python environment that can run our energy analysis scripts.

D. Installation

Installation of the required software for Ubuntu 24.04 is outlined below. Installation on MacOS should be similar, using `brew` instead of `apt`, and installing the MacOS `STM32CubeMonPwr` application instead of the Linux one. The following instructions were performed on a newly installed Ubuntu 24.04 system.

a) Packages installed via the Command Line.: We provide automation scripts in the `scripts/` directory to simplify installation. To install the needed system packages, run:

```
% cd scripts/install
% ./00-install-system-packages.sh
```

This script installs all necessary build prerequisites, including `build-essential`, `cmake`, `pkg-config`, `autotools`, `libjim-dev`, `jimsh`, and `openjdk-17-jre`. Java version 17 will be set as the system default if multiple versions are installed.

b) Embedded Toolchain Installation.: The ARM GNU Embedded Toolchain (v14.3) is installed automatically by:

```
% ./01-install-arm-toolchain.sh
```

This downloads the official multilib binaries into `~/.toolchains/arm-none-eabi-14.3`, updates `PATH` in `~/.bashrc`, and verifies the installation. After running, open a new shell or `source .bashrc` so the toolchain is available.

c) Python Environment.: A dedicated virtual environment at `~/.venvs/entobench-ae` is created by:

```
% ./02-setup-python-venv.sh
```

This installs common scientific packages (NumPy, Pandas, SciPy, Matplotlib). To use it later:

```
% source ~/.venvs/entobench-ae/bin/activate
```

d) Current Measurement and Logic Analyzer Support.: After completing the above, install support for the STLINK-V3PWR by running:

```
% ./03-setup-udev-stlinkv3pwr.sh
```

This adds a permissive `udev` rule and reloads rules so the device is accessible without `sudo`. Unplug and replug the STLINK-V3PWR once.

Two GUI applications must still be installed manually:

- **STM32CubeMonPwr.** Download from https://www.st.com/en/development-tools/stm32cubemonpwr.html. We used v1.2.1 (v1.1.1 also supported). Extract to `~/workspace/external`.
- **Saleae Logic 2.** Download the AppImage from https://support.saleae.com/logic-software/sw-installation and place it under `~/external/logic2/`.

After placing both applications, register convenient aliases:

```
% ./04-register-cmp-logic2.sh
% source ~/.bashrc
```

This sets up `cube-monitor-pwr` and `logic2`.

e) Verification.: Finally, verify the installation with:

```
% ./05-verify-setup.sh
```

This script checks that the toolchain, Python environment, Java runtime, and STLINK-V3PWR device permissions are all set up correctly.

E. Experiment workflow

As stated earlier, our experiment workflow requires manual interaction when collecting latency, peak power, and energy measurements. Cycle counts may always be gathered automatically using our build commands. We now provide a step-by-step example using an example kernel, that is not part of the benchmark suite that can be found in `benchmark/example`. The example kernel is `vector-vector add`. The same process can be replicated for all benchmarks shown in the Workload Characterization section.

1) **Prepare the experiment environment.**
   a) From the project root, set up the experiment and build folders:

   ```
   % ./scripts/flow/00-setup-experiment-env
   ```

   b) This creates measurement directories separated by MCU and pipeline stage, as well as build folders (`build-g474`, `build-u575`, `build-h7a3`).
   c) A list of benchmark target commands can be found in `docs/workload-char-targets.txt`.
2) **Launch required applications.**
   a) From the project root, you can conveniently launch the applications needed for full measurement using the following command.

   ```
   % scripts/flow/01-launch-apps
   ```

   b) Start `STM32CubeMonPwr` (CMP). **NOTE: CMP can error sometimes and which usually requires device reconnection and potentially closing and restarting the app.**
   c) Start `Logic 2`.
   d) Open a new terminal window for build commands.
   e) Configuration guides are provided in `docs`: `ae-gui-setup.md` and `ae-hardware-setup-guide.md`
3) **Run the example benchmark with current and logic analyzer acquisition.**
   a) In `Logic 2`, press *Play*.
   b) In CMP, press *Start Acquisition*.
   c) Navigate to the build folder corresponding to the connected MCU (e.g., G474) from the project root:

   ```
   % cd build/build-g474
   % make stm32-flash-bench-example-
       semihosted
   ```

   d) OpenOCD will flash the program and the Harness will print output in the terminal.
4) **Save measurement data.**
   a) In Logic 2:
      - Save a `.sal` file into `experiments/ae/m4/example/example-cache`. The folder has already been created for you.
      - Export raw data to the same directory.
   b) In CMP:
      - Press *Save Graph* and store the output in the same directory. **NOTE: The *Save Graph* button can be hard to click correctly. Try clicking in the bottom right corner.**
   c) File names are arbitrary; analysis scripts will rename them automatically.
5) **Analyze the experiment.**
   a) Activate the Python environment:

   ```
   % source ~/.venvs/entobench-ae/bin/
       activate
   ```

   b) Run the analysis script from the project root:

   ```
   % ./scripts/flow/02-analyze-energy.sh \
      -d experiments/ae/m4/example/benchmark
          -example-cache
   ```

   c) Outputs include cycle counts, average energy, peak power, and latency.

F. Evaluation and Expected Results

Compare the results in the terminal from running the example workflow above with results shown in the document found in `docs/expected-results.md`.

G. Experiment customization

To change experiment orchestration settings, especially available caches on or off, see the `docs/experiment-customization-example.md`.