A Human-Robot Interface for Urban Search and Rescue

Bruce A. Maxwell, Nicolas Ward, and Frederick Heckel
Swarthmore College
Swarthmore, PA 19081
{maxwell, nward2, fheckel1}@swarthmore.edu

Abstract

This paper describes the Swarthmore College entry in the 2003 Urban Search and Rescue [USR] Event at the 2003 American Association for Artificial Intelligence Robot Competition. The primary focus of Swarthmore's entry was the development of a fast, responsive user interface for effectively tele-operating a robot with some autonomous capability. This interface was inspired by first-person video game interfaces, and enabled Swarthmore to place second in the 2003 USR competition.

Introduction

The Urban Search and Rescue [USR] event at the American Association for Artificial Intelligence [AAAI] Robot Competition challenges robot teams to find victims in a simulated urban environment. In order to be successful at this task, the robot must answer three questions:

1. Where should it search?
2. How should it get there?
3. How does it identify victims?

The most successful teams so far answer these questions by providing a human operator with as much information as possible about the situation so that they can answer the questions accurately. Currently, the second question--regarding local navigation--is the only one of these three questions for which teams have attempted to give the robots autonomy.

It is our belief that the eventual solution to this problem will be a combination of effective user interfaces and robust autonomy on the part of the robot in answering the questions above. In other words, it is important to have a human in the loop, but the human should be processing and answering the most difficult questions encountered in the search, not providing input about basic search patterns or navigation except in difficult circumstances.

Since robust autonomous systems are still a difficult research problem, effective user interfaces are critical. Thus, the focus of our 2003 USR entry was to develop an effective, responsive user interface that permitted the operator to use a range of autonomy in navigation, and provide the greatest flexibility in the use of the robot's hardware. As a result of the effective interface and the robot's limited navigational autonomy, Swarthmore placed second in the 2003 competition.

Figure 1: One of the Magellans entering the USR arena.

Robot Hardware and Software

Swarthmore's robots in the USR competition were two iRobot/RWI Magellan Pro's with onboard 450MHz Pentium III computers running Linux. Figure 1 shows one of the robots entering the USR arena. The Magellans come with 16 sensor panels, each containing a bump sensor, an IR sensor, and a sonar. In addition, each Magellan has a Canon VC-C4 pan-tilt-zoom [PTZ] camera mounted on top, near the front, with a pan range of -100˚ to 100˚, a tilt range of -30˚ to 90˚, and a 16X optical zoom. The camera video is connected to a bt848-based PCI framegrabber, and it receives pan-tilt-zoom commands via the serial port.

Because the robots are wheeled, and do not have a high clearance, they are only able to enter the yellow area of the arena. So the focus of the interface is on making the operator more productive, providing a large quantity of accurate information about the environment, and enabling the robot to traverse the area faster.

The software used locally on the robot to manage the robot's navigation and sensors in the 2003 contest was based on the REAPER software originally developed at Swarthmore for the AAAI contest in 2000 (Maxwell *et. al.* 2001). Since then, the majors changes have involved upgrades to the vision system (Maxwell *et. al.* 2003), and the use of the Inter-Process Communication [IPC] package for all communication between processes (Simmons and James, 2001).

Human-Robot Interface

The primary focus of our work for the 2003 contest was the interface for managing the robot during the Urban Search and Rescue [USR] task. The interface is a set of non-local (to the robot) applications that permit an operator to manage one or two robots simultaneously. Communication with the processes local to the robot occurs via IPC. The interface implements the concept of sliding autonomy, where the operator is able to give the robots more or less decision-making ability depending upon the situation. This interface is described in detail in the following section.

Motivation

Early in the course of this design project, it was realized that the greatest advances in user interfaces [UI] in the past ten years have been made in the realm of video games. Computer and console games alike must have clear interfaces which provide the player with both controls and informational displays that allow for very fast response times to situations in a dynamic environment. These interfaces have developed at an incredibly rapid pace, with perhaps the pinnacle of UI design being that of so-called First-Person Shooter [FPS] games (Unreal Tournament, 1999). FPS games are the most appropriate interface paradigm for a project such as USR, as they represent a three dimensional world with which the player can interact. Navigating a three-dimensional game world in search of computer-generated enemies is analogous to the task of finding injured and/or trapped victims in a USR scenario. Mapping that three-dimensional environment down to a two-dimensional computer display is simple enough using a live video feed, but making it intelligible, informative, and useful for the robot operator is far more difficult.

