A Safeguarded Teleoperation Controller

Terrence Fong[1], Charles Thorpe[1] and Charles Baur[2]

[1]The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA

[2]Institut de Systèmes Robotiques
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne EPFL, Switzerland

Abstract

This paper presents a control system for mobile robots. The controller was developed to satisfy the needs of a wide range of operator interfaces and teleoperation in unknown, unstructured environments. In particular, the controller supports varying degrees of cooperation between the operator and robot, from direct to supervisory control. The controller has a modular architecture and includes interprocess communications, localization, map building, safeguarding, sensor management, and speech synthesis. In this paper, we describe the design of the controller and discuss its use in several applications.

1 Introduction

Since 1997, we have been developing tools and technology for vehicle teleoperation. Our goal is to make vehicle teleoperation easier and more productive for all users, novices and experts alike. Thus, we have been developing a new teleoperation control model (collaborative control) and operator interfaces incorporating sensor-fusion displays, gesture and haptic input, personal digital assistants (PDA), and the WorldWideWeb[5][6].

Although all our interfaces support remote driving, each interface has different characteristics and is intended for use under different conditions. Some of our interfaces are geared towards novices. Other interfaces are designed for trained experts. Additionally, we employ numerous teleoperation control models: continuous, shared/traded, collaborative, and supervisory. Finally, our interfaces operate on a variety of hardware (PDA to workstations) and over a wide range of communication links (28.8 kbps to 10 Mbps, with and without delay).

To meet the requirements of our interfaces, we have developed a mobile robot controller which supports varying degrees of cooperation between operator and robot. We designed the controller to be modular and to function in unknown and/or unstructured environments, both indoor and outdoor. Most importantly, however, the controller provides continuous safeguarding to ensure that the robot is kept safe regardless of control mode, operator input, and environmental hazards.

2 Related Work

2.1 Safeguarded Teleoperation

The safeguarded teleoperation concept was developed to enable remote driving of a lunar rover[11]. Command fusion enables operators to share control with a safeguarding system on-board the robot. In benign situations, the operator has full control of vehicle motion. In hazardous situations, however, the safeguarder modifies or overrides operator commands to maintain safety. The safeguarder, therefore, exhibits many characteristics of autonomous systems such as perception, command generation, etc.

Unlike the system described in [11], which was designed exclusively for untrained operators and continuous control, our controller supports a range of users (novices to experts) and intermittent as well as continuous control. Moreover, in addition to safeguarding vehicle motion (preventing collision and rollover), our controller monitors system health (vehicle power, motor stall, etc.) and "safes" the vehicle when necessary.

2.2 Control Systems for Teleoperation

Numerous researchers have addressed the problem of designing control systems for teleoperation. Although some restrict the term *teleoperation* to denote only direct, continuous control (i.e., no autonomous functions), we consider teleoperation to encompass the broader spectrum from manual to supervisory control[14]. Thus, teleoperation controllers encompass an extremely varied range of designs and techniques. The majority, however, can be described within the framework of one or more existing *robot control architectures*[9].

A parallel, three-layered control architecture for teleoperation of mobile robots is described in [13]. This controller provides reflexes for obstacle avoidance, plan learning, and compressed communications. A "generic" telerobotic controller is discussed in [8]. The design uses a network of low-level control behaviors switched on and off by a high-level symbolic layer. A mobile robot control system with multisensor feedback is presented in [12]. The system supports four teleoperation control modes (direct, traded, shared, supervisory) and allows operators to interactively assist in environment modelling.

3 Design

3.1 Requirements

All teleoperation interfaces include tools and displays to help the operator perceive the remote environment, to make decisions, and to generate commands [5]. An effective teleoperation controller, therefore, provides resources to support these tools and displays. In particular, the controller must supply capabilities and sensory feedback which make the interface work well and the remote task easy to perform.

Our interfaces are intended for remote driving in unstructured, unknown environments. Thus, we designed our controller to emphasize navigation and motion control. To support a range of human-robot interaction, the controller must provide a variety of motion commands (Table 1).

