Why robot programming is hard

Tessa Lau
Savioke Inc.
Santa Clara, CA
tlau@savioke.com

August 25, 2014

I never thought I would be a roboticist. In grad school, our robotics team spent months teaching little robot dogs to play soccer and solving other seemingly simple problems. Far better to work in the world of software, I figured, where the systems I built could make a real difference in people's lives. Software could automate repetitive tasks in computer use and free people from mundane work. So instead I developed expertise in human-computer interaction and intelligent user interfaces. My chosen field, end user programming, has the potential to enable millions of ordinary computer users to customize and adapt systems to their own needs.

But the world is changing. Computing has moved off the desktop and into the smartphones, smart environments, and the world around us. Innovation is happening in the physical world, using computing to effect material change in our environment. The past decades have seen industrial robots increase manufacturing efficiency on the factory floor. Even more, we are now seeing robots tackle tasks from our daily lives. Robotic vacuum cleaners are changing the way we clean our homes. Robotic lawnmowers keep the yard tidy without having to lift a finger. Nest's robotic thermostat learns your habits and keeps your home at the optimal temperature. Google's driverless car could change the way we commute.

All these innovations have become possible through multiple advances: in sensor technology, which robots use to perceive the world around them [7]; in planning and navigation, which enable robots to move around in the world unaided [8]; in human-safe motion controllers, which let robots move their limbs without endangering the people around them [9]; in the ROS open source robot operating system, which enables roboticists to build on each others' work without starting from scratch [15]; and many more advances in the broader field of robotics.

Yet the holy grail of personal service robots remains elusive. I believe there are three main reasons. First, there is a wide variety of tasks that people want service robots to perform, ranging from folding laundry, to finding lost car keys, to picking up the kids' toys, or doing the dishes and putting them away. Second, these tasks must be performed in a variety of environments, including cluttered homes and kitchens with different affordances such as modern drawer pulls or faucet handles. Finally and most importantly, the robots themselves must be easy to operate by non-technical people.

We have started to see robots enter the marketplace that address two out of those three challenges. For example, iRobot's Roomba[1] only vacuums floors, but it does so in a variety of environments and is simple to use even by non-technical folks. On the other hand, Rethink Robotics' Baxter robot[2] can be easily programmed to perform a variety of tasks in light industrial settings, but its lack of mobility makes it less suitable for the home. As another example, Willow Garage's PR-2 robot has been programmed to perform a variety of household tasks, in a variety of environments, but can only be operated by highly

[1] http://www.irobot.com
[2] http://www.rethinkrobotics.com/products/baxter/

skilled roboticists.

Yet I believe the time is ripe for a breakthrough in personal service robotics, starting with robots that perform simpler tasks in relatively structured environments such as hospitals or hotels, and driven by better design and usable interfaces that enable them to be used by non-expert users.

1 Case study: Robots for Humanity

One important use case for personal service robots will be in-home care and assistance for disabled adults. The Robots for Humanity project [3], a collaboration between Willow Garage and Georgia Tech, aimed to develop technology that enabled persons with motor impairments to perform personal care tasks that they otherwise might not have been able to perform without assistance, such as picking up a dropped object, fetching a towel, or even scratching an itch. Its first prototypes enabled Henry Evans, a mute quadriplegic, to act in the world for the first time since a brainstem stroke left him paralyzed. Through physical therapy, Henry had regained the ability to make limited head motions (enough to control a cursor with his gaze) and to click a button with one finger. Using an interface custom designed for him, Henry was able to instruct Willow Garage's PR-2 robots to open cabinets in his home, fetch food from the refrigerator, and even dole out candy to trick-or-treating kids on his behalf.

