A HMI Supporting Adjustable Autonomy of Rescue Robots

Andreas Birk and Max Pfingsthorn

School of Engineering and Science
International University Bremen
Campus Ring 1, D-28759 Bremen, Germany
a.birk@iu-bremen.de

Abstract. Human rescue workers are a scarce resource at disaster sites. But it is still a long way to go before fully autonomous rescue robots will be fieldable. The usefulness of rescue robots will hence strongly depend on the availability of user interfaces that enable a single first responder to operate a whole set of robots. For this challenge, it is important to preprocess and streamline the immense data flow from the robots and to assist the operator as much as possible in the processes of controlling the robots. This paper introduces an adaptive graphical interface supporting adjustable autonomy of rescue robots. The design is based on insights from the literature in this field where intensive surveys of the actual needs in this domain were compiled.

1 Introduction

Rescue robots have a large application potential as demonstrated for the first time on a larger scale in the efforts at the World Trade Center after the 9/11 event [Sny01]. For an overview of potential tasks of rescue robots and the related research in general see for example [RMH01]. One of the main challenges in using robots in search and rescue missions is to find a good trade-off between completely remotely operated devices and full autonomy. The complexity of search and rescue operations makes it difficult if not impossible to use fully autonomous devices. On the other hand, the amount of data and the drawbacks of limited communication possibilities make it undesirable if not unfeasible to put the full control of the robot into the hands of a human operator.

The goal of the IUB rescue robots team is to develop fieldable systems within the next years. Since the beginning of its research activities in this field in 2001, the team has participated in several RoboCup competitions to test its approaches [Bir05, BCK04, BKR+02]. In addition to work on mapping [CB05] and adhoc-networking [RB05], the development of the robots themselves based on the so-called CubeSystem [Bir04] is an area of research in the team [BKP03, BK03].

As mentioned, one of the the main challenges for using rescue robots is to find a good tradeoff between completely remotely operated devices and full autonomy [BK03]. Ideally, a single rescue worker supervises a multitude of semi-autonomous robots that provide only the most crucial data to an operators

A. Bredenfeld et al. (Eds.): RoboCup 2005, LNAI 4020, pp. 255–266, 2006.

station. Here, we present an approach to this challenge in form of an adaptive graphical human machine interface (HMI) supporting adjustable autonomy.

Adjustable Autonomy (AA) in general addresses the issue that, while autonomous agents get more and more powerful, it is still impossible for a real application to exclude the human operator from the loop of operations and retain the same effectiveness. Using Adjustable Autonomy, an agent can behave autonomously and dynamically change its level of independence, intelligence and controlfreely placeable. The concept of Adjustable Autonomy is very broad and does not only encompass robotics. It has been applied to many fields, such as office automation [SPT03], desktop software [Mim98], military [FCGB00], and space exploration [SBA+03].

A way to achieve Adjustable Autonomy is to support multiple levels of autonomy of the agent, e.g. fully autonomous, guided by the operator, and fully controlled by the operator [BDM02, GJCP01]. It is also conceivable to support a continuous range of autonomy levels, e.g. by employing a continuous measure of neglect and problem priorities [OG03].

For example, an ideal situation in Adjustable Autonomy would be the following: A rescuer at a disaster site employs the help of a team of robots to survey a part of a collapsed building. To be more effective, the rescuer commands the robots to spread out and to explore, detecting and locating victims, but also fire, etc. While the robots search the building, they may run into problems, like getting stuck on stairs, not knowing how to avoid an obstacle best, etc. All those problems are forwarded to the rescuer. That is the only time when the rescuer really has to take care of individual robots. Once a robot has found an interesting set of features indicating an object or event, the notice is also sent to the rescuer and marked in a collaborative map. All sensor data of the respective robot is made available to the rescuer to verify the findings of the robot. Once victims are found and a safe path to them has been identified, rescue crews are deployed to extract the victims from the building.

