Experiments in Adjustable Autonomy

Jacob W. Crandall, Michael A. Goodrich

Computer Science Department
Brigham Young University

Abstract

Human-robot interaction is becoming an increasingly important research area. In this paper, we present our work on designing a human-robot system with adjustable autonomy and describe not only the prototype interface but also the corresponding robot behaviors. In our approach, we grant the human meta-level control over the level of robot autonomy, but we allow the robot a varying amount of self-direction with each level. Within this framework of adjustable autonomy, we explore how existing robot control approaches can be adapted and extended to be compatible with adjustable autonomy

Introduction

The purpose of this research is to develop human-centered robot design concepts that apply in multiple robot settings. More specifically, we have been exploring the notion of adjustable autonomy and are constructing a prototype system. This prototype system allows a human user to interface with a remote robot at various levels of autonomy: full autonomy, goal biased autonomy, waypoint-based autonomy, intelligent teleoperation, and dormant. The objective is to allow a single human operator to interact with multiple robots and do so while maintaining reasonable workload and team efficiency.

Related Literature

Relevant research in human-robot interaction can be loosely classified under five topics: autonomous robots, teleoperation, adjustable autonomy, mixed initiatives, and advanced interfaces. Of these topics, research in teleoperation is most mature; we refer to Sheridan's work for an excellent overview of these topics [15]. Perhaps the most difficult obstacle to effective teleoperation occurs when there are communication delays between the human and the robot. The standard approach for dealing with these issues is to use supervisory control. Work on teleautonomy [5] and behavior-based teleoperation [16] are extensions to traditional supervisory control that are designed specifically to account for time delays.

Alternative approaches to teleautonomy that focus on the operator include the use of predictive displays [11] and the use of intelligent interface assistants [12]. Approaches that focus more on the human-robot interaction as a whole include safeguarded teleoperation [8, 10], mixed initiative systems [7], and adjustable autonomy-based methods [6].

Autonomous robot control and vehicle design has an extensive history. A complete review of the literature is beyond the scope of this paper, but we do note the seminal work of Brooks with behavior-based robotics [4]. We further note the excellent textbooks on the subject by Murphy [13] and by Arkin [3]. There are many approaches to behavior-based robotics, but in this paper we focus on approaches based on utilitarian voting schemes [14]. Hierarchical approaches, which are the other major approach to designing autonomous vehicles, are characterized by the NIST RCS architecture [1, 2].

Autonomy Modes and Justification

The purpose of this section is to describe the levels of autonomy that are being included in our human-robot system. Additionally, we discuss how the different autonomy levels are implemented. In the system we describe, the operator is given the authority to switch autonomy modes, but, within each mode, the robots have some authority over their behaviors.

Time Delays and Neglect

In designing an architecture that allows a human to interface with multiple robots, it is desirable to equip robots with enough autonomy to allow a single user to service multiple robots. To capture the mapping between user attention and robot autonomy, we introduced the neglect graph in Figure 1 [9]. The idea of the neglect graph is simple. Robot A's likely effectiveness, which measures how well the robot accomplishes its assigned task and how compatible the current task is with the human-robot team's mission, decreases when the operator turns attention from robot A to robot B; when robot A is neglected it becomes less effective.

0-7803-7087-2/01/$10.00 © 2001 IEEE

A common problem that arises in much of the literature on operating a remote robot is time delays. Round-trip time delays between earth and Mars are around 45 minutes, between earth and the moon are around 5 seconds, and between our laptop and our robot around 0.5 seconds. Since neglect is analogous to time delay, we can use techniques designed to handle time delays to develop a system with adjustable autonomy. For example, when the operator turns attention from robot A to robot B, the operator introduces a time delay, albeit a voluntary one, into the interaction loop between the operator and robot A. Depending on how many robots the operator is managing and depending on the mission specifications, it is desirable to adjust how much a robot is neglected. Adjusting neglect corresponds to switching between techniques for handling time delays in human-robot interaction.

As the level of neglect changes, an autonomy mode must be chosen that compensates for such neglect. In the literature review, several schemes were briefly discussed for dealing with time delays. Schemes devised for large time delays are appropriate for conditions of high neglect, and schemes devised for small time delays are appropriate for conditions of low neglect. At the lowest neglect level, shared control can be used for either instantaneous control or interaction under minimal time delays; at the highest neglect level, a fully autonomous robot is required.

We are now in a position to make two observations that appear important for designing robots and interface agents. First, the following rule of thumb seems to apply: *as autonomy level increases, the breadth of tasks that can be handled by a robot decreases.* Another way of stating this rule of thumb is that as efficiency increases tolerance to neglect decreases. Second, the objective of a good robot and interface agent design is to move the knee of the neglect curve as far to the right as possible; a well designed interface and robot can tolerate much more neglect than a poorly designed interface and robot.