Yet our experience with Robots for Humanity underscored how difficult it was for ordinary mortals to instruct robots what to do. Even using state-of-the-art robot visualization software developed at Willow Garage, simple teleoperation tasks such as opening a cabinet door was very difficult for Henry to accomplish. While Henry had only limited interaction capabilities, he was also strongly motivated because he has no alternatives. Able-bodied consumers, on the other hand, would quickly lose patience and simply fetch the towel themselves. If the state of the art in robot operation is so time-consuming, what hope do we have that ordinary mortals will be able to operate robots to perform these tasks in their own homes?

Granted, designing teleoperation interfaces for motor-impaired users is deliberately trying to solve the hardest problem first. Yet while Henry's case highlights the challenges of creating usable assistive robotics interfaces, it also illustrates the enormous benefit to humanity of enabling disabled adults to regain some of their independence.

With this goal in mind, personal service robots are more than a consumer toy, but a technology that could changes people's lives for the better. Why, then, are robots still so difficult to use? What makes robots so difficult to operate?

2 Robot programming

Wikipedia defines computer programming as follows (italicized text added for clarification):

> The purpose of programming is to find a sequence of instructions that will *[cause the computer to]* automate performing a specific task or solve a given problem.

If you replace computer with robot, we come to a definition of robot programming: finding the right series of instructions that cause a robot to automate a task or exhibit a certain behavior. Every robot makes available a certain set of functionality – its user-visible API. These are the primitive behaviors that have been programmed into the robot and can be activated by its user. End users must select which behavior to call when, in order to make the robot do what they want.

Many of the lessons learned from the fields of end user programming [4] and end user software engineering [10] also apply to robot programming, though robots' physical embodiment leads to additional challenges. I believe that the difficulty of the robot programming task boils down to several factors:

- **Direct contact with the robot/task**: programming is easier if the user can see and touch the robot and its desired task. Control from a distance makes the programming process harder [2]. For example, robots that are sent

into natural disaster zones as first responders may have to be operated from a distance, which increases the programming task's difficulty because the operator's senses are limited to what can be perceived through the robot's own cameras or sensors.

- **Task/environment variance**: if the task to be performed is always the same, the programming task is easier than if the goal task changes significantly or if the environment in which the robot must operate is changing. The program must be generalized to handle this variance, which makes the programming process more difficult.

- **Goal specification**: some tasks are easy to describe, such as navigation goals specified by a single point in 3-dimensional space. Other goals, such as washing the dishes or tidying up, are more difficult to specify. Successful completion may be more dependent on how exactly the robot performs the task. Generally speaking, the more difficult it is to frame the goal specification, the more difficult the programming task will be [11].

- **Amount of autonomy**: robots are gaining the ability to perform some functions autonomously, such as spatial navigation and object manipulation. The less autonomy exhibited by the robot, the easier it is to program the robot, because increased autonomy requires the programmer to develop more complex mental models of the robot's perceptions and behavior [12]. Teleoperation interfaces are fairly easy to use because there is generally a one-to-one mapping between the user's inputs and the robot's behavior. Once a robot starts to act autonomously, understanding its behavior (and debugging it!) becomes more challenging.

For personal robotics to truly succeed, regular people will need to use robot programming interfaces to tell robots what to do. These interfaces for robot programming will need to be simple enough for ordinary consumers to be able to pick up and use. In this article I detail some of the challenges that make robot control such an inherently difficult problem.

The explanation begins with a look at the state of the art in robot software today.

3 The state of the art

The Robot Operating System[16] is one of the software advances that supports the rapid development of new robotics technology. First developed in 2007 at the Stanford Artificial Intelligence Laboratory, ROS was maintained by Willow Garage from 2008-2013 and has recently transitioned into the stewardship of the non-profit Open Source Robotics Foundation.

ROS provides a distributed publish/subscribe messaging platform for a collection of *nodes* to communicate with each other. Each node encapsulates a small amount of robot functionality, and interfaces with the rest of the ROS ecosystem to make that functionality available through a common interface. For example, ROS nodes exist to stream sensor data from webcams and laser rangefinders, control the motors in robot arms, plan a path from one point to another, and recognize graspable objects in an image.