Design

The most successful FPS games have very simple interfaces: the main portion of the interface is dedicated to a large viewing area that contains the game world. Small status areas around the perimeter are allocated for information regarding the player's health and equipment. Most games also have a radar mini-map showing the direction to enemies or other players, as well as a separate screen containing a map of the current game level.

The robot interface focuses on three sources of information: a video stream from the robot's PTZ camera, a "radar" view visualizing the range sensor data (sonar and infrared, combined) from the robot, and a manually updated map view showing the robot's internal map of the navigated area. Together, these three windows, shown in Figure 2, bring the robot's view of the world to the remote operator.

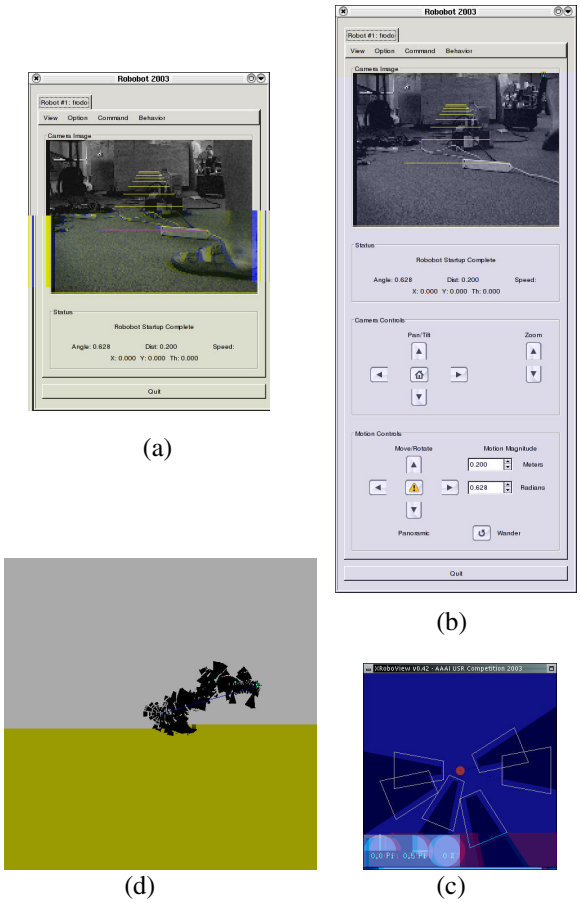

Figure 2: (a) Standard video and navigation interface screen, (b) expanded screen with button controls, (c) map screen showing a map built by the robot in USR run 3, (d) sensor and camera status display.

Simplifying the interface reduces on-screen clutter, but also precludes having visual duplicates of the robot's motion and camera controls. This means that most of the control must be done via the computer's keyboard, much like an FPS. The standard WASD directional pad is preferred by many right-handed gamers, because it puts most of the basic controls within one key of the fingers of the left hand, and frees up the right hand for using the mouse. Unfortunately, "mouselook", where moving the mouse rotates the player's head in the game world, was not something that could be implemented with the robot's existing PTZ camera implementation. Because of this limitation, a second d-pad control was created for the camera using UHJK on a QWERTY keyboard. With the operator's left and right hands resting on the keyboard, the vast majority of the interface controls are a single keypress away. The operator can use the fast-twitch muscles in their hand to react as quickly as possible to changing conditions in the robot's environment.

Interest Points

The mouse does play one important role in interacting with the video image received from the robot: it allows the operator to set and manipulate points of interest, including landmarks and victim locations, which are then stored in the robot's map. The operator's own human visual system provides the brain behind what the robot is seeing.

The operator can set an interest point on the ground by clicking on the image. The robot will use its current position and camera orientation data to calculate the location of the selected point in the world by assuming that the click occurred somewhere in the ground plane. This ability to orient on local landmarks reduces errors in localization.

The robot's internal localization is based on odometry and dead reckoning, which means that the robot's position and orientation become increasingly inaccurate as it moves. The orientation is the most likely of the three coordinates to be incorrect, so the operator can use previously set landmarks to correct the robot's orientation. If the operator sees that the current landmark--whose estimated position is highlighted in the display--no longer matches the real-world position where it was originally set, they can click on the landmark object again. The program uses the current position of the robot and the stored position of the landmark, along with the newly recalculated vector to the landmark, to reset the robot's global orientation. This correction can remove unwanted bends and curves on the map that are caused by a drift in the robot's orientation.

