Astronauts Must Program Robots

(A position paper for *To Boldly Go Where No Human-Robot Team Has Gone Before*)

Mark Yim
University of Pennsylvania,
Mechanical Engineering and Applied Mechanics
yim@ grasp.upenn.edu

Abstract

The traditional model for human-robot interaction is at the level of user-interface. For example, given a robot structure (say Robonaut - a relatively complex many DOF human-torso style robot) how should the robot be commanded by the human? We propose that for planetary exploration on the scale of Martian or Lunar habitation, a fixed robot/architecture will not be versatile enough for the variety of known and unexpected events. The astronauts will need to construct the tools to complete tasks from a rich set of modules. There needs to be new programming paradigms that suits the rapid resource constrained contingencies an astronaut may face.

Introduction

Do astronauts need to program robots? In one view astronauts are already programming robots depending on your definitions of *robot* and *programming*.

The word r*obot* has a wide range of definitions, depending on whether you are talking historically (about Karel Capek's play[1]), about science fiction, about toys, or about robotics research. By the broadest definition accepted by many robotics researchers:

> A robot is a machine or device that operates automatically or by remote control [2].

By this definition, a car using cruise control, a VCR recording a future TV show, even copy machines are robots. And as well they should be, they have the basic components of sensors, actuators and some form of computation just as Aibo (Sony's robot dog) or Asimo (Honda's humanoid robot). The difference is a question of level of sophistication, complexity and task.

The verb *program* in this context is defined as:

> to provide (a machine) with a set of coded working instructions"[2].

By this definition setting a VCR to record a show at a future date, setting the speed of a car for cruise control, or the number of duplex copies a photocopier should print are also examples of programming. As they should be, because in all of these instances, an embedded processor is receiving a set of commands by which to operate. Just as an Intel processor in a PC may receive a set of interpreted Java commands. The only difference again is the level of sophistication and complexity of the task.

Astronauts program scientific equipment to run tests, set the space shuttle to fly in specified modes etc. By these definitions it should be clear that astronauts have already been programming robots. The real issue is what level of sophistication and complexity of programming should be expected beyond simple mode settings or timed runs.

We argue that the level of sophistication must be significantly increased as the extraterrestrial tasks will require this especially in unexpected situations. At the same time, the robotics research community needs to develop programming paradigms that facilitate rapid implementation and is resource constrained yet still rich and powerful enough to construct robotic solutions for all reasonable contingencies.

Background

Need for sophistication

In Martian or Lunar habitation, human activities outside of controlled environments will likely be minimized for cost timeliness and safety concerns. Thus tasks such as habitation maintenance and repair will use remotely controlled mechanisms. These mechanisms can be tools that are autonomous or teleoperated, single function or multifunction machines, but in all cases these mechanisms are much more complex in operation than simple equipment such as a VCR or driving a car.

Conventional engineering thought is that most exploratory tasks that can be done by a human can be done with robotic technology, provided there is an understanding of the problem, time, and resources to develop robotic solutions. Without going into the argument about whether we should send robots and not humans to Mars or the moon, the main advantage of sending a human, is that the human can be creative.

When is creativity needed?

There are often events (or mistakes in modeling) that occur that are unexpected, especially in a case where exploration

is the primary focus. Errors and adjustments of varying magnitude occur on all space missions. These are clearly situations which we can't have developed all explicit technological solutions beforehand. As the complexity of tasks increases, the likelihood of severe unpredicted events increases. In the case of Lunar or Martian habitation, the scale of the task is immense, much greater than anything man has tried in space before.

If we accept that there will be a need for machines that can achieve a variety of (specified or unspecified) tasks, two questions remain, 1) What would be the context for programming these machines? 2) Do they need to be programmed on-site by the astronauts? Part of this depends on whether the robots are autonomous or teleoperated.

The question of **autonomy** is actually a question of degree. Truly autonomous systems are rare. One measure for the level of autonomy could be the rate of human communication with the system. By this measure, systems that are mostly autonomous are things like cruise missiles which are issued a command, then are never talked to again for the life of the device (albeit a relatively short life). On the other end of the scale, a model RC car which has constant communication from the driver would have little autonomy. The recent robotic Mars Exploration Rovers (MER) would fit somewhere in between receiving many commands, but not a constant stream.

More programming is required for higher levels of autonomy, but in most instances some level of autonomy aids in complex tasks. For instance, on the MER missions, the robots were commanded to go to way points and autonomously avoiding simple obstacles [3]. However, robot controlled autonomy is not the only place programming is required. User interfaces and data filtering are two examples.