Before ROS, roboticists had to create the software to control their robots from scratch, each doing the same work to create device drivers and rebuild libraries for their specific robot's hardware. Now, ROS has been ported to around 100 different models of robots worldwide[3], enabling roboticists to quickly bring a new robot model online.

However, ROS provides a fairly low-level interface to robot control. Designed for roboticists, the RViz interface (Figure 1) displays all the sensor feeds coming from the robot and lets one send low-level motion commands to the robot such as rolling forward or rotating its shoulder joint. The figure shows two different perspectives on the same robot examining objects on a table in front of it. The left figure shows a third-person view of the robot and the table, rendered on top of a two-dimensional map of its environment (the gray pattern in the background). The green overlays on top of the tabletop objects indicate that they

[3] http://www.ros.org/wiki/Robots

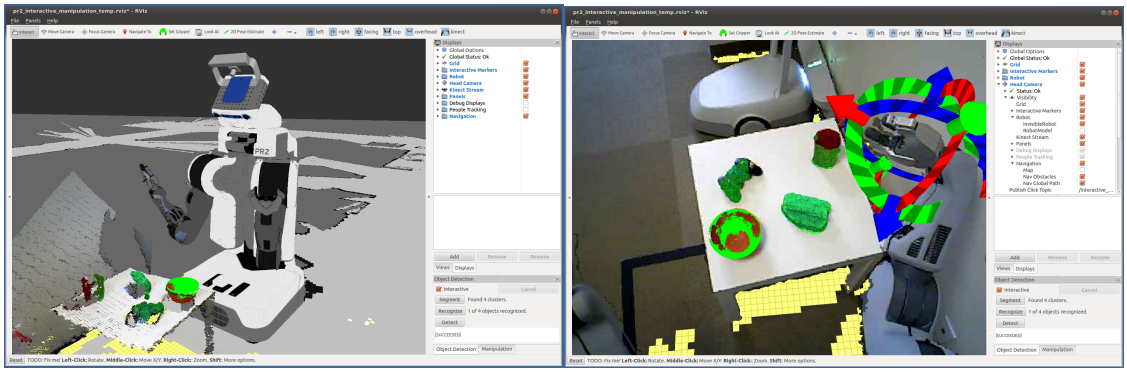

Figure 1: RViz interfaces for robot visualization and control

are recognized by the system as objects that can be picked up. The yellow grid pattern indicates horizontal surfaces that are detected, such as the floor. The right figure shows the view as soon from the robot's head-mounted camera. The red/green/blue arrows are control markers that can be used to rotate and move the robot's gripper in all directions.

4 Why robot programming is hard

The Rviz interface, while well suited for roboticists who need to examine the low-level sensor data and messages being processed by the robot, is not very user-friendly. It also illustrates some of the difficulties inherent in robot programming.

4.1 Concurrency

Many activities of daily living require coordinated action amongst multiple limbs. Anyone who has broken an arm, for example, understands the difficulty of trying to unscrew jar lids, pick up bulky objects, or chop vegetables with only a single arm. Controlling both arms simultaneously to perform coordinated actions is a matter of concurrent programming.

The problem becomes even more complex with robots that have many moving parts. For example, Willow Garage's PR-2 has two independently controlled 7-DOF arms, a head that can pan and tilt, a torso that can be raised and lowered, and a base that can move and turn in all directions. Robots are also equipped with a slew of sensors including cameras, depth sensors, laser rangefinders, touch sensors, and feedback about its electrical and motor systems. These sensors and actuators must be operated together in coordination to produce a desired robotic behavior.

Concurrent programming is notoriously difficult even for professional programmers due to the possibility for race conditions and the complexity of reasoning about what the program does [17]. Professional programmers learn to use advanced techniques such as multi-threading, event-driven callbacks, and coroutines in order to specify concurrent behavior in programs. Yet these concepts are notoriously difficult to use and debug, even for trained programmers; how can we expect non-technical users to be able to do the same?