The ability to set landmarks significantly improved the accuracy of the robot's generated map, and allowed the addition of some unique features. The method used for capturing the landmark point is essentially the mathematical inverse of the calculations necessary to draw ground plane bars on the image, which permit the operator to estimate distances in the scene.

When drawing the ground plane bars, a distance from the robot (0.5 meter intervals) and a heading (directly ahead) are converted into a pixel coordinate. For the landmarks, the image position of the mouse click is converted into a global 2-D position in the ground plane of the world. The first step is to calculate the vertical angle between the ground plane and the line of sight from the camera to the interest point,

$$\phi = \left(\frac{\pi}{2} - tilt\right) + \left(r - \frac{H}{2}\right)\frac{FOV_v}{H} \qquad (1)$$

where *tilt* is the vertical tilt of the camera, *r* is the row (y pixel) in the image where the mouse event occurred, *H* is the height of the image in pixels, and FOV_v is the vertical field of view of the camera. This is a linear approximation to the true tilt angle of the pixel, as it assumes each pixel corresponds to an equal angle in the field of view, but works well enough for the field of view used by the robot.

This angle can be used to calculate the distance along the ground plane from the camera to the interest point, and thus from the robot's current location (designated as the center of the robot) to the interest point,

$$d = \frac{C_z}{\tan(\phi)} + C_x \qquad (2)$$

where C_z is the vertical distance between the camera and the ground and C_x is the distance from the camera to the center of the robot.

In addition to the distance, we can also calculate the orientation of the interest point relative to the robot using a linear approximation,

$$\theta = \frac{FOV_h}{W}\left(\frac{W}{2} - c\right) + pan \qquad (3)$$

where FOV_h is the horizontal FOV of the camera, *W* is the width of the image in pixels, *c* is the column (x pixel) in the image where the mouse event occurred, and *pan* is the horizontal pan position of the camera.

Once the system has the distance and orientation of the interest point relative to the robot's current position, it generates a fixed global map location for the interest point. As the robot moves, the estimated location of the interest point is displayed on the operator's screen. If the estimated location and the actual location differ, then the operator can fix the robot's estimated orientation by clicking again on the interest point. We make the assumption that errors in (x, y) location are small compared to errors in orientation, and those assumptions allow us to correct theta based on the user specified landmarks.

The ability to select interest points, however, is more generally useful than just correcting the robot's orientation. After the user selects a point, the system can query the user for other data that should be associated with it. In the USR competition, for example, creating a victim interest point brings up a form giving the user the opportunity to fill out the information required for each victim as part of the competition. Figure 3 shows an example victim dialog box.

Using the victim data collected from the operator, the interface was able to generate a web page on the fly. In a real USR situation, this page could be served over a wireless network from the operator's computer to rescue workers with properly equipped PDAs. The rescuers would then have instant access to the victim's status, an image of the victim and the surrounding structure, a map to the victim with a robot-navigable path highlighted, and images of any navigation points set along that path. Once the map accuracy is significantly improved, this kind of information access will be highly useful to human rescue workers. Returning this data also fulfills the robot's role as a scout into a potentially dangerous situation.

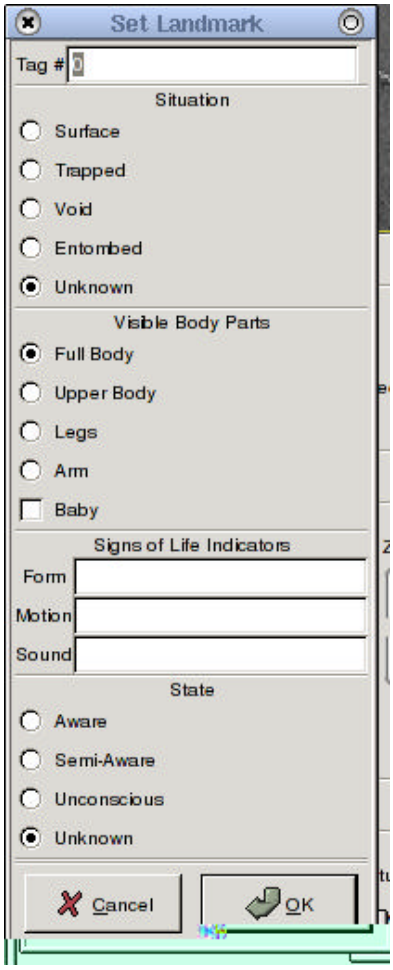

Figure 3: Dialog window for entering victim data.