Fig. 1. The IUB rescue robots at RoboCup

Previous approaches to a Graphical User Interface (GUI) for Adjustable Autonomy encompass a wide variety of settings and intentions. Most interesting for our interests is the investigation of designs by Goodrich and Olsen [GJCP01], Fong et al [FCTB01, FTB01, FCGB00, FTB02, FT01], Backes et al [BNP$^+$] and Bruemmer et al [BDM02]. In addition, evaluations in terms of usability are highly important as for example done by Olsen and Goodrich [OG03] and Scholtz et al [SYDY04].

The approaches taken in the aforementioned papers roughly fall into two classes. In the first one, a focus is made on direct control and portability of hardware. In the second, task planning and data visualization is emphasized. A general trend in this line of research is visible: Most rely on one main mode of direct visualization of the robot's state, either through a map generated by the robot or a direct video stream. This coarse form of data visualization is especially useful for gaining a quick understanding of the robot's situation, referred to as *Context Acquisition* [OG03]. In addition, the presented interfaces offer multiple ways of commanding the robot: through haptic and gesture manipulation [FCGB00], direct vector input and choosing a location on a map [FCTB01], and pure high-level task planning [BNP$^+$]. This process is referred to as *Expression* [OG03].

In an attempt to combine the best of both, i.e., direct interaction with the robot and data visualization, we attempt to merge both paradigms: to employ intuitive data visualization and direct intuitive control methods in order to optimize *Context Acquisition* and *Expression* time, exactly those deemed most costly by Olsen and Goodrich [OG03].

2 The Design Goals and Core Components

The main application target for the system are real-time, multi-robot tasks. While the developed interface framework will allow for various uses, main design arguments will be geared to the application in real-time environments. This includes a bigger data visualization area and rather small control areas, since most control is done via peripheral input devices.

Murphy and Rogers [MR96] pointed out inefficiencies in traditional teleoperation interfaces that are supposed to be corrected with this new interface framework. Those deficiencies include constant high-bandwidth demands, cognitive fatigue due to too many tasks done at once, and poor displays. It is mentioned that conservative teleoperation interfaces are inefficient because the operator usually only handles one robot and that reduces work efficiency by a factor of five to eight. Further important design guidelines were derived from the work of Olsen and Goodrich [OG03] and of Scholtz et al [SYDY04], which is based on an evaluation of existing human-robot interfaces.

Olsen and Goodrich hypothesize that the process of human-robot interaction can be broken down into four parts: *Subtask Selection* (picking a robot to service from a list), *Context Acquisition* (perceiving the robot's current state and problem), *Planning* (thinking of a way to solve the robot's problem), and *Expression* (stating the plan in robot oriented commands). Scholtz et al state from observations made during the World Championship Competition in

RoboCup Robot Rescue 2004 in Padova, Italy, that successful interfaces should include a representation for the global environment, display of the robot's current state, integrated display of sensor data, ability for self-diagnosis of the robot, and contex-sensitive information display.

Based on these guidelines, we developed an interface that is divided into three blocks, the Canvas, the Sidebar, and the Visibility Controller. Each core component is described in detail in the following.

2.1 The Canvas

The Canvas is a drawing space for 3D representations of the world, using OpenGL. Here, usually a map would be drawn and populated with robots. Due to the extreme flexibility of the design, almost any kind of visualization is possible here. Even 2D displays, like status bars for all connected robots or video displays are possible.

In the context of the above mentioned criteria, the Canvas represents a common ground for sensor data visualization. Therefore, a facility has been established to support easy fusing of sensor data by the interface itself. For example, both the map and the display of the robot and its state can be separate modules both drawing themselves to this canvas. As a result, the data is merged automatically, making it easier for the user to perceive the robot's current situation.

In addition, a camera implemented by the Canvas can be moved around in an intuitive fashion (very much like a First Person game interface) to allow viewing the depicted 3D environment from all angles. In addition, the camera can be "snapped" to the current robot's position and orientation (pose) such that it follows the movements of the robot. This gives rise to a separate viewing method, known from racing games which should especially come in handy when purely teleoperating the robot.

The Canvas enhances *Context Acquisition* as well as *Subtask Selection*, as all robots are shown and their state can be evaluated at the same time. In Scholtz's terms, the Canvas addresses the first three points.