Table 1. Motion control requirements

Control model	Motion commands
continuous	translation/rotation rates relative translate/rotate
shared/traded	translation/rotation rates absolute heading
collaborative	absolute heading translation/rotation rates relative translate/rotate pose (2D, 3D, path)
supervisory	relative translate/rotate pose (2D, 3D, path)

To support navigation, the controller must provide visual feedback (still images and/or video), spatial feedback (sensor-based maps), and situational feedback (robot health, position, command status). Additionally, because navigation is strongly dependent on perception, the controller is required to provide facilities for sensor management and processing. For example, sensor-based map building requires range data processing.

In order to support untrained users as well as operation in non-benign environments, the controller must be able to perform real-time, reactive safeguarding. Specifically, the controller should be capable of maintaining robot safety at all times. This entails avoiding collisions, avoiding rollover, monitoring health (power level, temperature, etc.) and safeing the vehicle when necessary.

Finally, because our interfaces operate on a variety of hardware, the controller must be able to function when using poor communication links. In particular, the controller, should still perform competently when limited bandwidth is available, when there is transmission delay, and when the link is unreliable.

3.2 Robot Hardware

We designed our controller to operate Pioneer[1] mobile robots. We are currently using a Pioneer2-AT (Figure 1) which is skid-steered and capable of traversing moderately rough terrain. It is equipped with a microprocessor-based servo controller, on-board computing (233 MHz Pentium MMX), 802.11 wireless ethernet.

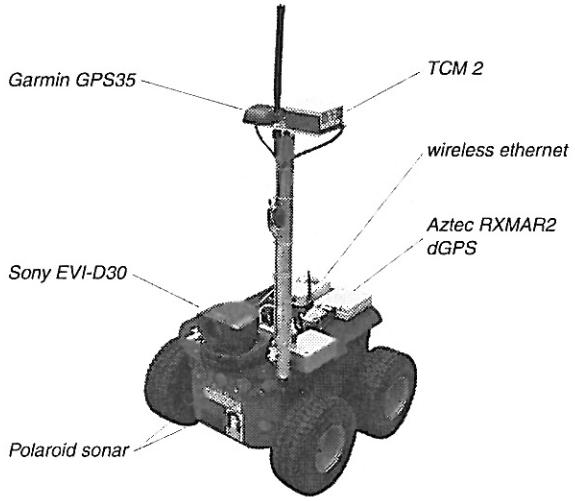

Figure 1. Pioneer2-AT mobile robot

We use numerous sensors for localization, mapping and operator feedback (Table 2). These include a pan/tilt/zoom color CCD camera, wheel encoders, differential GPS, a three-axis orientation sensor, and an ultrasonic sonar ring.

Table 2. P2AT sensor suite

Sensor	Description	Key characteristics
Sony EVI-D30	Color CCD camera 12x zoom, pan/tilt	4.4° to 48.8° HFOV ±100° pan, ±25° tilt
Garmin GPS35-HVS	12-channel C/A GPS receiver	20 cm resolution (Cartesian user grid)
Aztec RXMAR2	dGPS (RTCM) RDS receiver	up to 1m SEP
Precision Navigation TCM2	triaxial magneto- meter and biaxial inclinometer	±2° heading accuracy 0.1° tilt accuracy ±20° tilt range
Polaroid 600 sonar	time-of-flight ultrasonic ranging	15 cm to 10 m range 1% accuracy

We should note that the TCM2 (which is widely used) outputs roll, pitch, compass heading, magnetic field, and temperature measurements. Although the unit provides reliable static tilt data, dynamic performance and heading output is marginal at best [2].

[1] Pioneer is a trademark of ActivMedia, Inc.

3.3 Architecture

Our controller is implemented as a distributed set of modules, connected by interprocess communications. Some of the modules run standalone and operate asynchronously. Other modules, particularly those which process sensor data or operate robot hardware, have precise timing or data requirements and operate in the Saphira system.

Saphira is a framework for constructing mobile robot controllers and contains both a system and a robot control architecture [10]. The system architecture provides a micro-tasking operating system and functions for communicating with and operating robot hardware. The robot control architecture contains representations and routines for sensor processing, for environment mapping, and for controlling robot actions.

