Auton Robot
DOI 10.1007/s10514-006-9020-9

Building robot systems to interact with people in real environments

Bruce A. Maxwell

Received: 16 March 2006 / Revised: 5 October 2006 / Accepted: 14 November 2006
© Springer Science + Business Media, LLC 2006

Abstract Interacting with people poses a unique challenge for robots. While issues such as safety are well known, issues such as appearance and intuitive human-robot interfaces are also critical to success. The American Association for Artificial Intelligence Robot Competition has encouraged the development of robots that interact with people through competitions, challenges, and exhibitions. This paper describes the robot systems developed for eleven different events over eight years from 1998–2005 and examines the factors that led to both successes and failures. The primary lesson is that simplicity and robustness in sensing, behavior, and human-robot interactions helps to counteract the uncertainty caused by people.

Keywords Robot · Human-robot interaction · Teleoperation · Autonomous robots

1 Introduction

Building robots to interact with real people in real environments poses many challenges to both the robots and their developers. Situations you never imagined occur with regular frequency outside of the laboratory; and they teach us a lot about designing robots for the real world.

The American Association for Artificial Intelligence [AAAI] Robot Competition and Exhibition plays a significant role highlighting current robot capabilities and challenging researchers to come together and compare their approaches and methods. The competitions force us to move our robots into new environments, challenging them to interact with people in foreign territory. They also remind us about the importance of software engineering techniques, testing, integration, and more testing.

From 1998 to 2005 the author brought eight different teams of undergraduate students to the AAAI competitions where we participated in eleven different events using three different robot platforms. The 1998 team was a group of undergraduates from the University of North Dakota Computer Science NSF Research Experiences for Undergraduates program. The 1999–2005 teams were groups of computer science and engineering undergraduates from Swarthmore College. The undergraduate teams ranged in size from two to ten, and only in 2003 and 2004 were the same students working on the same robots for the same event. Except for the teams in 2000 and 2005, most of the students had never taken a robotics course or worked with a research-grade robot. Judicious re-use of the code base, and the basic concepts contained therein, as well as a solid platform for visual sensing that has continued to evolve, permitted each team to put together a robot system that was capable of performing the specified task.

The primary purpose of participating in the AAAI competitions was to introduce the students to robotics and to engage them with researchers and graduate students from other institutions. Over the course of eight years of robot development, however, we have also contributed to the technical development of the field, especially in the areas of social robot visual sensing and human-robot interfaces for robot teleoperation (Maxwell et al., 1999, 2001, 2003, 2004b). Our focus throughout has been on tasks that require some kind of human interaction, either between the physical robot and a person in the case of a social robot, or through an interface for remote operation of a robot for Urban Search and Rescue [USAR]. People add an element of uncertainty and

B. A. Maxwell (□)
Swarthmore College,
Swarthmore, PA 19081, USA
e-mail: maxwell@swarthmore.edu

unpredictability to the mix that is not necessarily consistent from year to year. For example, some years people crowded around the robots, other years they ignored them.

This paper looks at the overall trajectory of system development over eight years, focusing on how the systems have evolved and the reasoning behind those decisions. Having to test and demonstrate each of those systems at the AAAI competitions provided significant feedback that we tried to integrate into our decisions about system development the following year. In a sort of Darwinian selection process, the hardware and software that has survived from year to year are, in a very real sense, the fittest, most robust pieces of the robot system. There are actually a few functions from the 1998 robot system present in the vision module used in 2005.

2 Changing tasks, changing robots

During the eight year span our robots participated in three different events: Hors d'Oeuvres Anyone? (1998–2002), Urban Search and Rescue (2000–2004), and Open Interaction (2005). The Hors d'Oeuvres Anyone? event tasked robots to serve snacks to conference participants in a variety of different environments. From 1998–2000 the serving environment was a conference festival or reception with hundreds of people in a reasonably natural setting, interacting with one another in a typical fashion. Figure 1 shows our robot Alfred serving snacks in 1999. In 2001 the event took place in a cordoned off pen with a small number of people, and in 2002 the event was less constrained spatially, but the serving component did not take place during a conference reception. The task was expanded, however, to include an information serving task in the hotel lobby during coffee breaks, providing a more natural setting for the event.

The Urban Search and Rescue task, which started at AAAI in 2000 and was hosted by AAAI through 2004, challenged teams to develop autonomous and remotely operated/directed robots capable of searching a disaster scene. The National Institute of Standards and Technology [NIST] developed a mock disaster arena with three areas: red, orange, and yellow. The Red area challenged the mobility of robots and was difficult or impossible for most wheeled robots to enter. The Orange area was accessible to off-road style robots and presented an intermediate challenge to mobility, and the Yellow area was accessible to research-grade wheeled robots so that teams could focus on other aspects of robot design such as communications, user interfaces, sensor development, and assisted or fully autonomous systems. Figure 2 shows one of our robots in the yellow area near a simulated disaster victim.

In 2003 the Hors d'Oeuvres Anyone? task changed to the Open Interaction event where robots were asked to interact with people for whatever purpose the teams wished. A common theme among participants has been humor or providing information about the conference itself. We participated



Fig. 1 Alfred serving hors d'Oeuvres at the AAAI National Conference in 1999



Fig. 2 Frodo in the Urban Search and Rescue event at IJCAI in 2003

in the event at AAAI in 2005 with a robot that attempted to demonstrate an emotional model that could affect the physical actions of the robots. In addition, we tried to make the interaction humorous for the human participant by having the robot make fashion statements about the person's shirt color (Hollinger et al., 2006). Figure 3 shows one of our robots wandering through the lobby area in search of people.



Fig. 3 Sam wandering and commenting on shirts at AAAI in 2005

Table 1 provides a summary of the individual robot systems, and Fig. 4 shows photos of most of the robots as they competed in the events. The remainder of the paper describes the hardware and software organization of the robot systems, followed by an analysis of how the systems evolved over time and what pressures caused that evolution. Finally, the paper concludes with some thoughts on robot system development, with a focus on lessons that are especially pertinent to students creating systems for the real world in an academic environment.