We propose that the context for programming these machines should be a system modular in both hardware and software. The tried and true approach to any large complex problem is divide and conquer. Problems that can be broken down into simpler components each of which can be solved separately have a better chance of finding viable solutions. If we reverse the process and start with a set of modules that solve individual pieces we can assemble those pieces to conquer larger problems. We just need to start with the large enough set of modules so that it is likely that a solution exists in some combination. The tradeoff here is that space applications are mass and volume constrained; we cannot send an infinitely large module set.

In addition, another tradeoff is the granularity of modules. Theoretically, the finer grain the modularity, the larger space of possible assemblages of those modules and thus the greater the probability that a solution to any problem exists using those modules, but the longer and more difficult it is to implement that solution.

Historically astronauts have been very highly skilled highly intelligent people. It is likely most already know how to program in a variety of computer languages. So it is not a question of whether they should be trained to program, but rather whether sending the programming tools is an effective and efficient use of resources.

Programmability is a virtual feature taking little physical space, and it inherently increases the range of functionality of devices. If we want to maximize the likelihood of having a solution to unknown events, it follows that maximizing programmability will achieve this with minimal added resource cost.

Counter arguments

- **Local teleoperation is all that is required?**

Teleoperation of equipment by astronauts on Mars or the Moon will not have the long lag time and periodic blackouts that Earth-based teleoperation will have. Human creativity and adaptation can be utilized in teleoperation, and therefore may be enough to handle these remote operated contingencies.

The central argument of this paper is that tools will need to be constructed as problems are discovered - the form the tools take may not be known beforehand. The teleoperated user-interface must be constructed and "programmed" to fit the form of the tool.

The key idea here is that very likely, some form of autonomy may be incorporated as part of the user-interface. Small scripts that execute canned routines such as move forward 10 meters, or reach down with an arm until an object is touched. This type of added programmability comes with little cost to space and volume.

- **Programming can occur remotely from ground-based engineers?**

The Apollo 13 mission in 1970 is an example of a complex mission where unpredicted events nearly caused a disaster. Earthbound engineers' creativity (and duct tape) saved the lives of the astronauts including devising an hour long assembly sequence to mate a square CO_2 scrubber canister with a round hole. [4]

Whereas the repair of Apollo 13 heavily involved ground engineers telling astronauts what to do, this type of help won't work for Lunar and Martian habitation. In the Apollo case, the engineers had a good understanding of the problems, because they had very good models of the situation - exact duplicates of equipment). On Mars or the moon, problems are likely to occur which involve terrain interaction which cannot be duplicated exactly. This is

something that a person on site will be best able to analyze (feel, see, etc.) The communication lag from Mars may also prevent effective contingency solutions. The astronauts themselves will be the best ones to solve unexpected time-critical events.

- **We can send highly capable robots that can handle any contingency circumstance?**

A simple example of a very capable generic system that was not able to handle an emergency is the robotic arm on the recent Discovery space shuttle mission STS-114. Here loose gap filler needed to be removed. An astronaut rode on the end of the robotic arm to reach the desired location and performed the repair task. If the task could have been done without endangering humans in an extra vehicular activity (EVA), it would have been. The robotic arm could reach both loose gap filler areas, but was not used. partially because the robot arm was otherwise occupied with a 50 foot orbital booms sensor (OBSS) [5]. The arm was not expected to be used in this type of contingency case.

It doesn't make sense to have a special tool for every contingency, for example a special arm specially dedicated to "gap filler removal" and another for inspection and a third for deployment would take too much valuable space. Given the broad spectrum of possibilities on exploring a new planet, it is difficult to imagine any fixed architecture system that could handle any contingency circumstance.

- **Adding programmability and versatility will reduce efficiency and robustness?**

Any device that can do more than one thing cannot be more efficient or robust then a device optimized to do that one thing. This has been the way the space industry has been functioning since its conception. Virtually every satellite in space is has been redesigned and optimized for it's particular function with little regard to modularity, upgradability, reusability or versatility. It has worked before why change now?

The gap filler incident illustrates the problem of not having a versatility and programmability mindset. The shuttle arm is very versatile and could possibly have done the job (it reached the errant gap fillers) but it was holding the OBSS and had no place to put it. For well-scripted shuttle missions, the versatility mind-set may not make sense as there would be an equipment efficiency cost to gaining this flexibility (i.e. storage space for a 50 foot boom). However a habitation mission is much more complex and much more likely to have unscripted events. The cost of not having the flexibility and versatility to handle these broader contingencies will likely outweigh the loss of efficiency.