2.2 The Sidebar

The Sidebar constitutes a place to display control elements. As space is limited on the screen, this part is kept to a minimum of size. This is because in general, control is exerted via peripheral input devices, such as mouse, joystick or joypad, and steering wheels. Only displaying their state is useful for even more direct user feedback, as mentioned above.

The Sidebar also allows for the grouping of control elements. This gives a clearer structure to the controlling side of the interface and thus decreases the cognitive load necessary for the *Expression* process, as mentioned above.

In addition, the Sidebar has a built-in collapse and expand function. Each group can be collapsed (hidden) or expanded (shown) by the request of the user. This way, the user can choose which kind of control is supposed to be used. This feature covers Scholtz's last point.

2.3 The Visibility Controller

The last and least visible part is the Visibility Controller. Almost completely hidden to the user, other than a simple drop-down list to choose a visibility setting, the Visibility Controller takes care of scaling, hiding and positioning single elements that are displayed. It can handle both control and display elements.

The Visibility Controller is designed to store a vector of importances of interface parts. When a certain mode of operation is assumed (compare [GJCP01, BDM02]), the Visibility Manager will set the importances of all elements. The elements then know how to adjust their own behavior, form, and shape to reflect that importance value.

This concept of a Visibility Controller clearly addresses Scholtz's last point and Olsen and Goodrich's concepts of *Context Acquisition* and *Expression*, since it requires less time to choose an appropriate control from an already smaller list.

2.4 The Dynamics Aspects

As briefly described in the previous section, the interface presented also consists of a so called Visibility Controller. This module manages importances of single elements shown on the screen. According to these importances, the single modules representing drawable items perform different actions.

For example, a control element can decide to hide itself and to stop producing output if its importance falls beneath a certain threshold level, while a display element might implement a form of scaling (e.g. for video displays) or fading (e.g. for less importance of the map). Also rearrangement is possible, for example in the case of video displays. As the importance of a video display increases, it can change the arrangement of multiple video streams it is displaying, e.g. shifting from a stack structure to a grid in order to distribute the available space in a better way.

As Goodrich and Olsen [GJCP01] and Fong et al [FCTB01] point out in their treatises, choosing different modes of autonomy is a highly desirable feature. Especially, not only does the robot behavior change, but the interface should reflect this change as well. This gives further motivation for the Visibility Controller as it further addresses the problem of *Context Acquisition*, i.e., the mode of operation can be inferred through the arrangement of the display [OG03] and specifying context-sensitive information [SYDY04].

While it is reasonable to assume that such importances are changed dynamically and automatically by choosing a mode of operation, this fact is quite limiting for the user. During the *Planning* phase, optimal use has to be made of the displayed information in order to create a plan how to circumvent the robot's current problem. This might require reading displays with a lower importance. Hence, it is important to leave a chance of changing importances of single displays and controls to the user both during missions planning as well as on the fly.

For the interface and design presented, it does not make a difference how the changing of importances is implemented. However, the tools used to implement the whole interface already provide adequate ways of doing so. For example, the

OpenGL standard [SB] defines four component colors, including an Alpha component, which would make it easy to "fade" a display on the Canvas according to some function of the importance. Also, OpenGL defines scaling functions in all three directions. The Qt Toolkit [Tro], used to develop the standard GUI components of the interface, already allows for dynamic hiding and resizing of components.

3 The Framework Structure and Its Implementation

3.1 Qt Toolkit

The Qt Toolkit by Trolltech [Tro] is a very easy-to-use, cross-platform GUI toolkit with some more support for cross-platform development, like threading, file handling and the like. Qt employs their own model of event-driven programming, namely "signals" and "slots". Those are dynamically callable interfaces to events such as "clicked" for buttons and "activated" for combo boxes.

Qt abstracts windowing operations very nicely and it is as easy to construct an application out of ready made components as to implement a very special-purpose "widget", i.e. a custom-made component of the GUI. Also, Qt offers a straight-forward interface to mouse events and even to OpenGL, so the Qt Toolkit was the first choice as the underlying windowing library for this project.