Autonomy Modes

We have constructed (a) a set of robot control programs and (b) an interface system that allows a human to communicate with multiple robots (specifically, Nomad SuperScout robots) via an 11Mb/s wireless ethernet. We first focus on our robot control algorithms, which are built on utilitarian voting schemes. After discussing these control algorithms, we will discuss how we have used these algorithms to support adjustable autonomy in robot systems.

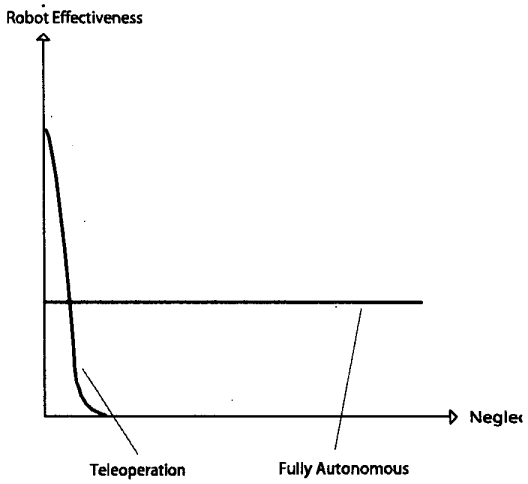

Figure 1 The neglect curve. The x-axis represents the amount of neglect that a robot receives, which can be loosely translated into how long since the operator has serviced the robot. The y-axis represents the subjective effectiveness of the robot. As neglect increases, effectiveness decreases. The nearly vertical curve represents a teleoperated robot which includes the potential for great effectiveness but which fails if the operator neglects the robot. The horizontal line represents a fully autonomous robot which includes less potential for effectiveness but which maintains this level regardless of operator input. The dashed curve represents intermediate types of semi-autonomous robots, such as a robot that uses waypoints, for which effectiveness decreases as neglect increases.

Utilitarian Voting Scheme for Navigation

Our utilitarian voting scheme uses three behaviors: a goal-achieving behavior, an obstacle-avoiding behavior, and a vetoing behavior.

The Goal-Achieving and Obstacle-Avoiding Behaviors At each iteration of our algorithm, eight voters are selected. Initially, an input vector (which can come from a variety of sources, depending on the current autonomy mode) is given to the robot. This input vector is considered an initial vote that proposes a magnitude and a direction for the robot to travel. The robot interprets this initial vote as a suggestion, and selects seven of its sonars to cast "votes" as well. These seven votes, along with the initial vote, determine the "best" direction for the robot to travel.

The way in which the seven other voters are selected is as follows: each sonar is assigned an angle value that is based on the angle it forms from the center of the robot, with, for example, sonar 0 corresponding to 90 degrees and sonar 12 corresponding to zero degrees (see Figure 2). We find the sonar for which the absolute value of the sonar angle minus the angle of the input vector is the smallest. In the case of figure 2, this sonar is sonar 14. This sonar and the three sonars adjacent to this sonar on both sides are the sonars that will affect the direction the robot chooses to take. The indices of these sonars are then put into an array S of voters. Continuing the example from the diagram, our array would have the following values: S = (1, 0, 15, 14, 13, 12, 11).

Next we define a rejection array R = ($R_0, R_1, R_2, R_3, R_4, R_5, R_6$) and a pull array P = ($P_0, P_1, P_2, P_3, P_4, P_5, P_6$), were each element of R and P have magnitudes between 0 and 1. These arrays designate the voting priorities that the voters have for rejecting or accepting a direction of travel. The votes V that each of the seven sonars casts in determining the "best" direction is obtained in the following way:

For all $S_i \in S$
  If (S_i <= WarningDist)
    $V_i = \left(\frac{S_i - WarningDist}{WarningDist} \right) * R_i$
  else if (S_i > SafeDist)
    $V_i = P_i$
  else
    $V_i = 0$

where SafeDist and WarningDist are predefined distances.

Figure 2 shows the regions of how the voter casts its vote. In effect, each voter (sonar) casts a vote on how good its direction is. Let d_i represent the sonar reading for sonar i. If d_i > SafeDist, the voter casts a "goal-achieving" vote. This vote is cast in a way to help the robot find an opening to reach its goal(s). The strength of the vote cast depends on the priority of the voter. If d_i <= *WarningDist* the voter casts an "obstacle-avoiding" vote. The vote is cast in a way to help the robot to avoid obstacles that are near it. The strength of this vote depends on both the priority of the voter and d_i. As d_i approaches zero, the vote cast by this voter approaches R_i. If *WarningDist* <= d_i < *SafeDist*, then the voter casts a neutral vote, and the voter has no effect on the outcome.