We use Saphira for several reasons: (1) it is a mature system and works well with Pioneer robots; (2) it provides efficient command fusion through fuzzy behaviors; (3) it is extensible, modular and portable; and (4) the micro-tasking operating system is synchronous and interrupt-driven, thus making it easy to implement modules with precise timing.

Table 3 lists the modules in our current controller and describes the function, execution style, and implementation of each. Figure 2 shows where the modules reside and how they are connected.

Table 3. Controller modules

name	function	exec. style	imple-mentation
Audio Manager	sound playback speech synthesis	asynch	standalone C
Camera Manager	camera control	asynch	standalone C
Hardware Control	servo control vehicle electronics	real-time	Pioneer μcontrol
Image Server	image capture	asynch	standalone C
Localizer	position estimation	synch (10 Hz)	Saphira C
MapMaker / MapServer	map building map generation	asynch	standalone C
Motion Control	high-level motion	synch (10 Hz)	Saphira C+behavior
Safeguarder	health monitoring motion safeguards	synch (10 Hz)	Saphira C+behavior
Sensor Modules	sensor processing	synch (10 Hz)	Saphira C
UIGateway	proxy server for user interfaces	synch (varies)	standalone C

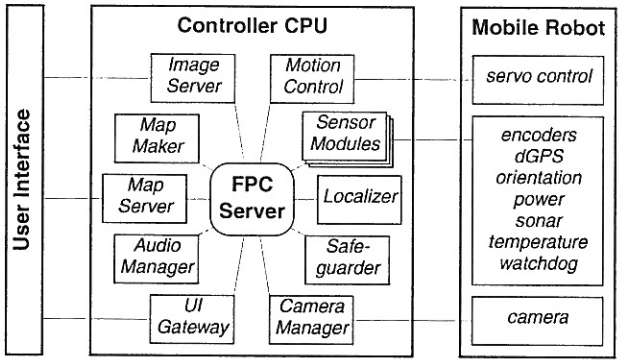

Figure 2. Controller architecture

3.4 Interprocess Communications

In the past, most robot software was designed as a single, monolithic block of code. Modern robotic systems, however, are constructed as a group of modules, each of which performs distinct processing functions. Modular design provides many benefits including encouraging team development, facilitating module implementation, and enabling distributed computation. At the same time, however, this approach requires that some mechanism be used to integrate modules and to distribute data between them. The most common mechanism is a network-based, interprocess communication toolkit.

Interprocess communication toolkits have long been used to support distributed and parallel computing. Although there are a large number of general purpose communication libraries, very few are appropriate for robotic applications. This is because the suitability of a toolkit is determined not merely by how efficiently it can move data, but rather by how well its communication paradigm (messaging model) and functions match the dataflow of the robot architecture. Thus, numerous interprocess communication toolkits have been developed for robotics including IPT, NDDS, NML, TCA/TCX/IPC, and RTC [7].

In our controller, we use the Fourth Planet Communicator (FPC) toolkit [4]. FPC's design was inspired by both message-based (e.g., TCA/TCX/IPC) and information-based (e.g., NDDS) systems. FPC uses a "publish and subscribe" framework with centralized caching for efficient, dynamically reconfigurable, and scalable data distribution.

We chose FPC for several reasons. First, it provides both reliable (for message sequences) and unreliable (for fast idempotent data) delivery. Second, its performance (message rate and latency) is well suited to the needs of our controller modules. Finally, it facilitates integration of diverse modules with multiple language interfaces (C, Java, Perl, TCL) and support for multiple operating systems (Linux, WinNT, IRIX, Solaris, HP-UX).

3.5 Modules

AudioServer

For some applications, particularly when the robot must operate around or with humans, audio plays an important role in human-robot interaction. Specifically, audio is a highly effective mechanism for conveying the robot's intent and for communicating information to humans. Thus, the AudioServer is designed to perform two functions: sound playback and speech synthesis.

We use sound playback to produce informative signals. For example, we use a train whistle to warn that the robot is approaching and to request that people move out of the way. We have found that a train whistle produces a significantly better response (i.e., people pay more heed and react more positively) than a horn or klaxon.