Some Solutions

Programming in this context does not refer to the classical situation where a programmer types at a keyboard. Rather, the astronaut is faced with a unexpected physical problem to solve. He must be able to solve the problems with the tools at hand in a reasonable amount of time, possibly minutes, in an emergency. So the two conflicting elements are 1) having a versatile and powerful tool set 2) minimizing time to achieve the task.

The broadest most versatile technological solution would be to have a team of engineers and the raw electro-mechanical components (motors, sensors, structural material) of say commercial companies Digikey, McMaster-Carr and Maxon Motors at your beck and call. Since this is obviously not feasible, the next best thing is to have a rich set of building blocks from which millions of different electromechanical tools can be created. In this context, programming is not just the implementation of a sequence of actions based on sensed inputs, but also the construction and configuration of a physical modular system.

One approach to achieve this is to use a robot system composed of mostly identical modules. Modular reconfigurable robotics has been studied for over ten years[6]. While much focus has been on the self-reconfigurability of these systems, they are just as effective (and perhaps more robust) as manually reconfigurable systems. Modular robotic systems are composed of modules each of which has a motor, sensors, and a computer. These systems have been demonstrated to do a wide variety of tasks including locomotion over rough terrain, using human equipment (riding a bicycle), crawling like a spider (Figure 1), digging in sand, using the snake-concertina gait through a gopher hole, manipulation of objects etc.

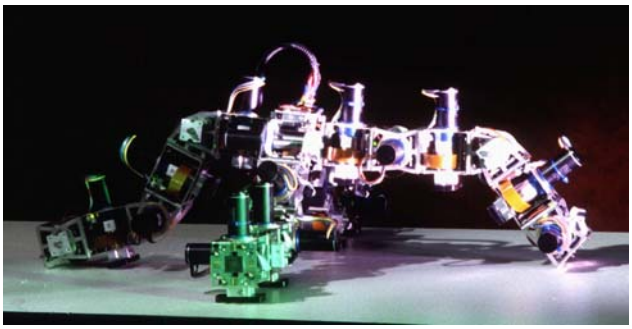

Figure 1: PolyBot G2 4-legged configuration with 24 modules.

Programming many degrees of freedom (DOF), as you might have with these modular robots, can be difficult. Many of the computational problems have time complexities exponential in the number of DOF. However,

there have been a few approaches aimed at making the programming of these systems easy and intuitive.

MIT's media lab has developed systems called Curlybot and Topobo[7] which use a "programming by demonstration" technique. The mechanisms' joints are physically manipulated; these motions are recorded and stored. The system can then use these motions played back, cycled or otherwise manipulated. This is similar to "puppetry" and an expanded version of when industrial robot arms used a "teach pendant" to input way points. The difference here is that a very large number of DOF can be taught to the robot in a very intuitive rapid and distributed manner.

Topobo is focused on the study of learning with children. It is fairly clear that implementing simple motions even with many degrees of freedom is straight forward, however achieving precise and coordinated or complex motions is still difficult. For example, some teenaged users of the system expressed frustration in not being able to get their creation to "walk" and settled on "crawling".

At PARC the PolyBot system uses a "posable programming" technique[8] combined with an XML based scripting language called PolyKinetic™. The most relevant demonstration of what is capable from this system occurred at a one day tutorial at the IROS 2003 conference.

After a morning of learning how the system worked, the participants entered a competition with a set of PolyBot modules and the PolyKinetic programming environment. Since the venue for IROS 2003 was in a casino in Las Vegas, the final task was to create a robot that could gamble:

- walk up to a toy slot machine,
- place a poker chip into it,
- depress a lever,
- catch the winnings (ejected poker chips) in a cup.

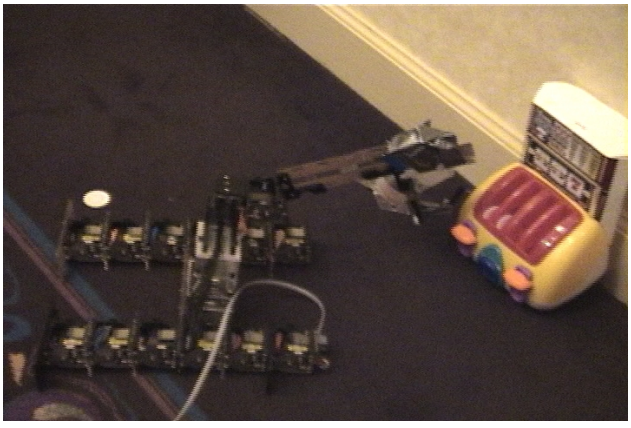

Figure 2. Double snake approaching the toy slot machine.