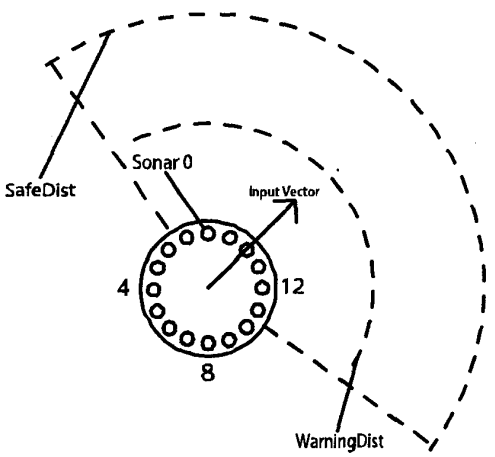

Figure 2 The circle with 16 small circles within it represents and abstract view of the robot with its sonars. The input vector is a vector indicating the general direction the robot should attempt to go, or the initial vote cast. The dotted part-circles represent the boundaries for what kinds of votes the sonar readings will cast. If the sonar reading falls within the "WarningDist" section, the sonar votes in opposition to its direction. If the reading falls between the "WarningDist" and "SafeDist" dotted part-circles, a neutral vote is cast, and if the reading falls beyond "SafeDist," a vote in favor of the sonars direction is cast.

After the above calculations are performed, we have an array of votes V (note that each assigned vote V_i corresponds to a sonar and, therefore, an angle as well) plus the input vector (the initial vote). Thus in all, we have eight vectors which will determine the "best" direction θ the robot should take. The x and y components of the "best" direction vector are then computed:

$$x = IV * Mag * \cos(Angle) + \sum_{i=0}^{6} V_i * \cos(SonarAngle_i)$$

$$y = IV * Mag * \sin(Angle) + \sum_{i=0}^{6} V_i * \sin(SonarAngle_i)$$

where *IV* is the weight the initial vote receives, *Mag* is the magnitude of the input (together, *IV* and *Mag* constitute the priority of the initial vote), *Angle* is the angle of the input vector, and *SonarAngle* is an array that maps each weight W_i to an angle.

We then compute θ: $\theta = \tan^{-1} \frac{y}{x}$.

As should be noted, we have left many of the variables with undefined values in the description of the voting scheme. Our current implementation sets the vector R to (.1, .4, .7, .8, .7, .4, .1), the vector P to (.1, .45, 1.0, 1.0, 1.0, .45, .1), *SafeDist* to 65 inches, *WarningDist* to 40 inches, and IV to 1.4. In our future work we will analyze why these values tend to work well, and what improvements can be made.

The Vetoing Behavior The "best" direction θ is the direction the robot selected. However, the algorithm doesn't guarantee that this direction is "safe." To guarantee that the robot will not collide with any object that it can see, we have also added a feature that supports guarded motion [8, 10]. We use a simple algorithm in which a "safe" region is defined by the sonar readings. By predicting where the robot will be at some future time t, the robot can determine if it will leave this region anytime in the near future if it continues the course it has selected. If the robot would leave this "safe" zone anytime in the near future, the "best" direction is vetoed and a different initial vector must be selected.

Adjustable Autonomy

The input vector mentioned previously can be found in a number of ways and, loosely, constitutes the goal of the robot. This allows us to use the same algorithm to direct the robot with different autonomy modes, since we can change the autonomy modes simply by obtaining the input vector in a different way. We have used this utilitarian voting scheme to implement three robot control programs. These control programs represent three different levels of autonomy.

Teleoperation – This is a shared control system. We use a Microsoft sidewinder joystick to obtain the desired input vector. This assisted teleoperation appears to relieve a lot of the workload from the human operator. Future work will validate this claim.

Waypoints and Heuristics – The input vector can be obtained by goals and heuristics that a human controller assigns to each robot. The human controller may drop icons on the map of the environment to influence the decisions that the robot makes. In our system, we have goal icons that indicate a robot's destination, arrow icons that tell the robot the general direction it should go when it is in a certain location, and rejection icons that indicate to the robot places that it should avoid. The vector obtained by the summation of these forces is the input vector for the robot in this mode.

Autonomy – If we assume that the input vector is always pointed straight ahead, the robot becomes an autonomous wanderer. This primitive wandering mode has shown to be quite remarkable for random exploration in the real world. Influencing a robot operating on this autonomy level with some kind of goal would give this mode added usefulness.