3.2 FAST Robots

As the interface has to rely on an infrastructure to be able to communicate with controlled robots, the HMI is based on the FAST Robots framework (Framework Architecture for Self-Controlled and Teleoperated Robots) [KCP+03]. This middleware used for high-level sensor abstraction and communications employs a strong generalization for sending any type of serializable data over a network. Initially, the framework was designed for remote data collection for virtual experiments, but progressed to being a very extensible architecture for rapid prototyping in robotics.

In Figure 2 on the left, the basic structure of the architecture can be seen. Communications flow starts at the level of the sensor as it is queried in regular time intervals by the owning "experiment". The experiment reads the data and sends it on via its network link. On the other side of the network link, the corresponding "monitor" reads the information, and due to a stored ID number, forwards the data to the right "display". In the case of control data flow, the situation is reversed: The "control" is given data by the user and, due to the asynchronous nature of Graphical User Interfaces, queries the monitor to forward its data via its network link. The corresponding experiment receives the data and, employing a similar distribution scheme like the monitor did for sensor data, forwards the control data to the right "actuator".

The right side of figure 2 shows a depiction of the relationship between a sensor and a display implementation. Such a pairing shares another class: A NetworkData subclass implementing the specific way data from this sensor is

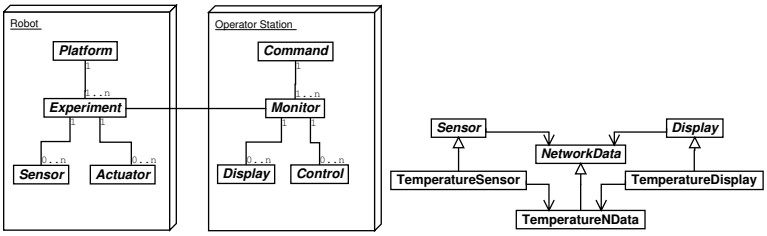

Fig. 2. The structure of FAST Robots (left) and the Sensor/Display/NetworkData triple (right)

serialized and deserialized to a form suitable for transportation over the network. It is conceivable to implement different NetworkData subclasses for a single type of sensor/display pair as different compression algorithms might be used, e.g. JPG, PNG or even Wavelet compression for pictures.

While the design of FAST Robots is very general and platform-independent, the current implementation heavily uses the Qt Toolkit mentioned above for various subtasks, such as threading, concurrency, data serialization and the GUI of the "Command". Lately, efforts have been made to port FAST Robots to other architectures and libraries.

3.3 Interface Framework

The interface presented focuses on the Controller side of the schematic shown in Figure 2 and will thus be called "FAST Commander". In simple words, a new front end for FAST Robots applications was to be developed. Hence, the new interface should be as flexible and as extensible as possible in order to be as versatile as FAST Robots.

The UML diagram shown in Figure 3 shows the structure of the design. There are four main components:

- The RobotList, which manages all Robots connected and ensures forwarding of data. Also, the current robot is chosen here and data coming from the current robot is forwarded through a special channel. If a certain type of data from a robot is important it is also forwarded while the robot is not the current robot. In addition, specific types of data can be muted, i.e. they do not get forwarded at all. Most importantly, the Robot maps robot-specific sensor IDs to application wide keys, e.g. from '1' to "video0" meaning that sensor 1 on this robot is the first video sensor.
- The GLViewCanvas, which presents all GLCanvasItems to the user. Those items are the main means of displaying sensor and other (e.g. mapping) data. The GLViewCanvas implements general necessities, such as a freely movable camera to change perspective of the display and processing mouse events, which are forwarded to the associated GLCanvasItems. In addition, the canvas also implements a sort of z-index for each GLCanvasItem, which

