RESEARCH STATEMENT

John-Paul Ore

Systems interacting with the physical world (e.g., robots, cyber-physical systems, embedded systems) depend on the correct interplay of code and the physical environment. However, our methods for analyzing these systems often intentionally decouple code from the physical world to make the analysis simpler. The broad aim of my research is to *improve the reliability of systems we build by reuniting code analysis with the physical environments in which code executes.*

My technical approach involves decomposing the logic of program analysis into atomic bits, adding insights from robotics, and using the combination to reveal meaningful implications. The program analysis includes facts about a program's execution, i.e., *for this code block to execute, what must be true?* The insights from robotics include semantics from variable names like `dist_obj` $\rightsquigarrow$ *distance in meters measured by laser scanner.* Together, robot insights and program facts enable new ways to think about questions of specific system behavior within contexts, such as: *"What is the behavior of a robot with this particular sensor and state estimator in the code:* `if(abs(dist_obj - current_position) ≤ threshold) {halt();}`?" The code behavior alone is perfectly clear, but the actual system behavior depends on sensor noise impacting `dist_obj`, state estimation uncertainty impacting `current_position`, and the actuator commanded to `halt`. Armed with this new information, we might conclude, *"the robot is less likely to halt if an obstacle absorbs* 850 *nm light."* At a higher level, I seek to expand program analysis to reason about *desirable properties* of robotic systems as they interact with the physical world: spatial reasoning, forces and energy, sensing and actuation, and time.

My contributions include new software engineering methods that enable dimensional analysis *without developer annotations* for the Robot Operating System [C-6], new open-source tools that implement these methods (Phriky [C-7], Phys [C-10]), open datasets of *dimensional inconsistencies* found in the wild [C-10], and novel techniques in aerial field robotics for environmental monitoring [C-1, C-5, J-1, J-4]. I see field robotics and software engineering (SE) as complementary disciplines with an increasingly fruitful intersection. Field robotics exposes the shortcomings of our ability to reason about whole systems with different levels of abstraction within realistic environments, and SE increasingly enables new capabilities to develop more dependable safety-critical systems. I have a graduate advisor from each of these two domains, to forge new insights from the amalgamation of both. I have published at top venues in both domains (SE: *FSE*, *ASE*, *ISSTA*; Robotics: *JFR*, *ICRA*, *IROS*, *FSR*). I am convinced that the intersection of program reasoning and robotics is full of deep problems and big solutions.

Below I briefly summarize my research in SE and robotics, then discuss directions for future work.

Program Analysis for Robot Software

Robot software is exposed to special hazards, including *dimensional inconsistencies*, i.e., accidentally adding two `doubles`, a valid operation, that represent incompatible quantities, velocity and distance. This hazard has become more pronounced with the surging popularity of the Robot Operating System (ROS), especially in the misuse of the ROS-API [C-8]. Incorporating dimensions and physical units in programming languages has been studied by myriad efforts for decades, and theoretically solved with developer type annotations. However, making these annotations incur time and toolchain penalties and are frequently incorrect [C-9], so ROS developers 'vote with their keyboards' and adopt type annotations in only 1.4% systems we examined [C-6]. These inconsistencies hinder code reuse, reduce code quality, and expose developers to software defects within both their code and when sharing code.

My research provides the benefits of type checking *without* the burden of type annotations by employing *Abstract Type Inference* (ATI). This idea enriches program analysis making *implicit* meaning in the program *explicit*. For example, a variable `vel` is of datatype `float` that is implicitly an angular velocity (s^{-1}). Once explicit, this physical unit type information can be used to detect type inconsistencies, an infrequent but potentially impactful software defect [C-8].

My research contributions in this area include: 1) a human study assessing the burden of making type annotations, showing that developers are only 51% accurate and that making a single correct annotation takes more than 2 minutes [C-9] (IRB# 20170817412EX); 2) a method of ATI for ROS systems that is lightweight, intraprocedural, and semi-flow sensitive [C-6]; 3) an implementation of this method in an open-sourced tool, Phriky[1] [C-7] that received the 'Best Tool Demonstration Award' at ISSTA'17; 4) an evaluation of 5.9M lines of ROS code, finding 6% of $3,484$ repositories contain dimensional inconsistencies [C-8]; and 5) a novel method of performing ATI with uncertain evidence in variable names using probabilistic graphical models (belief networks) in collaboration with SE researchers from Purdue [C-10], and an implementation of this approach in the open tool Phys that significantly increases the power of detecting inconsistencies.