Results and Evaluation

The competition provided a number of insights into the strengths and weaknesses of the user interface. Some of these were a result of design choices, while others were a result of hardware or software limitations on the robot itself.

One of the design choices, for example, was to make the default screen use the keyboard exclusively for low level control of the robot. For an experienced user, the improvement gained by using only the keyboard exclusively is valuable and saves time relative to a mouse. Unfortunately, having a large number of controls accessible only by potentially hard-to-remember keystrokes dramatically increases the interface learning curve. An operator who has trained on the interface will have no problems, and a new operator who has experience playing FPS games with similar key layouts will be able to adjust quickly.

New operators with little or no gaming experience, on the other hand, may find the interface too obscured. During the time that they would spend learning the keyboard layout they would be effectively useless in a real operational setting, due to very slow response times and an increased incidence of mistaken keypresses that could endanger the robot through inadvertent motion.

This problem became obvious when a new operator with effectively no gaming experience attempted to use the interface. Even after a brief training period, and making the mouse-based controls visible, the operator was barely able to get the robot to move, let alone use it to search for victims. When considering the trade-off between efficiency for a trained user and usability for a wide range of inexperienced users, the increased efficiency granted by the keyboard interface is more important in the field. However, the mouse-based interface is still necessary if there is a need for naive users to manage the robots. Training a new user would not require a significant investment in time or resources, but enough to acclimate the user to the interface.

One of the limitations of the underlying system that we found during the competition was that the landmark correction system itself was error-prone. Setting a landmark too close to the robot or setting and resetting landmarks too often could lead to sudden jumps in orientation. Setting one landmark some distance ahead of the robot and then not setting a new landmark until the first one was reached gave better maps. Since the setting of landmarks is totally up to the discretion of the operator, this practice either comes with experience or as part of the operator's training.

A second limitation of the underlying system was that the autonomous navigation was too jerky to permit effective monitoring of long forward motions by the operator. Thus, although the magnitude of the robot's forward motion or rotation could be set to an arbitrary value before sending a move or rotate command, we found that small rotations or motions worked best. Asking the robot to travel several meters in a straight line using its own obstacle avoidance simply did not work well in the enclosed spaces with noisy sensors. This is clearly an area for future improvement in the underlying navigation system.

Making large rotations (greater than an eighth of a turn, or about 45˚) seriously exacerbated the motion errors, and also tended to disorient the operator when the video blurred and lagged. Using short forward motions (less than 0.5 meters) and small angles (around $\pi/10$ radians) allowed the operator to nudge the robot in the desired direction without worrying as much about the robot's own behaviors suddenly kicking in to avoid an obstacle. Smaller motions also made far more accurate maps, which made the victim data web pages more useful.

For the camera's PTZ controls, a small step angle (around $\pi/8$ radians) was set for tilting and panning. There are also *jump-to* options for moving the camera to the limit of its PTZ ranges, as well as for quickly resetting the camera to a straight ahead view.

Finally, user disorientation can be a serious problem for any sort of tele-operation interface. For this project, there were several occasions where the operator was driving the robot around with the camera off-center, so the operator's perception of forward did not match the actual orientation of the robot. This type of human error slows down progress considerably, since the operator is constantly fighting the robot's correct behavior. When using veto mode (when the operator completely overrides the robot's behaviors), the operator must be very careful to orient themselves and the robot properly. Otherwise, they stand a chance of damaging the robot or getting it trapped. Usually this problem was caused by the operator ignoring one of the status displays, like the camera orientation or the range information. These problems inspired the operator mantra "Always trust your robot!", since a remote operator is forced to make some (possibly incorrect) assumptions about the robot's situation.

Conclusions and Future Work

One of the biggest weaknesses of the interface actually has nothing to do with the interface itself: the frame rate of the video transmitted by the robot is too low. The choppy image disconnects the operator from the reality viewed by the robot, and may allow some important information to be missed while the robot is in motion. Because of this, it is hoped that the video transmission speed can be improved independently of the interface program.

There are two ways of improving the video rate, and both of them will probably be implemented in the future. The robot's wireless system could be switched from the older and slower 802.11 protocol to the faster 802.11b protocol, while sacrificing some of the operational range, thus increasing the bandwidth of the signal. In addition to increasing the bandwidth, we can implement some form of compression, since the video data is currently sent as raw data through IPC over TCP/IP. The compression may involve using a standard image compression library to reduce the size of each video frame, and it could even use some form of temporal compression so that only those image pixels which have changed since the last frame are transmitted. As well as increasing the frame rate, the system could also use the added bandwidth make the image larger. A higher resolution image would have a marked effect on picture quality.

