SYSTEMS-LEVEL PROGRAMMING ENVIRONMENTS FOR ROBOTICS

SCOTT C. LIVINGSTON

keywords: programming languages, robotics, experiment methods

1. Motivation

Repeatability is a foundation of science, yet robotics experiments are notoriously difficult to reproduce. This hinders progress in research and translation of results into industrial settings. This was once a problem in the world of (pure) software, but has since been largely solved by standardized languages, libraries, and platforms. We argue that repeatability can be addressed by the practical and theoretical development of a programming environment for robotics. This position paper identifies research challenges in that direction and how it may be a high-impact domain for "cyber-physical systems."

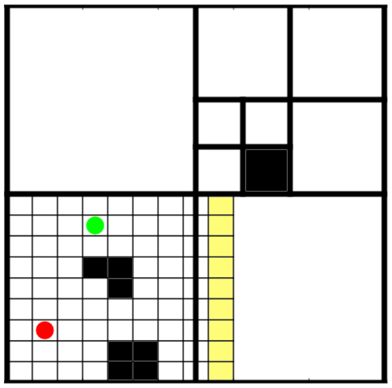
While none of the previous attempts at robot-specific programming languages have become widely-adopted, there has been recent progress on other fronts that could provide a basis for success. In terms of theory, recent work in contract-based design (e.g., [4]) provides rules to ensure overall correctness when composing modules that individually meet interface specifications. Progress in synthesis (e.g., [1]) has led to algorithms for converting formal specifications into finite-state machines realizing desired behavior. If formal specifications are regarded as providing a sort of high-level assembly language, then the aforementioned research provides algorithms for linking and assembling. Toward using these methods to control physical systems, recent work includes the introduction of motion planning algorithms realizing formal specifications (e.g., [2], [3]). The increasingly popular ROS (robot operating system) provides a common base on which to build software concerning robotics.

2. Proposed work

We begin with a challenge problem: create a toolchain for the development of robot software that guarantees specified properties are met *by the deployed system.* It should permit reuse of code on distinct platforms, or describe violated properties when this is not possible. Consider the following scenario. Suppose a person buys a new PR2 robot. While there is existing infrastructure that simplifies beginning on PR2s, she still must create roslaunch files to connect implementations of the planning and map-building algorithms to it that she wants to use. Though this is straightforward (and tedious), she will not automatically obtain properties about the composed system. Instead, these must be manually computed. For instance, while the map is being learned, will navigation requests be properly delayed or modified according to uncertainty? What is the maximum speed at which an obstacle may be approached before the robot cannot safely stop before collision? Note that answering the previous question depends on delays introduced by the various subsystems of a ROS graph and the hardware on the PR2; it is not merely a question of maximum deceleration.

The author is affiliated with the California Institute of Technology. Contact email address is `slivingston@cds.caltech.edu`.

As apparent from the above, while important infrastructure exists for building robots, engineers are still required to perform significant manual analysis to verify properties. Thus we propose as a research program the creation of a programming environment that, in addition to providing for composition of modules and code generation for target hardware, provides automation of checking desired properties. This will require breaking through the pure software perspective afforded by existing tools and studying how these interact with the physical entities of robot hardware.



As a motivating problem context, consider building a controller that realizes a specification tolerant to the uncertainties arising during simultaneous localization and mapping (SLAM). Task specifications consist of three parts: globally-scoped behaviors, class-based behaviors, and moving obstacle avoidance. The programming environment would support libraries for SLAM so that initialization code would be as simple as `import slam; M = slam.init(LaserScan, DiffDriveModel)`. During map learning, the current position estimate could be obtained with `p = M.currentpos()`. In the specification, globally-scoped behaviors depend directly on position, so corresponding code is in terms of `p`. Class-based behaviors depend on recurring structures in the map, e.g., "always eventually visit all offices." An example predicate is `p in offices[i]`, where `offices` is an array of rooms identified thus far in the map `M`. Moving obstacle avoidance would be achieved using a finer discretization. Indeed, it could be solved as a subroutine imported from another library. This subroutine would provide contracts concerning start and end-states, which would be used in turn by the compiler to infer realizability of the overall program. This basic setting is illustrated on the left. Notice the finer grid overlayed on a coarse quadtree. At compile-time, properties like probability of specification satisfaction can be inferred from estimates of uncertainty provided by the `slam` package.

3. Potential impact

The proposed work—creation of a programming enviroment for robotics based on formal methods—will require us to address open problems for "cyber-physical systems." Questions include: How should uncertainty propagate across contracts, yielding a probability of satisfying requirements for the overall system? How should time semantics be declared at interfaces among modules, or is it possible to assume a single timing model without loss of generality?

References

[1] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78:911–938, May 2012.

[2] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Proc. of the American Control Conference (ACC)*, pages 735–742, Montréal, Canada, June 2012.

[3] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. on Automatic Control*, 53(1):287–297, February 2008.

[4] Stavros Tripakis, Ben Lickly, Thomas A. Henzinger, and Edward A. Lee. A theory of synchronous relational interfaces. *ACM Transactions on Programming Languages and Systems*, 33(4):1–41, July 2011.