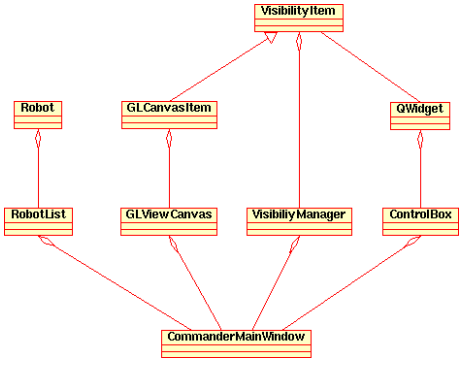

Fig. 3. The UML diagram of the framework provided

is used to sort the associated items. While mouse events are forwarded in reverse z-order (top-most item first), the drawing is done in z-order (bottom-most item first). Since sorting is only done during insertion, there is no additional cost and a more natural feel is achieved for the mouse interaction.
- The ControlBox (previously called Sidebar), which holds QWidgets (the most general GUI component class from the Qt Toolkit), is a container for control components. Control components are grouped and may be hidden (collapsed) or shown (expanded) as a group.
- The VisibilityManager, which implements dynamic aspects of the interface, as described in section 2.4. Implementation wise, the Visibility Manager does not only know what to do with VisibilityItem subclasses (i.e. how to set their importance) but also what to do with QWidgets (e.g. hide them if the importance drops beneath a certain threshold) and group names from the ControlBox (same as for QWidgets).

All these components are always present in any instantiation of the FAST Commander interface. Any additional components, e.g. a map canvas item, a joystick control, a connection to a specific robot and its mappings, are added at mission specification. This ensures high customizability and task independence.

3.4 The Basic Displays

Figure 4 shows a screenshot of the core displays:

- A Map Display, which draws a 3D map according to the robot's sensor input. Cells believed to be empty are drawn flat and in Green color, whereas cells believed to be occupied are drawn in red with a height proportional to the strength of the belief. Importance is implemented as a proportional fading.
- A Robot Display, which draws the current state and pose of a single robot in a sensible position relative to the map. The robot may assume a 3D pose and the display can also indicate if the robot is active. Additionally, two gauges exist that can indicate an overall health status.

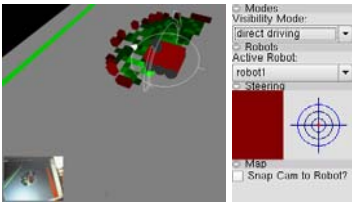

Fig. 4. Core displays shown in the GLViewCanvas (left) and controls grouped on an example sidebar (right)

– A Video Display, which draws video streams as a stack of video pictures starting from the lower left corner. Their size can vary dynamically and scaling is applied for both resizing of the GLViewCanvas and reflecting the importance level.

3.5 Core Controls

The first control is the Joystick control. Apart from standard controls such as a component allowing to choose a robot for activation, changing the visibility arrangement, and controlling the GLViewCanvas, specific controls are implemented that allow further access to the display's features, e.g., controlling the map. The following controls are present in the interface (see Figure 4 for a picture of an example sidebar):

– The VisibilityControl, which lets the user choose one of some preset arrangement of importances geared toward a specific mode of operation. In this example case, choices included "map-based driving" and "direct driving".
– The RobotListControl, which offers a choice of all connected robots to change focus to. Once a different robot is selected, all control messages are sent to that robot.
– The JoystickControl, which gives direct feedback of the read joystick position and button states. The red line on the bullseye on the right indicates the current displacement of the joystick. The red bar on the left is divided into segments representing all buttons on the joystick. When a button is pressed, the corresponding segment turns green.

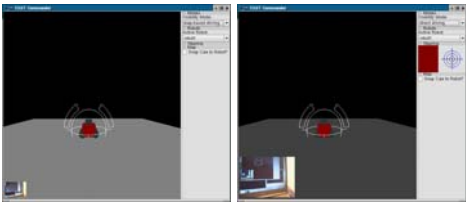

Fig. 5. An example of the Visibility Manager, where attention is directed by scaling a video stream up

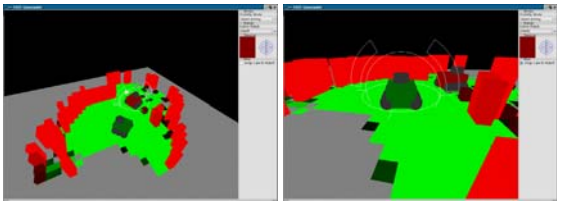

Fig. 6. An example of the GLViewCanvas and the RobotList, where it is possible to either have a free moving camera (left) to get an overview or to snap the camera view to a robot (right)