Participants were able to use features such as toolboxes to create user interfaces for teleoperation and parameters. One group constructed a design that contained a "double-snake" locomotion platform (Figure 2) as well as an arm and chip deployment mechanism. After the three hour design/construction time limit, they demonstrated almost all the tasks, only barely missing catching the chips as they were ejected from the toy slot machine. It may be noted that one of the 6 member team in this case was a senior robotics researcher at JPL. (videos can be seen at http://www.parc.com/modrobots/chain/polybot/IROS.html)

This demonstrated that a group of researchers could configure a set of modules and program them to solve a relatively complex task all within tight time constraints.

Tasks like this could be translated to more NASA relevant missions.

New programming models need to be developed that allow astronauts to quickly assembly robots and program them for mission relevant tasks.

Just as hardware elements may combine to form robotic structures like arms for manipulation or digging, software elements may be created to perform subtasks for example controlling an arm for position control or force control. There are several examples of software components, such as JPL's CLARAty project[9]. CLARAty "Coupled Layer Architecture for Robotic Autonomy" is a repository of software modules based in C++. However the composability of these elements is the key. Having astronauts coding C++ in space may not be optimal, especially if it is time consuming.

The greatest need is in combining the hardware and software modules in a rapid and intuitive manner for a deployable system. Solutions must be generated not in months sitting in a sterile lab, but in hours (or minutes) in the field. Posable programming is intuitive and easy to implement, but not powerful enough. A more powerful method that can enable sensor interpretation and feedback control would work better.

Conclusion:

Many of the arguments presented here are based on the presumption that the complexity of Mars and Lunar habitation is much greater than anything seen before. And if it is true, then perhaps the level of sophistication in programming needs to be extended to a similar level of complexity. The main problem is that it is difficult to quantify complexity in missions and perhaps even more difficult to see whether strategies such as on-site programming can do anything to alleviate unexpected events.

One process for testing the validity of these approaches is to run case studies or explicit experiments.

Finding the best solution may include the DARPA grand challenge approach. Create a competition where engineers may bring whatever equipment, robotic modules, and software that they want (as long as it satisfies mass and volume constraints). Give them a situation they must solve: e.g. your team is on the moon, a meteorite has hit your communications antenna, develop a robot to go out there insert this rod into the communications array. You have 2 hours.

References

[1] Capek K, *RUR (Rossum's Universal Robots)*, 1921 Play originating the term "Robot"

[2] http://www.dictionary.com

[3] Biesiadecki J J., Baumgartner, E.T., Bonitz, R.G., Cooper, B.K., Hartman, F.R., Leger, P.C., Maimone, M.W., Maxwell, S.A., Trebi-Ollenu, A., Tunstel, E.W., and J. R. Wright. "Mars exploration rover surface operations:Driving opportunity at meridiani planum". In *IEEE Con-ference on Systems, Man and Cybernetics*, , Hawaii, USA, October 2005

.

[4] Jones, E, "The Frustrations of Fra Mauro: Part I" The Apollo Lunar Surface Journal, 1995

[5]www.space.com/missionlaunches/050731_sts114_gapfillers.html

[6]Yim, M., Roufas, K., Duff, D., Zhang, Y., Eldershaw, C., and Homans, S.,"Modular Reconfigurable Robots in Space Applications", Autonomous Robot Journal, special issue for Robots in Space, Springer Verlag, 2003.

[7] Raffle, H., Parkes, A. and Ishii, H. Topobo: A Constructive Assembly System with Kinetic Memory, *Proceedings of Conference on Human Factors in Computing Systems* (CHI 2004), (Vienna, Austria, April 26 - April 30, 2004).

[8] Golovinsky, A., Yim, M., Zhang, Y., Eldershaw, C. and Duff, D., "PolyBot and PolyKineticTM System: A Modular Robotic Platform for Education," *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, (ICRA), New Orleans, April 26-30, 2004

[9] I.A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, "CLARAty: Challenges and Steps Toward Reusable Robotic Software," submitted to the International Journal of Advanced Robotic Systems