3 Hardware

Over the course of eight years we used three different robot platforms and a stationary workstation (Alfredo in 2000). The three robot platforms were an ActivMedia Pioneer I with a laptop (Fig. 4(a)), a Nomadics Super Scout II with an onboard Pentium (Fig. 4(b) and (e)), and an RWI (later iRobot) Magellan Pro, also with an onboard Pentium (Fig. 4(c), (f–i). All three are small ($\leq$ 45 cm tall, and $\leq$ 50 cm radius), research-grade robots with two individually powered parallel wheels and a castor for balance. All three have some capacity to power additional devices from their batteries, and all three can carry payloads of 10–12 lb, although either modification puts significant strain on their battery life. We used the Pioneer because it was the only available research robot within the budget of the project in 1998. Similarly, the Scouts were the best available low-cost small robots in 1999. The Magellans were the prizes for our first place finishes in the 1999 and 2000 competitions.

All of the robots were equipped with cameras, and these evolved from a Quickcam connected to the serial port in 1998, to a color CCD camera and framegrabber in 1999 and 2000, to Canon VC-C4 pan-tilt-zoom [PTZ] cameras from 2001 on. The switch to a CCD camera and framegrabber enabled faster video speeds with higher quality than available on a Quickcam. The move to a fast, light, easily controllable pan-tilt-zoom camera enabled more sophisticated human-robot interaction as well as eliminating the need for multiple cameras in order to visually examine different directions.

Interaction with people caused us to make a number of modifications to the robots in order to improve the safety of the robots and enable effective human-robot interaction. The first modification we made to both the Pioneer and the Scout was the addition of bump sensors placed low enough to detect someone's shoe. During testing we found that the robot's sonars were blind to a person's shoes or feet and the legs did not always show up well enough to keep the robot at a safe distance. The low bump sensors solved the problem. (It is interesting to note that two of the members of the 1998

Table 1 Summary of our AAAI robot competition entries from 1998–2005

	Year							
	1998	1999	2000	2001	2002	2003	2004	2005
Tasks	Serving	Serving	Serving USAR	Serving USAR	Serving USAR	USAR	USAR	Fashion-Conscious Robot
Robots	R.U.S.T.Y.	Alfred	Santino[a], Mario[b] & Alfredo[c]	Frodo & Gollum	Frodo & Gollum	Frodo & Gollum	Frodo & Gollum	Frodo
Platform	Pioneer I	Super Scout II	Super Scout II[a] Magellan Pro[b] Workstation[c]	Magellan Pro	Magellan Pro	Magellan Pro +	Magellan Pro +	Magellan Pro +
Hardware	Laptop Quickcam Microphone Speakers Sonars Bump Sensors Whisker Sensors	CCD Camera Microphone Speakers Sonars Bump Sensors Front Bumper	CCD Camera[ac] Microphone[ac] Speakers[abc] Sonars[ab] IR Sensors[b] Bump Sensors[ab] Front Bumper[a] LCD Monitor[a] 19″ Monitor[c] Movable Tray[a]	PTZ Camera Speakers Sonars IR Sensors Bump Sensors	PTZ Camera Speakers Sonars IR Sensors Bump Sensors Keyboard LCD Monitor	PTZ Camera Sonars IR Sensors Bump Sensors	PTZ Camera Sonars IR Sensors Bump Sensors	PTZ Camera Speakers Sonars IR Sensors Bump Sensors
Communication	Shared Memory	Shared Memory	Shared Memory	IPC	IPC	IPC	IPC & GCM	IPC & GCM
Modules	Navigation Vision Speech Recognition Recorded Voice Controller	Navigation Vision Speech Recognition Recorded Voice Controller	Navigation Vision Speech Recognition Synthesized Voice Synthesized Face Robot Communication Controller Mapping (USAR)	Navigation Vision Synthesized Voice Controller Teleoperation (USAR) Mapping (USAR) Planning (USAR)	Navigation Vision Information Interface Synthesized Voice Teleoperation (USAR) Mapping (USAR)	Navigation Vision Mapping Game-Interface v1	Navigation Vision Mapping Game-Interface v2 Robomon	Navigation Vision Recorded Voice Emotional Model Controller