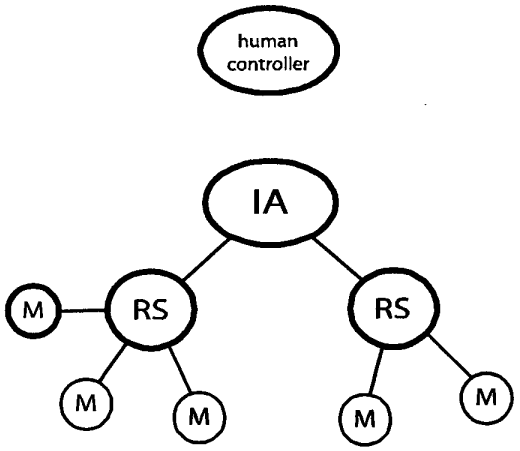

Figure 3 Diagram of the communication architecture of the human-robot system. The interface agent serves as the communication link between the robots and the human controller. Any number of control modules can be loaded onto each robot. The lines between nodes represent TCP/IP socket connections, while the dashed line indicates communication between human and machine. This architecture provides easy use of the principles of adjustable autonomy as the user may access the various autonomy loads of each robot through the mediating interface agent.

Communication Architecture

To facilitate improved communication between a human and robots in a human-robot system, we have developed an architecture that incorporates a human operator, robots, robot control programs, and an interface agent. This architecture is useful for robot systems that provide adjustable autonomy. Figure 3 provides a diagram of our architecture.

Interface Agent

Communication between the various robots and human operator in our human-robot system takes place through the mediating interface agent. Since communication between human and robot must be two-way, the interface agent must be able to transfer information in a sensible and understandable form from human to robot and from robot to human. This

communication of information is done through a graphical user interface.

As we mentioned earlier, the object of a good robot and interface agent design is to move the knee of the neglect curve from Figure 1 as far to the right as possible. To do so, the human operator must be able to easily sense what is going on with the robots in his or her system. Additionally, he or she must be able to communicate as naturally as possible with them.

Figure 4 is a screen shot of the interface agent's GUI. The bottom left corner of the GUI provides a list of the robots in the system and the tasks that have been assigned to each of them. In the bottom middle, the "cockpit" of the robot currently being serviced is displayed. The cockpit includes the readings from a digital compass, a video image captured from the robot, and a graphical display of the robot's sonar readings. The bottom right corner of the GUI displays the general state information of the robot currently being serviced. This part of the GUI provides the human controller with the ability to queue tasks on that robot. For example, if a robot is currently performing a certain task, but its help is temporarily wanted elsewhere, the human controller may assign the robot a new task. Before completing the old task, the robot performs the more urgent task. The old task is put into a queue to be performed at a later time. The center of the GUI is a grid that contains a 2-D "god's eye" view of the environment that the robots have explored. This view contains graphical information of the current location of each robot, the map the robots have built and the goals, waypoints, and heuristics that have been assigned to the robots. Additional features of the GUI include drop down menus, dragable icons and buttons.

Although this user interface has proved useful and has many good features, improvements are needed to improve natural communication. Future work will include these improvements [9].

Robot Server

The robot server on each robot is the center of communication for that robot. This program controls the robot after receiving commands from other programs, and sends information to the interface agent. Several basic functioning autonomy levels are made available on the actual robot server so as to always provide the human user with some levels of autonomy on each robot. Additional autonomy levels can be loaded onto the robot online through additional program controllers.

Control Programs

In addition to having a good communication system, the robots must have the ability to perform at many levels of autonomy. The human operator must be able to direct the robots at various levels of autonomy so as to be able to keep a balance between helping individual robots to carry out very complex tasks, and not loosing "sight" of the rest of the robot team.