In addition to the low level concurrency needed to control multiple robot parts, there is also a need to specify concurrent robot behavior at a higher level [1]. For example, a user may want to give the robot background tasks (such as cleaning or plugging oneself in when the battery is low) which should be scheduled in concert with foreground tasks (such as fetching a drink).

4.2 Uncertainty

Another challenge that robot programmers face is specifying how a robot should behave in an uncertain world. The world is constantly changing: any model that a robot could build of where things are located will be obsolete by the time it turns around [6]. People walk around, objects get lost, furniture gets moved. Moreover, limitations in current sensing technology make it difficult to build a perfectly accurate model of the world [9]. For example, differences in lighting make the same object look very different to an RGB camera. Therefore a robot cannot necessarily trust what it perceives, and it must be able to deal with the possibility that what it thinks is true is not actually true.

In addition, robot hardware itself often produces unpredictable results. Unlike software, mechanical parts can slip, burn out, or break. They take time to actuate and do not come to a stop immediately. Giving a robot the same instruction multiple times may or may not result in the same outcome each time.

Thus robot programs must have mechanisms in place to deal with uncertain truth and unpredictable hardware. Robot control loops usually sense the physical hardware during operation to determine whether a motion was carried out successfully. If an operator is not physically co-located or co-temporaneous with the robot, she cannot see whether the robot carried out the desired motion, so she must think through possible outcomes while writing a program and craft appropriate responses for each eventuality. For example, while instructing a robot to pick up an object, a programmer must specify what to do if the object is not *where* it thinks it is, is not *what* it thinks it is, or if the robot fails to perform the desired grasp. Should it simply try again? Move a little bit to get a better view and then try again? Ask a human?

In order to deal with these various types of uncertainty, professional programmers learn to use probabilistic models [5] and craft fault-tolerant code [18] with frequent error checking, which seems like too much to ask of casual programmers who just want a robot to do something useful in the home.

4.3 Commonsense reasoning

Reasoning about objects in the world requires some amount of common sense which has yet to be programmed consistently into robots. For example, a robot who has been asked to pick up and deliver a coffee cup may well select a grasp that involves dipping its fingers into the liquid and picking up the cup by its side wall. Although the grasp would be valid, this behavior would be judged somewhat undesireable by human observers.

As another example, robots that are capable of automated navigation may fail to observe human norms when navigating through human spaces. The custom of staying to the right when passing in a hallway, for example, can cause confusion if a robot charges down the middle of the hall and does not yield to oncoming passerby.

Endowing computers with common sense is still an active research area [14]. Techniques for making use of common sense reasoning have not yet made it into the vocabulary of professional programmers. Yet these concepts will be taken for granted by end users of robots, who will expect their robots to obey common social norms as they perform their tasks.

4.4 A 3D world

Unlike traditional computer interfaces which are limited to two dimensions, robot control requires manipulating objects in a three-dimensional world. Visualizing a 3D world on existing 2D displays requires the extra dimension to be projected down into two dimensions, giving rise to the popular "first person shooter" style of interface. These interfaces display the world as perceived from a virtual camera, where objects can be occluded by other objects. Additional camera controls are needed to pan/zoom around the space to overcome occlusion. These controls are often non-intuitive, even to seasoned computer users.

Occlusion presents a challenge because robots must perceive an object in order to manipulate it. Often times, the robot itself might occlude the object it is trying to manipulate! For example, consider a humanoid robot with head-mounted vision sensors using an arm to pick up an object from the table in front

of it. While moving the arm into position to grasp the object, the arm will likely come between the head and the object, obscuring the robot's vision.

One solution has been to mount additional cameras on different parts of the robot, such as on the robot's forearm or wrist, to get a better view on the object being picked up. However, making sense of multiple translated and rotated video streams can be confusing for those of us accustomed to perceiving the world only through head-mounted eye sensors!