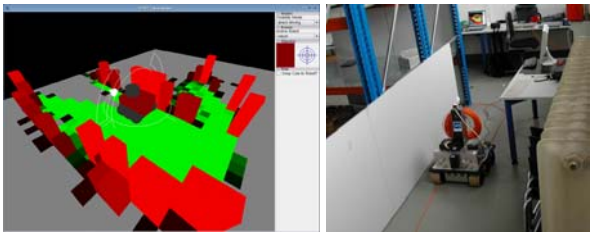

Fig. 7. A test run with Papa Goose (Top: The control interface, Bottom: The robot in the lab)

– The CanvasControl, which gives the single option of snapping the camera of the canvas to the poses of the current robot. This option gives rise to the earlier mentioned "race car"-like driving experience.

4 Conclusion

An adaptive human machine interface (HMI) for rescue robots was presented. The HMI supports adjustable autonomy by automatically changing its display and control functions based on relevance measures, the current situation the robots encounter, and user preferences. The according parameters and rules can be specified during mission planning before the actual run as well as on the fly. The design follows general guidelines from the literature, based on intensive surveys of existing similar systems as well as evaluations of approaches in the particular domain of rescue robots.

References

[BCK04] Andreas Birk, Stefano Carpin, and Holger Kenn. The IUB 2003 rescue robot team. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Artificial Intelligence (LNAI)*. Springer, 2004.

[BDM02] David J. Bruemmer, Donald D. Dudenhoeffer, and Julie L. Marble. Dynamic-autonomy for urban search and rescue. In *Proceedings of the 2002 AAAI Mobile Robot Workshop*, Edmonton, Canada, 2002.

[Bir04] Andreas Birk. Fast robot prototyping with the CubeSystem. In *Proceedings of the International Conference on Robotics and Automation, ICRA'2004*. IEEE Press, 2004.

[Bir05] Andreas Birk. The IUB 2004 rescue robot team. In Daniele Nardi, Martin Riedmiller, and Claude Sammut, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence (LNAI)*. Springer, 2005.

[BK03] Andreas Birk and Holger Kenn. A control architecture for a rescue robot ensuring safe semi-autonomous operation. In Gal Kaminka, Pedro U. Lima, and Raul Rojas, editors, *RoboCup-02: Robot Soccer World Cup VI*, volume 2752 of *LNAI*, pages 254–262. Springer, 2003.

[BKP03] Andreas Birk, Holger Kenn, and Max Pfingsthorn. The iub rescue robots: From webcams to lifesavers. In *1st International Workshop on Advances in Service Robotics (ASER'03)*. 2003.

[BKR+02] Andreas Birk, Holger Kenn, Martijn Rooker, Agrawal Akhil, Balan Horia Vlad, Burger Nina, Burger-Scheidlin Christoph, Devanathan Vinod, Erhan Dumitru, Hepes Ioan, Jain Aakash, Jain Premvir, Liebald Benjamin, Luksys Gediminas, Marisano James, Pfeil Andreas, Pfingsthorn Max, Sojakova Kristina, Suwanketnikom Jormquan, and Wucherpfennig Julian. The IUB 2002 rescue robot team. In Gal Kaminka, Pedro U. Lima, and Raul Rojas, editors, *RoboCup-02: Robot Soccer World Cup VI*, LNAI. Springer, 2002.

[BNP+] Paul G. Backes, Jeffrey S. Norris, Mark W. Powell, Marsette A. Vona, Robert Steinke, and Justin Wick. The science activity planner for the mars exploration rover mission: Fido field test results. In *Proceedings of the 2003 IEEE Aerospace Conference*, Big Sky, MT, USA.

[CB05] Stefano Carpin and Andreas Birk. Stochastic map merging in rescue environments. In Daniele Nardi, Martin Riedmiller, and Claude Sammut, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence (LNAI)*, page p.483ff. Springer, 2005.