[1] Phrikê ("freaky") is the Greek spirit of horror ($\varphi\varrho\iota\kappa\eta$).

My tools for ATI target ROS `C++` for impact. However, my work on the *type annotation burden* [C-9] and belief networks [C-10] applies generally to untyped languages (like `Javascript`), where burgeoning interest in activating the power of type checking within untyped contexts has resulted in new industrial tools (Microsoft →`TypeScript`, Facebook →`Flow`) that can directly benefit from my research.

Phriky contributed to a successful NSF small-grant application I helped write (NSF-CCF #1718040, $484,694), giving me familiarly with the process and funding model. Helping write a successful NSF grant showed me the value of: 1) prior published work; 2) working prototypes; and, 3) the importance of expressing an alluring vision of the impact of the work.

Next steps in this area include: assessing the impact of these tools on developers; creating and assessing a physical units type assistant; building a dataset of *reproducible* physical unit inconsistencies on robots in simulation; a large-scale application and comparison of robot software static analyzers; an extension to include units-of-measure checking (*km* versus *cm*); and, extending the belief networks to include untapped sources of uncertain evidence, such as code comments, predictions from deep networks, and predictions from clustering variables by semantics.

Field Robotics and Aerial Water Sampling

My experience in field robotics upholds the adage: *"Robotics is the science of cables and connectors."* My experiences in field robotics have convinced me of the profound complexity and grand challenge of unstructured environments. Field robotics provides SE with both an instructive proving ground and also an expanding horizon of challenges.

My contributions in field robotics focus on Earth's most precious natural resource: fresh surface water. Autonomously collecting water samples from the air with a Unmanned Aerial Vehicle (UAV) is a challenging and unforgiving problem domain, especially when the robot is not waterproof. The technical challenges include forming an accurate altitude estimation near water while filtering noisy sensor readings caused by the dangling water collection tube [C-1, J-1], delivering a cable-attached sensor payload to a precise depth and holding it there until the sensor values settle ($\approx 3s$) [C-5], modeling the non-linear dynamics of the dangling tube when combined with wind gusts and GPS drift in simulation to establish implementation guidelines for sampling rate and sensor delay [J-4], and remaining within the payload constraints of a micro-UAV with enough payload left over for the water sample. Aerial field robotics also requires operating in compliance with federal and institutional requirements. Overall, the implementation challenges span: the hardware-software interface, embedded system design and programming, wireless communication, robotic middleware, higher-level behavior, and all the interfaces between these elements. To help mitigate the complexity in these many layers and to guard the system during field experiments, I implemented safety monitors using state machines (DFAs) that detect unsafe system behavior that automatically triggers recovery strategies. I also used my software analysis tools to avoid dimensional inconsistencies.

My work pioneered the use of aerial robots for water sampling in collaboration with hydrologists and conservation biologists from UC Berkeley [C-1, J-1], validating techniques now used by eDNA researchers in Japan, Arctic researchers in Sweden, hydrologists in California, and mining engineers in Australia. These efforts are inherently interdisciplinary, and I have benefited from collaborating with talented mechanical and electrical engineers. My colleagues and I received a US patent [P-1]. The working prototype of our aerial water sampler helped us win funding (#2013-67021-20947 USDA-NIFI-NRI, $956,210).

I foresee the focus of my next work in this domain to shift from pure field robotics to collaborative endeavors simultaneously serving both field robotics and new SE threads. **Next steps** in this area include: sensing and characterizing *environmental invariants* in the field to guide simulation; developing structured representations of environmental assumptions at runtime as new inputs to program analysis; field test ordering with expanded notions of test coverage of sensors, actuators, and software; and deeper use of simulation at runtime updated in the current operating environment to predict mission success.