Typical input devices such as mice and touchscreens are also only good for interacting in two dimensions. For example, tapping on a point in a camera image does not uniquely identify a point in 3D space, but a line of possible points. When telling a robot to put something down, how can it know which of those points was meant? Current interfaces, such as the RViz display shown in Figure 1, require complex camera manipulation and precise 3D positioning in order to uniquely specify the desired location.

5 End user programming for robots

The field of end user programming [13, 4] has developed a variety of solutions to enable users of software systems to create or modify software artifacts without having to learn the intricacies of programming.

The key insight for these EUP systems is to reduce the complexity of a programming task through simplification. The three primary methods include (a) visual programming: simplifying or eliminating tricky syntax through graphical representations; (b) domain-specific languages: limiting concepts to only those necessary in a particular domain of use, such as spreadsheet formulas; and (c) programming by demonstration or example: letting users focus on concrete examples rather than difficult abstractions.

However, this simplicity typically comes at the price of expressiveness. By reducing the complexity exposed in an interface, we also reduce the number or variety of functions that can be performed in that interface. In order to make robots that are more tractable for non-technical users, we will have to simplify key aspects of robot behavior, thus sacrificing robots' ability to perform a full range of behavior.

5.1 Higher-level primitives

One way to trade off expressiveness for simplicity is to abstract away details of robot behavior into higher-level primitives. This abstraction can be difficult for roboticists to accept, who have become accustomed to having fine-grained control over robot behavior and error handling. Yet abstracting away these details is going to be critical for end users to be able to develop a working understanding of a robot's functionality.

Roboticists can then implement these primitives using advanced techniques to solve the difficult problems encountered during robot programming. For example, a good autonomous navigation component could incorporate a set of recovery behaviors to try on navigation failure, such as turning around, waiting for people/obstacles to clear out of its path, and backing up and retrying. It could also bake in some concurrent strategies, such as turning the head to look in the direction the robot is going.

We made use of these autonomous functions to develop a system at Willow Garage called Continuous Ops (short for "Continuous Operations") that enabled casual users to control PR-2 robots through a web interface (Figure 2). The interface was inspired by the head of Willow Garage, who noted that even though he had a lab full of robots, he himself wasn't able to show off any of the functionality of his robots to visitors without a roboticist present to operate them.

In response, we developed a web interface, accessible from standard laptops or tablets, that provided the ability to tell a PR-2 robot to perform a variety of tasks. It could unplug itself from the power supply, navigate autonomously to anywhere in the bulding, position itself in front of a table, automatically identify graspable objects on that table, and pick up an object. The interface enabled non-technical users to activate several core autonomous functions of the PR-2 through an easy-to-use interface.

The drawback of this interface was that the user lost fine-grained control over exactly how the robot

