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.

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. Unlocking the potential of autonomous systems in real-world environments will require new techniques and tools for reasoning about the system and the program together.

My research in program analysis is rooted in compiler technology but informed and enriched by field robotics. My experiences in field robotics have convinced me of the profound complexity and grand challenge of unstructured environments. I see field robotics and software engineering (SE) as complementary disciplines. 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 believe the intersection of SE and robotics will become an increasingly fruitful area, and I have published at top venues in both domains (SE: FSE, ASE, ISSTA; robotics: IROS, FSR, JFR, ICRA).

My research approach emphasizes three aspects: empirical evaluation, tool building, and collaboration. During my doctoral research, I collaborated with Berkeley hydrologists and Purdue SE researchers. I have built open-source tools and open datasets both individually (Phriky [C-7]) and collaboratively (Phys [C-10]). These endeavors are enriched by discussions about what level of empirical evidence was required by different communities. I admire that the field robotics community requires ideas to vetted outside the lab, in brutally realistic, outdoor environments (“simulation is doomed to succeed”). I find empirical evaluation to be a critical component of good science. My future research will continue to be characterized by rigorous evaluation, open tools and datasets, and collaboration, both cross-institutional and cross-disciplinary.

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 velocity to distance, or integrating third-party code that abuses the semantic conventions of middleware messages. Incorporating dimensions and physical units in programming languages has been long-studied, and theoretically solved with type annotations. However, this incurs time and toolchain penalties. With the surging popularity of the Robot Operating System (ROS), developers ‘voted with their keyboards’ and few adopted type annotations [C-8]. These inconsistencies hinder code reuse and expose developers to software defects within both their code and when sharing code.

My research has sought to provide 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. My work on Phriky and Phys implements dataflow analysis combined with ATI.

My research contributions in this area include: 1) a human study assessing the burden of making type annotations [C-9] (IRB# 20170817412EX); and 2) a method of ATI for ROS systems that is light-weight, intra-procedural, and semi-flow sensitive [C-6]; 3) implementing this method in an open-sourced tool, Phriky [C-7]; 4) an evaluation of 5.9M lines of ROS code, finding 6% of 3,484 repositories contain dimensional inconsistencies [C-8]; 5) a novel method of performing ATI with evidence in variable names using probabilistic graphical models (belief networks) [C-10].

My work on the annotation burden [C-9] applies more generally to Javascript type checking, where burgeoning interest in industry applying type systems to untyped domains has resulted in new tools (Microsoft → TypeScript,
The tool Phriky contributed to an NSF small-grant application I helped write (#NSF-CCF 1718040 $484,694), giving me familiarity with the process and funding model. Helping write a successful NSF grant showed me the value of having: 1) prior published work; 2) working prototypes, and; 3) the importance of expressing an alluring vision of the impact of the work.

This work is also part of a first step toward extending program analysis to the robotics domain by first connecting program variables to their real-world meaning. Only then can we hypothesize about how systems comprised of software and hardware will behave in the field.

**Field Robotic and Aerial Water Sampling**

My experience in field robotics upholds the adage: “Robotics is the science of cables and connectors.” Successfully deploying systems in the field requires correctly connecting a wide range of elements that span: the hardware-software interface, embedded system design and programming, wireless communication, robotic middleware, higher-level behavior, and all the interfaces between these elements. Aerial field robotics also requires operating in compliance with federal and institutional requirements. These efforts are inherently interdisciplinary, and I have benefited from collaborating with talented mechanical and electrical engineers, as well as conservation biologists, limnologists, and hydrologists.

In field robotics, my work pioneered the use of aerial robots for water sampling [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. My colleagues and I received a US patent [P-1]. The working prototype of our aerial water sampler helped us win funding (# USDA-NRI, $956,210). We extended this work with a novel altitude controller using a precise submerged sensor to enable unprecedented precision in water science datasets [C-5, J-4].

Field robotics provides SE with both an instructive proving ground and also an expanding horizon of challenges.

**Future Work**

The overarching goal of my future work is to attack fundamental problems that require both SE and robotics, and aim at 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 system 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 intersectional research. I eagerly look forward to exploring, over the next few years, several promising ideas on improving program analysis for robot software.

**Code Aware Robot Simulation.** 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 provide a data structure through which concerns in the simulation can be linked to concerns in code. This connection between simulations and programs can enrich robot simulation by making it aware of what is happening in the code.

Once deployed, it is often too late for a simulation to provide timely feedback because the models do not match current or changing environmental conditions or lack relevant details specific to the deployed system. The idea in this vein is that simulation should be used at runtime with an updated model of the environment and system, with the ability for the software to adapt to new conditions but with verifiable properties. Determining the right granularity to maintain simulation performance while yielding timely and useful information is a key challenge. Further, designing experiments to validate these techniques in the field requires a working implementation and understanding the concerns of field robotics practitioners.
Spatial Equivalence Classes and Spatial Logics. Spatial-temporal equivalence classes seek to find sequences of system states invariant under translations in space and time. The idea is to reuse parts of simulations and avoid simulation that are equivalent to simulations that we have already run. The challenge in this area is defining abstraction operators that exclude unnecessary details while retaining sufficient descriptive power to make useful assertions about system properties.

Temporal logics have proven to be a robust abstraction to specify desirable properties of systems. However, temporal logic is 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.

Connecting Programs to the Real World

Program analysis relies on several powerful transformations of source code into graphs: control flow graphs, dependency graphs, Abstract Syntax Trees. These representations lift program analysis into data structures that exhibit critical program properties, like domination and reachability. Recent work in robotics and artificial intelligence leverage hypergraphs that connect multiple levels of abstractions. For example, the 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 a connection to semantic understandings of programs in the current context.

The idea is to bridge the gap between relationships between entities in program analysis to 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 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.