Future Work

The overarching goal of my future work is to attack fundamental problems that require both SE and robotics while helping deploy software and field systems at scale. Some of society's most formidable challenges, like engineering to mitigate climate change, require expansive autonomous fleets for environmental monitoring and manipulation. Assuring the quality of these systems requires bridging many gaps between theory and practice, as our current systems are painfully fragile. These gaps continue to inform my research on software for robots and pique my interest in the intersection. In addition to my established research pipelines, below I briefly describe new and promising extensions.

Code-Aware Robot Simulation and Scenario Generation. High-resolution physical simulations provide essential and cost-effective validation of robotic systems. However, robotic simulation is intentionally and architecturally separate from the inner workings of the software that reads sensors and commands actuators. Recent standardizations in robot simulation tools (SDFormat) provide a data structure through which concerns in the simulation can be linked to concerns in code. For example, the code implies that system behavior depends on temperature, but the simulation does not model temperature. This connection between simulations and programs can enrich robot simulation by making it aware of what is happening in the code.

Today, many robot simulation scenarios are hand-crafted by simulation designers (or the system developers themselves) who possess insights into the system's expected behavior. I aim to develop *scenario generation* approaches to explore ways to leverage knowledge of the system's code combined with insights from the robotics domain to generate nearly adversarial test scenarios. For example, we force a robot to localize in an increasingly feature-sparse environment until localization occasionally fails, then we add features until localization barely succeeds. We have observed that the stochastic outcome of these simulation scenarios is often due to models of the sensor noise. Test scenarios that barely pass can detect regression errors caused by other system modifications. This effort seeks to automatically generate test suites with scenarios that "push the limits" in many meaningful ways, thereby saving tremendous developer effort during prototyping and development.

Connecting Programs to the Real World. Program analysis relies on several powerful transformations of source code into abstractions: control flow graphs, dependency graphs, abstract syntax trees. These representations lift program analysis into abstract representations that model critical program properties, like domination and reachability. Recent work in robotics and artificial intelligence (AI) leverage *hypergraphs* that connect multiple levels of abstractions. For example, map coordinates (x_1, y_1) have an edge to a semantic graph node (*BlueChair*), and a trivial robot path can be represented by $[(x_1, y_1), (x_2, y_2), (x_3, y_3)]$ ***or*** $(BlueChair, YellowHallway, RedDoor)$. These kinds of abstractions have been fruitful for advances in AI, and this idea is to extend existing program analysis graphs with connections to semantic understandings of programs in the current context.

The idea is to bridge the gap between relationships between entities in program analysis and relationships between entities in the real world. This might enable program analysis to reason about the interplay of variable values and the future state of the physical system, such as: *"the integrator term of the PID will not wind up beyond threshold X in region Y of the map given the current plan and state estimation, with confidence Z."* My existing work begins to bridge this gap by inferring physical units for program variables, and future work seeks to make far-reaching connections. Inferring the semantic meaning of code within the context of an environment at runtime might be a gateway to assuring critical safety properties of autonomous systems.

Spatiotemporal Equivalence Classes, and Spatial Logics. I seek to develop *spatiotemporal equivalence classes* that represent sequences of system states in an abstraction that is invariant under translations in space and time while retaining sufficient descriptive power to make useful assertions about system properties. The idea is to only run parts of simulations that are unique with respect to these abstractions. *Temporal logics* have proven to be a robust abstraction to specify desirable properties of systems. However, temporal logics and their spatial logic extensions are currently insufficient to describe desirable properties of changing complex spatial relationships in dynamic environments. This line of research builds on a semantic understanding of what variables mean in the world [C-6, C-7, C-9, C-10] and adds ideas about orientation and frame-of-reference.

For example, a 2D robot might measure the distances to all obstacles within sensor range and compute a clutterness metric for that location. The idea is to use combinations of these descriptors as a vector that describes abstract properties of this environment. In simulation, these properties might help identify novel test scenarios, giving new notions of scenario coverage that re-invent robot simulation testing. Our initial observations in this area indicate promise in combining metrics that are egocentric (from the robot's sensor values) and allocentric (the relation between objects in the environment). Every incremental extension of spatial reasoning brings us closer to realizing a vision of verification of system behavior *during operation and in response to changing circumstances.* Cumulatively, new extensions to temporal logic could transform our ability to predict when robot systems will be safe and reliable.