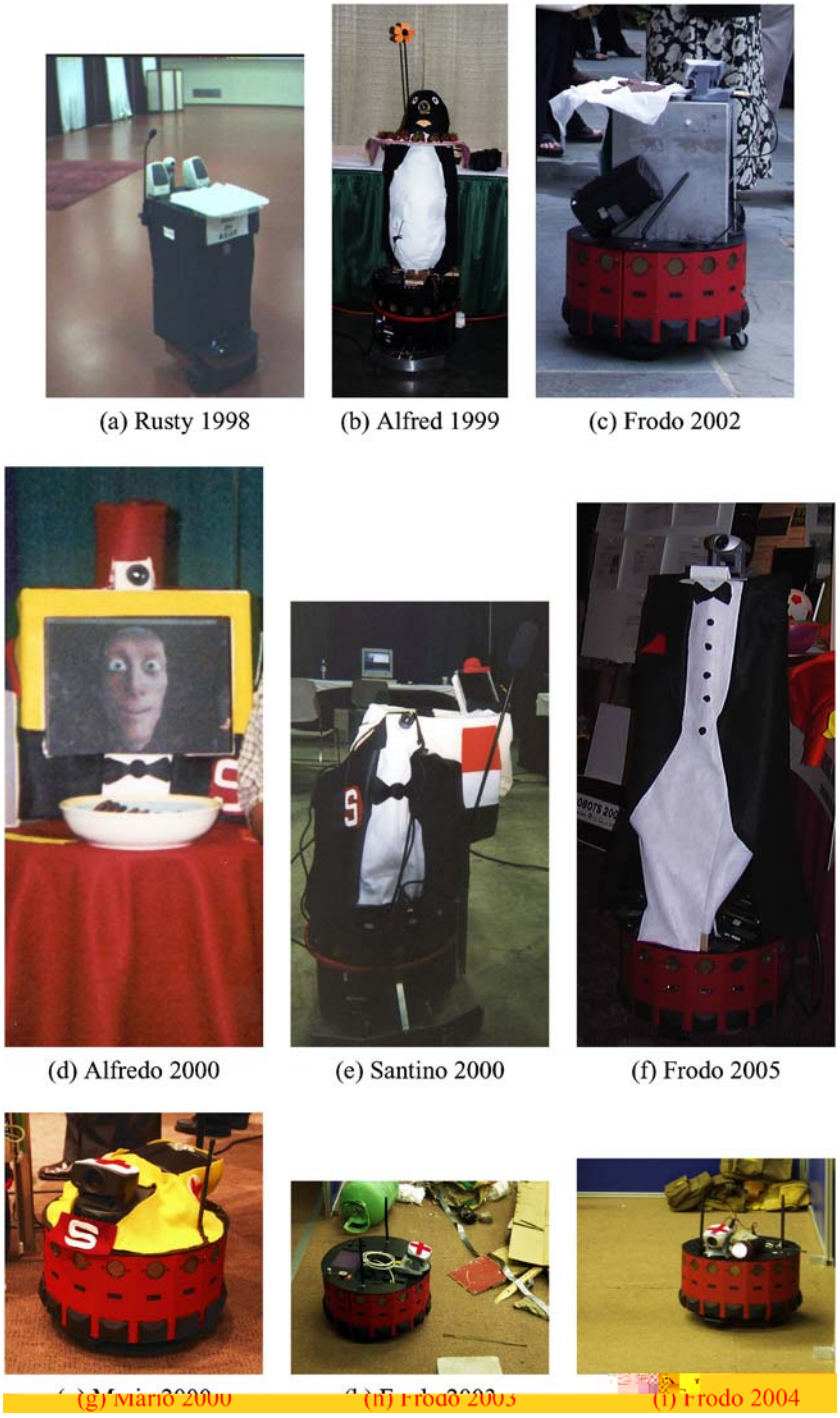

Fig. 4 Photographs of the robots at the AAAI competitions

robot team went on to work for ActivMedia and that the Pioneer II, the successor to our Pioneer, had the option of a set of low bump sensors; they are standard equipment on the PeopleBot.) For the Pioneer we put whisker sensors—wires soldered to a switch—in front of the wheels on either side.

On the Scout we built a sturdy bumper with four contact sensors on it and attached it to the front of the robot so it was about 4 cm above the ground, as can be seen in Fig. 4(b) and (e). It also served as a safety measure to keep the robot from pitching forward, as the Scout has no support in front of the drive wheels. Because the Magellan's IR sensors work well enough at short range to detect legs, we have never felt a need to add low bump sensors to that platform.

The second major modification driven by human-robot interaction was to add height to the robots. In Fig. 4, note that all of the serving robots have been built up to some degree, typically adding about 1 m to their height. The height is necessary for two reasons. First, people don't

like to bend down to interact with a robot or get their hors d'oeuvres. Second, it is difficult to detect people visually from a height of 40 cm. For example, in 2005 when we used the Viola-Jones face detection algorithm—which uses facial features rather than skin color—height became especially important because people's features change significantly when the camera is looking up at a steep angle (Viola and Jones, 2001). In all cases we used lightweight materials to minimize the load on the robot and felt to cover the structure underneath and give the robots a pleasing appearance. In Fig. 4(f), for example, the camera is sitting on a 1 m long aluminum rod and the rest of the structure is cardboard.

The third major modification driven by human interaction was upgrading the processor and wireless hardware on the Magellan robots. In particular, we needed to speed up video transmission for the USAR task. Improving the processor helped us to implement compression algorithms, and the additional wireless bandwidth sped up video transmission. In 2003 and 2004 we upgraded the processors from 450 MHz Pentium Pro systems to 900 MHz PIII systems, reducing the power requirements in the process. We also upgraded them to 802.1 lb wireless bridges, giving them significantly higher practical bandwith and again reducing power needs. As part of the upgrade we also replaced the main power board with a simpler custom designed board. The old board was prone to failure, and the new CPU did not require as many source voltages. Anyone interested in the plans for the board can contact the author.

In addition to the above modifications, human interaction taught us two other lessons about hardware. First, speakers are an extremely important part of the system, both their placement and quality. In a large room with many people talking, it is important to have loud speakers with minimal distortion. We spent much of our weight budget on speakers powered with four D batteries in order to achieve that goal. The speakers are visible in Fig. 4(c), pointed up towards where a person would be when interacting with the robot. Taller robots can use smaller speakers by placing them closer to the person, which works better in a crowded situation.

Second, physical appearance makes a big difference in the way the robot is perceived. Kids hugged Alfred in 1999 (Fig. 4(b)) and Frodo in 2005 (Fig. 4(f)), but Frodo in 2002 (Fig. 4(c)) was not as successful in getting attention, despite being the most capable and robust serving robot we had developed. Overall, our robots with a more anthropomorphic appearance tended to be more successful in human-robot interactions, and had more of them than our robots with a more mechanical appearance. This finding is not surprising given that people tend to anthropomorphize robots and find it easier to interact with items with a more human appearance (Nass et al., 1995). While this may seem obvious, physical appearance is often the last thing considered when developing a robot system because the focus of development is on software, hardware, and integration.

4 Software organization

4.1 Low-level computer-robot communication

At the most basic level a program running on a computer onboard the robot must communicate via some port to the microcontroller or subsystem that actually controls the robot's motors and sensors. Most commercial robot systems provide at least one layer, and in some cases multiple layers, of abstraction between the raw information flowing to and from the microcontroller and the programmer's code.

The Pioneer I, for example, came with the Saphira package, and the Magellan Pro came with the Mobility package. Both were significant pieces of software that permitted the user to fairly easily implement tasks such as moving while avoiding objects. Both were also intended to integrate multiple software modules handling different aspects of the robot's task. However, both had steep learning curves and significant overhead (especially Mobility), making them difficult for an undergraduate to pick up in just a few weeks. These packages also did not work across multiple platforms. In our case, from 1999 on we were using both the Scouts and the Magellans for development, and Mobility would not run on the Scout.

The Scout's interface was the most basic and was a library of calls that enabled the programmer to specify wheel velocities and get information from the sensors. To make our code uniform across different platforms, therefore, we implemented our own abstraction layer for the Magellan, called Mage, which created a library of calls for the Magellan Pro that were identical to the Scout interface, making the C code portable between the two systems with a few well-chosen flags in the makefiles. We made Mage available to the community in 2001, and anyone interested in working with it can contact the author (Maxwell et al., 2001).