[FCGB00] Terrence Fong, Francois Conti, Sebastien Grange, and Charles Baur. Novel interfaces for remote driving: gesture, haptic and pda. In *SPIE Telemanipulator and Telepresence Technologies VII*, Boston, MA, November 2000.

[FCTB01] Terrence Fong, Nathalie Cabrol, Charles Thorpe, and Charles Baur. A personal user interface for collaborative human-robot exploration. In *6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*, Montreal, Canada, June 2001.

[FT01] T. Fong and C. Thorpe. Vehicle teleoperation interfaces. *Autonomous Robots*, 11(1), July 2001.

[FTB01] Terrence Fong, Charles Thorpe, and Charles Baur. Advanced interfaces for vehicle teleoperation: Collaborative control, sensor fusion displays, and remote driving tools. *Autonomous Robots*, 11(1), July 2001.

[FTB02] Terrence Fong, Charles Thorpe, and Charles Baur. Robot as partner: Vehicle teleoperation with collaborative control. In *Multi-Robot Systems: From Swarms to Intelligent Automata*. Kluwer, 2002.

[GJCP01] Michael A. Goodrich, Dan R. Olsen Jr., Jacob W. Crandall, and Thomas J. Palmer. Experiments in adjustable autonomy. In *Autonomy, Delegation, and Control: Interacting with Autonomous Agents*. IJCAI workshop, 2001.

[KCP+03] Holger Kenn, Stefano Carpin, Max Pfingsthorn, Benjamin Liebald, Ioan Hepes, Catalin Ciocov, and Andreas Birk. Fast-robotics: a rapid-prototyping framework for intelligent mobile robotics. In *Proceedings of the 2003 IASTED International Conference on Artificial Intelligence and Applications*, pages 76–81, Benalmadena, Malaga, Spain, 2003.

[Mim98] Yoshiaki Mima. Bali: A live desktop for mobile agents. In *Proceedings of the 1998 IEEE Third Asian Pacific Computer and Human Interaction*, Kangawa, Japan, July 1998.

[MR96] R. Murphy and E. Rogers. Cooperative assistance for remote robot supervision, 1996.

[OG03] Dan R. Olsen and Michael A. Goodrich. Metrics for evaluating human-robot interactions. In *Proceedings of PERMIS 2003*, September 2003.

[RB05] Martijn Rooker and Andreas Birk. Combining exploration and ad-hoc networking in robocup rescue. In Daniele Nardi, Martin Riedmiller, and Claude Sammut, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages pp.236–246. Springer, 2005.

[RMH01] M. Micire R. Murphy, J. Casper and J. Hyams. Potential tasks and research issues for mobile robots in robocup rescue. In Tucker Balch Peter Stone and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, Lecture keywordss in Artificial Intelligence 2019. Springer Verlag, 2001.

[SB] SGI and Architectural Review Board. The OpenGL Standard. Available at: http://www.opengl.org/.

[SBA+03] Maarten Sierhuis, Jeffrey M. Bradshaw, Alessandro Acquisti, Ron van Hoof, Renia Jeffers, and Andrzej Uszok. Human-agent teamwork and adjustable autonomy in practice. In *Proceedings of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2003, NARA*, Japan, May 2003.

[Sny01] Rosalyn Graham Snyder. Robots assist in search and rescue efforts at wtc. *IEEE Robotics and Automation Magazine*, 8(4):26–28, December 2001.

[SPT03] Paul Scerri, David V. Pynadath, and Milind Tambe. Towards adjustable autonomy for the real world. *Journal of AArtificialIntelligence Research*, 17:171–228, 2003.

[SYDY04] Jean Scholtz, Jeff Young, Jill L. Drury, and Holly A. Yanco. Evaluation of human-robot interaction awareness in search and rescue. In *Proceedings of the International Conference on Robotics and Automation, ICRA'2004*. IEEE Press, 2004.

[Tro] Trolltech. The Qt Graphical User Interface Toolkit. Available at: http://www.trolltech.com/.