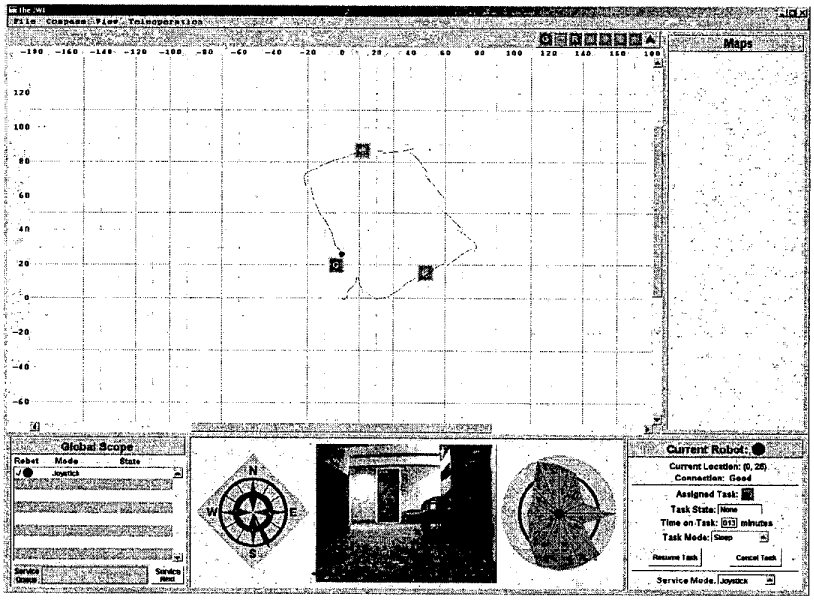

Figure 4 A screen shot of the Interface Agent's GUI. The GUI includes sensor readings from the robots, robot state, job queuing, and a "god's-eye" perspective of the environment the robots are exploring.

For these purpose, our human-robot communication architecture allows the human operator to load up various common and special purpose controller programs at startup and during runtime. When these controller programs are loaded onto the robot, the human controller can select them and switch between them through the interface agent.

As can be seen in Figure 3, the controller programs on each robot communicate, with the robot server on that robot. This is done through a simple text protocol in which the controller program receives information about the state of the robot, and directs the robot server to carry out commands on the robot.

This architecture is effective in providing adjustable autonomy to the robot system because of the ability it gives the human controller to use robots at any level of autonomy the robot is capable of performing. Controller programs for various autonomy levels need only be loaded onto a robot, and the human can easily access the autonomy modes.

Conclusions

We have built a system that supports adjustable autonomy. Adjustable autonomy in a robot system is facilitated by an interface agent, which mediates between a human controller and the robots in the system. The robots must have the ability to perform at many autonomy levels, and the human controller must be able to access these modes.

References

[1] Albus, J.S. 1991. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics* 21(3):473-509.

[2] Albus, J.S. 2000. 4-D/RCS reference model architecture for unmanned ground vehicles. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation.*

[3] Arkin, R. C. 1998. *Behavior-Based Robotics.* Cambridge, Massachusetts: MIT Press.

[4] Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2:14-23.

[5] Conway, L.; Volz, R. A.; and Wlaker, M. W. 1990. Teleautonomous systems: Projecting and coordinating intelligent action at a distance. *IEEE Transactions on Robotics and Automation* 6(2).

[6] Dorais, G. A.; Bonasso, R. P.; Kortenkamp, D.; Pell, B.; and Schreckenghost, D. 1998. Adjustable autonomy for human-centered autonomouse systems on mars. In *Proceedings of the First International Conference of the Mars Society.*

[7] Fong, T.; Thorpe, C.; and Baur, C. 1999. Collaborative control: A robot-centric model for vehicle teleoperation. In *AAAI 1999 Spring Symposium: Agents with Adjustable Autonomy,.* Stanford, CA: AAAI.

[8] Fong, T.; Thorpe, C.; and Baur, C. 2001. A safeguarded teleoperation controller. In *IEEE International Conference on Advanced Robotics* (ICAR).

[9] Goodrich, M. A.; Olsen, D. R.; Crandall, J. W.; Palmer, T. J. 2001. Experiments in Adjustable Autonomy. To appear in *Proceedings of the IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*

[10] Krotkov, E.; Simmons, R.; Cozman, F.; and Koenig, S. 1996. Safeguarded teleoperation for lunar rovers: from human factors to field trials. In *IEEE Planetary Rover Technology and Systems Workshop.*

[11] Lane, J.C.; Carignan, C. R.; and Akin, D. L. 2000. Advanced operator interface design for complex space terobots. In *Vehicle Teleoperation Interfaces Workshop, IEEE International Conference on Robotics and Automation.*

[12] Murphy, R. R., and Rogers, E. 1996. Cooperative assistance for remote robot supervision. *Presence* 5(2): 224-240.

[13] Murphy, R. R. 2000. *Introduction to AI Robotics.* MIT Press.

[14] Rosenblatt, J. K. 1995. DAMN: A distributed architecture for mobile navigation. In *Proc. Of the AAAI Sptring Symp. On Lessons Learned from Implemented Software Architectures for Physical Agents.*

[15] Sheridan, T. B. 1992. *Telerobotics, Automation, and Human Supervisory Control.* MIT Press.

[16] Stein, M. R. 1994. *Behavior-Based Control for Time-Delayed Teleoperation.* Ph.D. Dissertation, University of Pennsylvania.