While we gave up some of the easy object avoidance and navigation features of Mobility by implementing our own low-level library, it significantly reduced the learning curve for the students to get started working with the robots, it permitted us to develop a single navigation module that would run on both robot systems, and it gave us more precise and responsive control of the robots' physical motion.

4.2 Module organization

From the first year of our participation in AAAI, we designed the robot(s) to have capabilities including visual sensing, speech recognition, speech synthesis or recorded playback, navigation, and a control module for managing the robot's

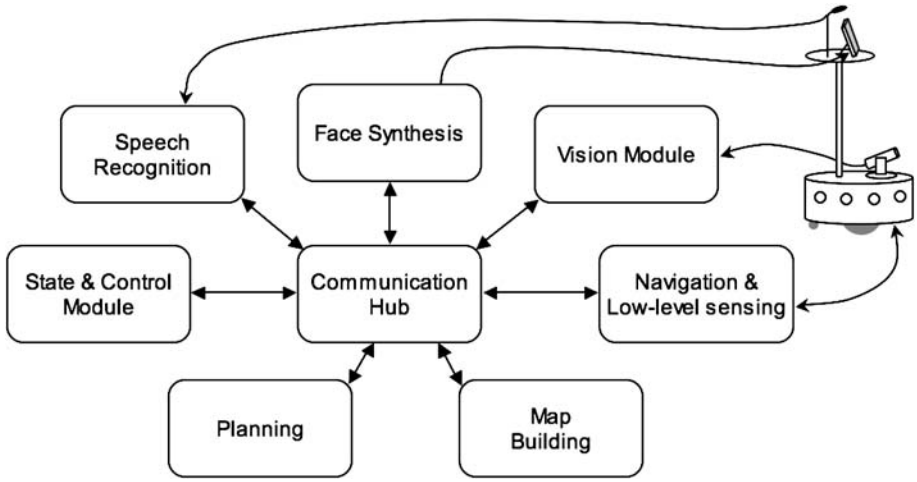

Fig. 5 Diagram of REAPER architecture. Modules handle raw sensory data and pass on synthesized information to decision-making modules. All modules communicate via a centralized method

state. Given the different, asynchronous requirements of each task, and following good software engineering principles, we designed a modular software architecture. Modularization also permitted independent development and testing of each capability by small teams of students. We maintained the same fundamental concept of multiple independent modules throughout all of the robot systems.

The basic concepts are embodied in what we called the REAPER [REflexive Architecture for PErceptual Robotics] architecture (Maxwell et al., 2001). The purpose of each module is to handle one of the following: sensing, reflexes, control, communication, or debugging. The fundamental concept behind REAPER is that the central control module—whether it is a state machine or other mechanism—cannot process a flood of sensory data in real time. Nor should the central control module make low-level decisions like how fast to turn each wheel ten times per second. It does require real-time updates of symbolic information indicating what is going on in the world around it. To provide this higher level symbolic information the sensor and reflex modules gather and filter the raw sensor information, handling all of the preprocessing and intermediate actions between high-level commands or goals. A diagram of the REAPER concept is given in Fig. 5. For a more detailed description of the design of a single module, (Maxwell et al., 2003) gives a complete description of the vision module.

The REAPER design methodology incorporates a number of the lessons learned about robot software architecture over the past two decades. In particular, it follows the recommendations of Bryson that autonomous agent architectures need:

- a modular structure,
- a means to control action and perception sequences for complex tasks, and
- a mechanism for reacting quickly to changes in the environment (Bryson, 2000).

REAPER is related to behavior-based systems, such as those proposed by Brooks, in that it distributes low-level interpretation and control of sensors and actuators to modules (Brooks, 1986). These modules run in parallel, and those connected to actuators are designed to react to changes in the environment on their own. Within the modules, a subsumption approach or weighted mixing of a set of behaviors are useful ways of generating appropriate low-level actions.

REAPER is not a pure behavior-based system, however, because it is designed to facilitate the mixing of symbolic and subsymbolic, or reactive strategies. In particular, the sensing and acting modules receive symbolic commands from a controller module that tells them their current set of low-level goals. Thus, REAPER also falls into the category of hierarchical architectures, which facilitate the development of action and perception sequences for complex tasks. While our implementation of REAPER has so far been a two-level architecture, our fundamental approach can easily support a three-level architecture with a knowledge-representation and reasoning system as the top-level. The overall controller, in this case, could be either the middle level, as in Gat's ATLANTIS architecture (Gat, 1998), or the top-level as in Bonasso's 3T architecture (Bonasso et al., 1997).

It is important to note here that all of our robot systems are based on finite state machines as their primary top-level control mechanism for the following reasons.

- FSMs are easy to design and debug for simple tasks such as serving snacks.
- FSMs can be designed to gracefully exit an action sequence while attempting a task, such as serving, that goes wrong due to sensor errors or interaction failure and return to a state such as Wander that enables the robot to try again.
- FSMs can easily be designed with generic states, such as Idle or Stop, into which the module can be put by an external message.

Finally, REAPER's modular design facilitates cross-platform and multi-platform development and simplifies debugging and testing of the individual modules. This is particularly true in the case of the sensing modules, which we extensively tested on their own before putting them on the robot platform. As noted by Bryson, modularity in design has become a staple of robot architectures for just these reasons (Bryson, 2000).

4.3 Self-monitoring architecture

There are several issues that arise in real systems, however, that can make modular systems difficult to handle, especially in a timed, high-pressure situation. Our experience in the AAAI competitions highlighted several issues that affect modularized systems.

- Starting the system can be a time-consuming process that requires attention to detail since there may be many programs that need to be started, or restarted, with configuration and parameters properly set.
- One module going down during a run can either freeze the robot or cause potentially dangerous situations.
- Recovering from crashes is a time-consuming process.

In the USAR task, in particular, we needed to be able to recover from crashes quickly given the 20 min hard time limit on each competition run.