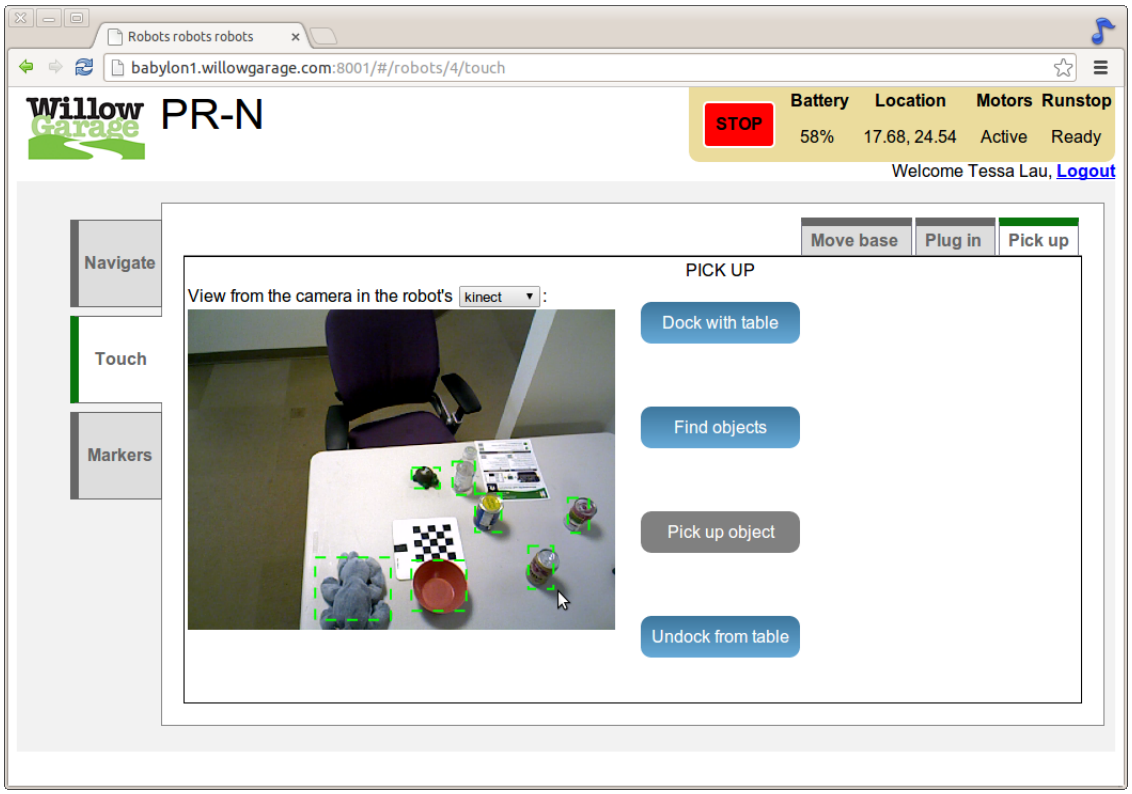

Figure 2: Continuous Ops interface for picking up objects on a table

performed these tasks. For example, the interface only supported using the robot's right arm to pick up an object; the left arm was unused. It did not allow the user to specify exactly how to pick up an object (e.g. from above or from the side), even though these parameters can be customized at a lower level of control.

The challenge in designing a simpler interface is to carefully craft exactly which functionality to expose, and which functionality can be safely hidden from casual users. For the intended users of the Continuous Ops interface, the provided functionality was sufficient and satisfied the requirements of being easy to use and always available. In fact, we heard of several cases where people used the interface remotely (e.g. while away at a conference) to show off the functionality of the robots in the lab.

5.2 Shared autonomy

Another way to simplify the programming problem for end users is to take advantage of *shared autonomy*. The idea behind shared autonomy is to fall back on teleoperator assistance when autonomy fails, as it will invariably do for the near future. A teleoperator does not necessarily have to be the person who wrote the program. Instead, they could function similarly to staff in telephone call centers, where a team of trained professionals handle incidents as they arise and get the robots back on track to fulfill their programmed goals.

With a human expert on call to get a robot out of situations it was not programmed to handle, we can simplify the programming problem for end users so that they do not have to specify error recovery behaviors for every single corner case.

As a stepping-stone towards full autonomy, this approach also enables the testing of end user programmable systems without needing a fully reliable set of autonomous primitives in place to support them.

This approach worked well during a multi-day deployment of a delivery system at Willow Garage. Users could request delivery of chocolates or snacks to their office, and the system would autonomously retrieve those items and deliver them to their office door. In the rare cases that the autonomous behaviors failed, several roboticists were notified by email and played the role of teleoperators. They initiated recovery behaviors to get the robots back on track. From the end users' perspective, the robot performed its tasks fully autonomously, sparing them from having to worry about specifying recovery behaviors.

5.3 New 3D input devices

Finally, video gaming technology is driving a new crop of input and output devices that make it easier to interact with and perceive the 3D world around us. These devices will make it easier for robots to perceive the world in three dimensions, and for people to instruct those robots using more natural gestures. Input devices such as the Razer Hydra, Leap Motion Controller, and Microsoft Kinect can sense motion in 3D, enabling robots to better sense motion and objects in the world. 3D output devices such as the Oculus Rift project stereo images in 3D so that people can perceive what robots are seeing in three dimensions.