The interface as it stands right now is certainly usable, and by most standards is decent. However, there is still room for improvement. Before changing the UI itself, communication from the robot to the user needs to be improved. Most of these steps are a form of idiot-proofing to prevent the user from making a potentially catastrophic error. Alerts will be added to prevent the robot from moving forward if the PTZ camera is not facing forward, to avoid crashes when using veto mode, and to look for sudden and impossible jumps in position or orientation. Such emergency information should prevent the operator from accidentally damaging the robot through human error. One thing that might improve the initial experience for a new operator would be the use of a keyboard overlay with clearly labeled controls. For a larger hardware investment, some sort of custom joystick and keypad system might make the interaction with the interface even easier to use.

All of the rest of the potential changes fall under the umbrella goal of making the interface even more like an FPS game. The first is to emphasize the video feed, making it the largest displayed widget. This means improving the video transmission to allow for a larger image, as well as removing unnecessary widgets and controls from around the video display. Once this is done, more of the status controls can be moved from textual elements to a transparent heads-up display [HUD] system overlaid onto the video itself. This would eliminate the need for the separate xroboview application, since the PTZ display and "radar" would both be drawn (albeit on a smaller scale) directly onto the video image instead of in their own window. Finally, because the map already only updates at the command of the user, it would be moved from the separate xmap application to being displayed in place of the video image. Toggling between the normal view and the map screen would be allowed only when the robot was stationary.

Implementing mouselook--where the orientation of either the robot or the camera follows the mouse--while in video mode would also be an improvement.

Last, but certainly not least, the entire view that is now contained in one image should become a full-screen interface, with all of the afore-mentioned changes implemented. Building all of this functionality into a HUD, and including the afore-mentioned operator alerts should allow the operator to avoid some of the problem that had to be dealt with during the competition. At that point the interface would be indistinguishable from any number of FPS games currently on the market. This complete transformation is the ultimate goal of the USR tele-operation interface.

Additional work is planned for the software running on the robots themselves. This work will affect obstacle avoidance/path planning, mapping, and cooperation between the robots. The first major addition is a new navigation module,. At the core of this new module is the concept of Velocity Space, which will do a better job of keeping the robots' trajectories more even, improving their speed, and improving their response to commands (Fox *et. al.* 1997). The current control module is not perfectly suited to the task of tele-operated urban search and rescue, but the new navigation module should fill the niche quite nicely.

In addition to the navigation module, we will be improving the mapping module, and hopefully integrating simultaneous localization and mapping [SLAM] to get more accurate maps. The new mapping system will also provide a

more general solution which can be applied to other problem domains (such as Robot Soccer) in a more effective manner.

Related to this is our plan to enable the robots to communicate with each other-- when synthesized with SLAM, our robots as a group should be able to generate very good maps. Some level of strategy will also be explored with multiple robots, such as having three robots, each following the one in front. When a decision point (fork in the path) is reached, one robot will stay behind, to enable us to more quickly explore an area without backtracking.

Acknowledgements

This work was supported in part by NSF IIS-0308186, and the American Association for Artificial Intelligence.

References

[1] Dieter Fox, Wolfram Burgard, and Sebastian Thrun, "The Dynamic Window Approach to Collision Avoidance", *IEEE Robotics & Automation Magazine*, 4(1), March 1997.

[2] B. A. Maxwell, L. A. Meeden, N. S. Addo, P. Dickson, N. Fairfield, N. Johnson, E. G. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter, E. Silk, 2001, "REAPER: A Reflexive Architecture for Perceptive Agents", *AI Magazine*, American Association for Artificial Intelligence, 22(1): 53-66.

[3] B. A. Maxwell, N. Fairfield, N. Johnson, P. Malla, P. Dickson, S. Kim, S. Wojtkowski, T. Stepleton, "A Real-Time Vision Module for Interactive Perceptual Agents", *Machine Vision and Applications*, 14, pp. 72-82, 2003.

[4] Reid Simmons and Dale James, *Inter-Process Communication: A Reference Manual*, Carnegie Mellon University, March 2001.

[5] *Unreal Tournament*, Epic Games Inc, 1999.