In addition to the above items, we wanted the robot system to have a list of available capabilities so that it could handle queries from an external module—such as a user interface for remote operation—and start up the proper modules to provide the requested capabilities. We also desired fault tolerance in a situation such as wireless communication failure during remote operation, in which case the robot ought to be aware of the failure and take appropriate steps. In short, we wanted a robot monitor with knowledge of the available modules and their capabilities that could start and stop modules as needed, monitor them while running, restart them if necessary and start up an emergency controller to get out of situations that might otherwise cause failure. To that end, we started developing Robomon in 2003 and used it effectively in the 2004 USAR competition.

Robomon sits on top of the horizontal REAPER structure and does not take part in making decisions about the robot's actions or interact with sensor data. It is responsible for managing the meta-health of the robot software. Robomon is configured using an XML file that specifies the modules it needs to handle. Each module entry in the file specifies a unique name for the module, whether it is *managed*, *external*, or *critical*, the location of the module binary (if appropriate), a timeout in seconds to detect module death, and a maximum number of retry attempts to make to restart the module.

When Robomon first starts up, it locates all the managed, non-external module binaries by accessing the configuration file. All managed modules must have a query startup mode in which they print out the capabilities they possess. Robomon calls each of these modules in the query mode and reads in their output. It then populates the capability tables of each module with the provided information. External managed modules are expected to broadcast a capability message when they initially connect to the central communications hub, which for our implementation is the Inter-Process Communication manager [IPC] (Simmons and James, 2001).

Modules can be divided into distinct categories based on a combination of three criteria.

- *Managed* Modules capable of communicating with Robomon; it can track the status of these modules and may be able to issue a limited set of commands. Most modules are expected to be managed.
- *External* Modules can be stopped and started by Robomon. A module can be managed and external (for example, a remote operation interface) or unmanaged and not external (this may include software not designed for use with message passing system, but still capable of performing some useful task for the robot).
- *Critical* If a critical module fails in the midst of robot operation, Robomon will issue an idle command to all managed modules. This might be useful in a mapping task if the mapping module crashes, but navigation and control are still active. Robomon would idle the other modules, bring the mapping module back up, then continue. Critical modules are expected to reactivate the other modules when they come back up by sending the appropriate messages.

One of the nice features of Robomon is that it gives a remote operation interface the ability to query the robot for a set of capabilities and then let Robomon determine how to provide those capabilities. Therefore, the remote interface needs to know only about capabilities, not the exact details of how to start and stop different software modules. Figure 6 shows the overall software architecture for the USAR task, showing how Robomon fits into the communications structure.

When choosing modules to fulfill a capability request, several factors must be considered. First, the set of modules should span the entire capability request, if possible. Second, it should be the smallest set of modules that provide full coverage. Finally, Robomon must avoid starting up modules that may conflict with one another over control of a resource. For example, initializing two navigation modules on the same robot creates a conflict over which module actually gets to control the robot. The solution used by Robomon for this problem has two steps.

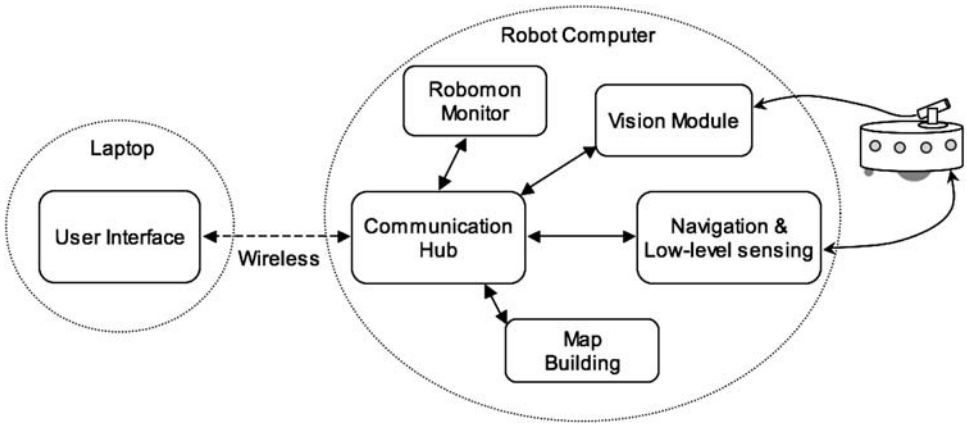

Fig. 6 Diagram of software architecture for the USAR task, including the Robomon monitoring program. Note that part of the system, e.g. the user interface, does not run on the same computer as the rest of the modules

First all capabilities which can be fulfilled by only one module must be located. The modules selected in the first step are critical modules and, once they are chosen, all other modules that conflict with them are removed from the pool of potential modules. The second step is to use an iterative depth first search to choose the smallest group of modules to fill the remaining capabilities. This is not an optimal solution, and for large numbers of modules and capabilities may become untenable. For the number of systems running on most mobile robots it should be sufficient.

An additional special case exists for control modules capable of giving commands that will be executed by the robot. If Robomon is placed into active mode, which must be done explicitly, it will attempt to keep a module with the CON_REAL capability running at all times. The CON_REAL capability means the module is capable of controlling the robot by issuing navigational commands and responding to sensor readings in real time. For example. the State module of Fig. 5 and the User Interface module of Fig. 6 are both modules with the CON_REAL capability.

When Robomon is in the "active" mode, it's job is to keep a CON_REAL module running at all times. If, for example, communication between a remote operator interface and the robot is cut off, Robomon, which is running on the robot, would be responsible for starting up a local control module capable of controlling the robot. The local control module might move the robot to a location where communications could be restored, or it could attempt to continue the task autonomously. In addition to the responsibility of keeping a control module running, if multiple control modules attempt to take control, Robomon has a mechanism to negotiate between them, and choose the higher priority module, as specified in the configuration file.

Just as the REAPER organization grew organically from our experience with integrating multiple capabilities on a single robot, so Robomon grew from our experience at AAAI in managing and developing modular robot systems.

4.4 Module communication

Module communication is a critical piece of the system. The ideal communication method would have low overhead, be fast, properly handle asynchronous message passing, and permit communication using the same methodology between modules regardless of whether they are running on the physical robot. Unfortunately, not all of these constraints can be equally well satisfied.

For the first four years we used shared memory as the message and data passing medium between all of the modules on a robot. In 1998 and 1999 all of the modules were on the same computer, making shared memory a fast, low-overhead means of communication. The major issues were timing and properly using flags.