These new devices could enable a new generation of robot control interfaces that make interacting with robots more natural and intuitive. For example, David Gossow at Willow Garage developed a research prototype called the PR-2 Surrogate which provided a fully-immersive teleoperation interface for the PR-2 robot. It used a Razer Hydra control to track your hand positions, which were then mapped directly to the robot's hand positions. It also displayed a stereo camera feed from the robot's head-mounted cameras into an Oculus Rift device, giving you a stereo view of what the robot was seeing. The Rift's head tracking capability was mapped to moving the robot's head, enabling you to look around the robot's environment just by turning your head. This demo showed that these new input/output technologies hold much promise for making it easier to interact with, control, and program complex robots.

6 Conclusion

Personal service robots have an enormous potential to do good for humanity by enabling people to live more independent lives and free them from mundane housekeeping chores. However, controlling these robots boils down to a small matter of programming. In this article I have explained why programming robots is inherently difficult, even for skilled programmers. Drawing lessons from the field of end user programming, I have proposed several paths forward that may enable the creation of more accessible interfaces for personal robots in the future.

References

[1] Maya Cakmak and Leila Takayama. Towards a comprehensive chore list for domestic robots. In *Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction*, HRI '13, pages 93–94, Piscataway, NJ, USA, 2013. IEEE Press.

[2] J. Y.C. Chen, E. C. Haas, and M. J. Barnes. Human performance issues and user interface design for teleoperated robots. *Trans. Sys. Man Cyber Part C*, 37(6):1231–1245, November 2007.

[3] T.L. Chen, M. Ciocarlie, S. Cousins, P.M. Grice, K. Hawkins, Kaijen Hsiao, C.C. Kemp, Chih-Hung King, D.A. Lazewatsky, A.E. Leeper, Hai Nguyen, A. Paepcke, C. Pantofaru, W.D. Smart, and L. Takayama. Robots for humanity: Using assistive robotics to empower people with disabilities. *Robotics Automation Magazine, IEEE*, 20(1):30–39, 2013.

[4] Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols, editors. *No Code Required: Giv-*

ing Users Tools to Transform the Web. Morgan Kaufmann, 2010.

[5] Adnan Darwiche. Bayesian networks. *Commun. ACM*, 53(12):80–90, December 2010.

[6] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[7] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In the 12th International Symposium on Experimental Robotics (ISER*, 2010.

[8] L.D. Jackel, Douglas Hackett, Eric Krotkov, Michael Perschbacher, James Pippine, and Charles Sullivan. How darpa structures its robotics programs to improve locomotion and navigation. *Commun. ACM*, 50(11):55–59, November 2007.

[9] Charles C Kemp, Aaron Edsinger, and Eduardo Torres-Jara. Challenges for robot manipulation in human environments. *IEEE Robotics and Automation Magazine*, 14(1):20, 2007.

[10] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3):21:1–21:44, April 2011.

[11] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, VLHCC '04, pages 199–206, Washington, DC, USA, 2004. IEEE Computer Society.

[12] Todd Kulesza, Simone Stumpf, Margaret Burnett, and Irwin Kwan. Tell me more?: The effects of mental model soundness on personalizing an intelligent agent. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1–10, New York, NY, USA, 2012. ACM.

[13] Henry Lieberman, editor. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann, San Francisco, CA, USA, 2001.

[14] Erik T. Mueller. Automating commonsense reasoning using the event calculus. *Commun. ACM*, 52(1):113–117, January 2009.

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. *ICRA workshop on open source software*, 3(3.2):5, 2009.

[16] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 2009.

[17] Nir Shavit. Data structures in the multicore age. *Commun. ACM*, 54(3):76–84, March 2011.

[18] Alex Wright. Contemporary approaches to fault tolerance. *Commun. ACM*, 52(7):13–15, July 2009.