The control module was the most difficult to integrate because of timing and synchronization issues. In particular, the state machine had to avoid switching between states too quickly. Since the control module did not include any of the low-level sensing or interaction, it iterated extremely quickly and could move between states before the other modules could react to the previous state. Thus, the control module had to watch flags from the other modules to determine when actions were complete before moving on or making another state-transition decision. The strength of this approach was that the state machine could sit back and sample high-level information asynchronously, reacting to changes in the world smoothly and quickly.

When we required communication between robots in 2000, we built a communication module that was responsible for opening a socket over wireless ethernet to a communication module on the other robot and passing data back and forth. This approach required separate mechanisms for inter-robot versus intra-robot communication.

In 2001, when we first implemented remote operation for the USAR event (the 2000 entry was autonomous), we had no mechanism in place for passing large amounts of data-such as

imagery-across the wireless connection to the operator. Our initial implementation, in fact, used X-forwarding through ssh to create a small window with live video on the operator's desktop. While easy to implement, X-forwarding is not an efficient method of passing large amounts of data through a network and the latency is high (several seconds). High video latency in the remote operator control loop either caused instability in robot control or required the operator to slow down and wait periodically to ensure he or she was viewing the current situation.

In 2002, partly in response to a desire to create a uniform communication methodology, and partly because we were involved in the GRACE project with Carnegie Mellon University, we switched the communication system over to IPC, the Inter-Process Communication package (Simmons and James, 2001). Switching to IPC allowed us to more easily handle asynchronous message passing and let each module be built in a similar manner. Because it is built on top of TCP/IP it also made intra- and inter-robot communication look identical, making it easy for any module to communicate directly with a remote host.

Over the course of the 2002 and 2003 competitions we rewrote all of the modules to use IPC. In the first pass, we created individual message types for each module, leading to a proliferation of message types, each with slightly different data requirements. However, seeing the similarity in many of the message types between modules, we decided to pursue what we called the Generic Common Messages [GCM] framework whereby each software module is required to support the same set of messages. The concept work for Robomon was ongoing at the same time, and the GCM enabled Robomon to talk with every module in a uniform manner.

In 2004 and 2005 all of the software modules supported GCM, which provided the following benefits.

- We could plug and play different implementations of the same module.
- In an emergency situation, a single message could Idle or Halt all modules simultaneously.
- When designing new modules, we had a skeleton module to build upon that supported the critical commands (Idle, Quit, Still Alive).

The major software issue we have had with IPC has been working with multiple threads within a single module. Because IPC maintains queues of incoming and outgoing messages it is essential that all IPC calls occur in a single thread. In multi-threaded applications, such as often occurs in user interfaces, this requires a clear understanding of which functions will occur within which threads.

Overall, the move to IPC improved the system software design by providing a structured, asynchronous message passing system that worked for modules both onboard and remote from the robot. It also enabled the Robomon monitoring system as well as a generic message set that all modules support. These characteristics have made it easier to design and integrate new modules into the robot system.

Table 2 Summary of performance results at AAAI 1998–2005

Year	Serving	USAR	Open interaction	Other
1998	1st			Most popular robot
1999	1st			Best Integration
2000	1st	1st		Best Integration
2001	Participation	1st		
2002	3rd	2nd		
2003		2nd		
2004		1st		
2005			Participation	

5 Performance evaluation

As detailed above, the performance of the robots in real-world situations strongly affected our designs in subsequent years. Table 2 shows the record of awards and finishes in each event in which we participated from 1998 to 2005. It is also worth highlighting the best and worst performances and providing some insight into why they occurred.

There were four categories that strongly related to the overall performance of the robots: sensing, hardware, user interfaces, and integration and testing.

5.1 Sensing

Developing robust and reliable sensing mechanisms is one of the most important aspects of robot design. The overall performance of each robot strongly correlated with the robustness of the sensing mechanisms, and inversely correlated with how much the robot relied on complex sensing processes requiring a sequence of decisions. Sensing mechanisms with fast training or calibration procedures also tended to outperform mechanisms requiring parameter selection by hand.

The following are some of the sensing mechanism failures.

- 1998: We developed a version of Fleck and Forsyth's skin-detection algorithm to enable the robot to detect faces and find people (Fleck et al., 1996). We were dismayed to arrive at the convention center to find that the whole room was painted in colors indistinguishable from skin using the algorithm. We ended up using only motion detection,

which was invariant to color, to identify when a person was in front of the robot.
- 1998–99: We designed our speech system to recognize short phrases as part of the interaction. In the qualifying rounds with the judges it worked fine. However, during the main event in a loud room with hundreds of people it could not recognize the phrases with sufficient reliability.
- 2001: We developed a method for name tag detection and reading, which required the robot to find the tag, zoom in on the text, and then execute optical character recognition against a dictionary of names. Each of the pieces worked robustly in isolation, but the system as a whole was too slow unless someone stood very still in front of the robot for about 30s. While we could demonstrate the system to a patient audience, no one was willing to stand still that long during the actual serving event.
- 2005: We tried to have the robot identify people's faces, locate their approximate shirt location, and then comment on the shirt color. When we arrived at the venue, the lighting was sporadic, dark, and differently colored in different locations, causing the colors to change from location to location. Even using a grey calibration card located on the robot to recalibrate the camera once per minute, the process was still sensitive to illumination conditions. We had to reduce the number of colors and call more shirts generic colors like "bright", or "dull" in order to avoid having the robot call someone's white shirt red.

The strengths in sensing were in both vision and sonar/IR navigation tasks.

- 1998–2002: The judges in the snack serving AAAI competitions always wore bright pink ribbons on their badges. Each year we tried to have the robot sense the pink robbons and modify the interaction in some way in response. In 1998 we developed a simple calibration program that used a linear search method to identify appropriate color thresholds for the pink ribbons. The calibration program required a half-dozen training image pairs with and without pink badges in them and then calculated appropriate thresholds to separate the images with a pink badge from the images without one. Because of the simple calibration procedure and the unique color of the ribbons, the pink badge detection was robust, did not tend to generate false positives, and could identify pink badges from several meters away, enabling humorous additions to the interaction.
- 1998–2002: Motion detection was a good way to differentiate plants and walls from people in a serving task. It was also possible to make rough estimates of their distance from the robot by considering the maximum height of the motion in the image because the camera was usually angled upwards (people further away would be lower in the image) (Maxwell et al., 2003).
- 2000: We tried again to develop a face detection system based on color. Upon arriving in the Orlando convention center, we found that the lighting was set up like a checkerboard with alternating reddish and greenish lights. We took over 50 training images of faces under the various conditions and developed a training program to calculate appropriate parameters for the color model, as well as taking into account other face features. The result was robust enough to use in the competition, even given the varying lighting conditions.
- 2000–2002: Italian flag detection–red next to white, next to green-using a state-machine approach worked robustly and let our robots identify one another at a distance (Maxwell et al., 2003).
- 2000: Given the failure of the speech recognition systems in 1998 and 1999, we limited speech recognition on Santino to "yes" and "no" during the Hors d'Oeuvres Anyone? event in 2000. Even in the loud, crowded room, the recognition rate for the limited vocabulary was good enough that Santino was a popular attraction at the event.
- 2004: We implemented velocity space navigation (Fox et al., 1997) for the USAR tasks and were able to significantly improve the speed and safety of the robot over prior implementations using a reactive strategy (Maxwell et al., 2004a).

The 1998, 2000 and 2005 competitions all highlighted the importance of automatic calibration routines. The 1998 face color model, for which we had no such routine, did not end up contributing to the robot's performance in the competition. However, the pink ribbon detector, the face detector in 2000, and the shirt color classifier, all of which did have some type of automatic calibration, or learning procedure associated with them, did contribute positively to the robot's performance. The finding that learning or calibration procedures help performance is not novel in robotics or computer vision, but it does reinforce their importance in real world problems. It should also provide an important reminder to people entering the field, who may opt for the quicker solution of hand-tuning an algorithm rather than the more robust, but often more complex, solution of integrating a framework for calibration or learning into a sensing process.

5.2 Hardware

Hardware failures eventually catch everyone in robotics. That was the case for us in 2001 in the serving task. One of the robots burned out a motor, forcing it to be stationary for the event.

In 2000, we had both robots go down the day before the event: one with a power board failure, the other with a blown

fuse. Fortunately, we were able to repair both reasonably quickly, as one of the robot manufacturers had a booth at the AAAI conference and could provide a new power board for the Magellan. The moral of the story is to have spare parts with you, always.

Other than those two cases, we have been fortunate in having stable hardware, despite straining their weight envelope and batteries. However, except for the 2000 serving event in which Santino (Fig. 4(e)) could raise and lower his tray, we have not significantly modified the original robot bases, reducing the chance of accidents.

The best hardware performance from the robots was in 2004 in the USAR event and 2005 in the Open Interaction event. The new CPU and wireless adapters, combined with upgraded batteries, permitted the robots to move faster and more robustly than in prior events. Sometimes, hardware upgrades are necessary to achieve higher levels of performance. However, efficiency in software design can always make a given set of hardware work better.

5.3 User interfaces

Like sensing mechanisms, we have found that the performance of the robots is inversely proportional to the complexity of the human-robot interface. In quiet environments with users that are purposeful in their interactions with the robots, complex interactions can be successful, humorous, and effective. However, in a situation such as a conference event in a large room with hundreds of people who tend to ignore the robots, complex interactions break down quickly. Some of the less successful interactions are as follows.

- 1999: Alfred's extended interaction process, which was scripted and quite humorous, required patience from the person to get through the 60s interaction. During the qualifying round it worked well. During the conference event people just wanted to grab a snack and move on without otherwise interacting with the robot. Alfred often ended up trying to talk to empty space.
- 2001–2002: For the serving task we dropped the speech recognition and had the robot try to approach people, offer them an hors d'oeuvres, and then move on after pausing for a minute. If you want the robot to serve hors d'oeuvres in a crowded room, it works quite well. But it doesn't engage people (like judges). People also appear to expect some kind of speech recognition from robots, even if it is just yes or no.
- 2001–2002: The USAR interface was a set of command line debugging tools and X-forwarded video. It required 14 windows to operate, and vision commands had to be issued in a separate window from the navigation commands. Needless to say, it was easy to get confused and difficult to get a sense of the robot's situation.

We also had a number of successful human-robot interfaces.

- 2000: Santino had a simple interaction. He asked if the person wanted an hors d'oeuvre, and if they said yes, he raised his tray. If they said no, he moved on. People lined up to interact with the robot, and often went back multiple times if the robot misunderstood a "yes" to be a "no". The immediate physical, and appropriate, reaction of the robot raising the tray was successful as part of a serving interaction. In addition, the simple speech recognition task made it sufficiently robust that the robot did the right thing most of the time. The 2000 serving performance was, by far, the most successful in the loud, large room environment.
- 2001–2002: Removing the speech recognition capability from the robots was both a failure and a strength. It was a strength in that the robots did their job effectively, gave away a lot of hors d'oeuvres and covered a large area. It was a weakness in that people did not engage the robots or interact with them beyond taking an hors d'oeuvres.
- 2004: Implementation of a joystick-based interface for USAR, combined with the CPU and wireless upgrades, made our interface much more robust and responsive. In later testing, 5-year old children were able to effectively and safely drive the robots.

One design issue that appears to be task dependent is whether to use a recorded voice or a synthesized voice for the robot. In 1998, 1999, and 2005 we used recorded voices in the serving and interaction tasks, while from 2000–2002 we used synthetic voices. As good as the synthesized voices have become, the recorded voices are still easier to understand and project much more emotion and personality. The tradeoff, of course, is that a robot using recorded voices cannot dynamically change what it says in response to inputs. In the 2000–2002 years we felt that the benefits of flexibility offset the reduced comprehensibility and personality. In 2005, however, while the synthetic voices had improved and flexibility would have been nice, we decided that comprehensibility and emotion in the phrases was the more important factor. We made that decision after testing the robot using synthetic speech with people from outside the lab. The lack of emotion in the synthetic speech made it difficult for the robot to convey its emotional state, which was the whole purpose of that particular system.

5.4 Integration and testing

In robotics, there is no substitute for time spent on integration and testing. Not surprisingly, given our performances in

Table 2, we spent the most time on integration and testing in 1998, 1999, and 2004. In 1998 we planned for three weeks (out of ten) to integrate the different modules together and executed a dry run at an event at the local art museum a week before AAAI. That revealed a number of strengths and weaknesses in the robot that helped us to focus our efforts for the competition.

In 1999 we also executed a dry run with a group of people recruited from around Swarthmore College two weeks prior to AAAI. Seeing the interaction between the robot and people not familiar with it led to a number of changes in the script and the overall method the robot used to engage people. We also made modifications to speed up the motion detection algorithm because of a delay caused by that sensing mechanism at one point in the interaction.

An anecdote on Alfred in 1999 provides a specific example of how the time spent on integration impacted our performance. During the judging round, Alfred was successfully serving and interacting with the judges, who had managed to give predictable responses for the speech recognition system so far. Then, in response to the robot asking the judge whether he wanted an hors d'oeuvre, the judge said "Yes, thank you" To provide some context, Alfred was set up to respond with a rude comment if the person didn't include a "please" in their response to the question. Those of us not directly involved with the speech recognition module held our breath, thinking that the robot would now insult the judge, who was obviously trying to be polite and had just substituted "thank you" for "please". But Alfred came back with a flattering comment for the judge and the student in charge of the speech system grinned at the rest of us and said "I handled that". A performance like that does not happen unless you have enough time to test out the system with real people so that rare or unlikely events occur during testing and can subsequently be handled correctly.

In 2004 we had the benefit of having the same people working on the robot for a second year. The 2003 USAR event served as an effective evaluation of the system, and our participation in a user study on USAR interfaces helped us to focus on how to improve it (Yanco et al., 2004). However, the 2004 system ended up being a complete rewrite of most of the software modules and the user interface, so it still required extensive testing. Since the developers had a year of experience with the system, they were able to have a system in place and ready for testing several weeks prior to the AAAI event.

In short, our performance in 2000 is actually the anomaly in that we did much of the final integration and testing at the AAAI competition itself. We also had ten people on the team that year, several of whom were veterans of the 1999 robot team. Those factors, along with some luck, overcame the lack of sufficient integration and testing time prior to the event. However, there were some uncomfortable times just before the competition when it was not clear if the system would work, and most of us got very little sleep.

6 Lessons for real world robotics

Taking our robots out of the lab and having them interact with real people in real situations taught us a number of lessons about robot design. Many of them are well-known, universal design principles, but some are particular to the design of robots that will interact with people.

Summarizing some of the findings above:

- Your lab is not the real world. Support your algorithms, particularly sensing algorithms, with a fast calibration or training framework so that your system can adapt to new locations.
- Complex sensing processes are difficult to make sufficiently robust for effective use by a robot in real-time situations. They need to be extensively tested in realistic situations prior to installation on the robot. Interactions with people can occur quickly, so sensing processes that take a long time-such as name tag reading-should be used only when the interaction is guaranteed to be long enough for the sensing process to complete.
- Complex human-robot interactions are only effective when people want to engage the robots and the conditions permit effective sensing. Interactions that involve the recognition of complex phrases or a large variety of words must be extensively tested and incorporate processes for graceful failure. Keep interactions simple unless the situation requires a more complex script and both parties have a copy.
- Time spent on integration and testing is rewarded by performance. You make your own luck.
- Testing with novice audiences—not the robot developers—is critical. First, the main mode of interaction may require knowledge that novice audiences do not have but that is assumed by the robot developers. That disconnect can cause the interaction to break down because the novice audience does not respond in the expected manner. Second, only by testing with novice audiences are uncommon events likely to occur during the interaction; the robot developers know what the robot expects and generally follow those expectations during testing to enable the robot to perform correctly.
- Hardware and software robustness can only be evaluated under realistic conditions. Put all the accessories on the robot and turn them on well before the demonstration to discover their effect on the system as a whole.

The trajectory of changes to our robots reflect a number of these lessons. The serving interface, for example, got simpler from 1998 through 2002. The 2002 serving robot was perhaps the most effective from an objective standpoint (snacks

served and area covered), but did not do well in the judging because of the very simple interface. Likewise, the USAR remote operation interface evolved from a very complex one in 2001–2002 to a very simple joystick-based game-like interface in 2004 (Maxwell et al., 2004b). That change was reflected in our performance in 2004, which was objectively and subjectively our best performance in that task.

The major lesson we have learned, however, is that people do not treat your robot the way you do (or the way you expect them to) unless the interface is incredibly obvious and predictable. The following are a few anecdotes.

- An individual walked up to Alfred (Fig. 4(b)) grabbed the microphone and tried to turn on the robot by using it as a switch.
- Several children hugged Alfred.
- People commonly talk to the robot, even if it has no obvious microphone or speech recognition capability.
- People commonly talk to robots as though they could pass a Turing test.
- People kick the bump sensors.
- People like to step in front of the robot to see what happens.
- Kids like to chase robots or trap them in a corner.
- Sometimes people completely ignore a robot.
- Sometimes people crowd around a robot so tightly the robot can't safely move.

In all of these situations, the robot does not have to be perfect. It just has to do about the right thing most of the time, and avoid doing the absolutely wrong thing all of the time. Doing very little for short periods of time is often the right thing for a robot to do.

In summary, simplicity and robustness are the two most important traits when building robot systems that interact with real people. Those two characteristics, which are often related, make the robots predictable, safe, and less susceptible to doing the wrong thing when people and strange environments challenge the assumptions of the robot's designers.

Acknowledgments This work was partially funded by NSF grants IIS-0308186, IIS-0415224, CDA 0098802, and CDA 9619805. Swarthmore College also provided support for summer students to work on the robot projects. The American Association for Artificial Intelligence provided travel funding for t



Bruce A. Maxwell received B.S. and B.A. degrees from Swarthmore College, an M. Phil. from Cambridge University, and a Ph.D. in Robotics from Carnegie Mellon University. He is currently an Associate Professor of Engineering at Swarthmore College in Pennsylvania. In addition to bringing undergraduate teams to participate in the AAAI Robot Competitions from 1998–2005, he was general co-chair for the competition in 2003, and supervised the Open Interaction Event in 2004. Dr. Maxwell's interests include human-robot interaction, social robots, computer vision, computer graphics, and visualization and data analysis.