We use speech synthesis for information which cannot be conveyed by sound alone, such as status messages ("turning right", "stop", etc.), health warnings ("low battery"), and alerts ("motor stall"). The AudioServer produces speech with the MBROLA speech synthesizer [3].

CameraManager

The CameraManager operates the robot's camera systems. its primary function is to control steerable CCD cameras, i.e., cameras mounted on or incorporating a positioning mechanism. The CameraManager is also used to configure imaging parameters (gain, aperture, magnification, etc.). Whenever it changes a camera's configuration, the CameraManager outputs a message describing the camera's state (position, magnification, field-of-view, etc.).

ImageServer

Remote driving is an inherently visual task, especially for unstructured and/or unknown terrain. In many vehicle teleoperation systems, video is the primary source of visual feedback. High-quality video, however, uses significant communication bandwidth. Moreover, for applications with poor communications (low bandwidth and/or high transmission delay), video may not be practical.

As an alternative to video, we have designed an event-driven ImageServer. It minimizes bandwidth consumption by outputting images only when certain events occur. Specifically, the ImageServer captures (or loads) a frame, compresses it into a JPEG image, and sends it whenever the operator issues a request, the robot stops, an obstacle (static or moving) is detected, or an interframe timer expires.

We have found that event-driven imagery is a flexible mechanism for visual feedback. For example, if an application allows use of a high-bandwidth, low-latency communication link, we set the interframe timer to a low value (e.g., 0.2 sec). This results in an image stream which provides a fair approximation of video. Alternatively, if the link is low-bandwidth or has high delay, we set the timer to a high value. In this case, images are transmitted only when important teleoperation events occur. Since we are most likely to be using intermittent control (e.g., waypoint-based driving) in this situation, event-driven imagery works well.

Localizer

The Localizer estimates vehicle position and orientation. On the P2AT, we estimate position using odometry and dGPS, and orientation using the TCM2 and odometry. When the Localizer is running, it continually outputs its pose estimate and localization uncertainty. The Localizer provides estimates in two coordinate frames.

The navigation (world) frame is inertially-fixed and locally-level: $\hat{x}$ is east, $\hat{y}$ is true north, and $\hat{z}$ is up (i.e., gravity aligned). When we have valid dGPS fixes, the world frame coincides with the regional Cartesian user grid (e.g., UTM). The body (local) frame is vehicle-fixed with the origin set at the center of rotation. On the P2AT, this is the mid-point of the longitudinal center-line at axle height. The body-frame axes are: $\hat{x}$ forward, $\hat{y}$ left, and $\hat{z}$ up.

MapMaker

Although image-based driving is an efficient command mechanism, it may fail to provide sufficient contextual cues for good situational awareness. Maps can remedy this by providing reference to environmental features, explored regions and traversed path. In addition, maps can be efficiently used for collision and obstacle avoidance.

The MapMaker builds maps using a 2D histogram occupancy grid and range sensors. Our method is inspired by [1], but has several differences. First, we use a fixed-sized grid (20x20m with 10cm cells). If the robot approaches a border, we shift cells to keep the robot in the grid. Second, each cell holds a signed, 8-bit certainty value[2] (CV). This wider range improves map appearance and speeds safeguarding (e.g., collision avoidance only considers cells with CV>0). Third, in addition to updating the grid when moving, we update when stopped. Finally, we update the entire grid to reflect localization uncertainty: as it increases, we increment/decrement CV's towards zero.

MapServer

The MapServer provides maps as images. Whenever it receives a request, the MapServer queries the MapMaker for the relevant histogram grid region and converts CV's to gray-level[3]. Clear areas appear as white, obstacles as black, and unknown as light-gray. The MapServer can generate

[2] CV range is -127 (clear) to 127 (obstacle). 0 indicates "unknown".

[3] gray-level = CV + 127

maps in either the world or local frame, with arbitrary resolution (it performs sub/super sampling) and of any extent (regions outside of the grid are marked "unknown"). Figure 3 shows some typical maps generated by the MapServer.

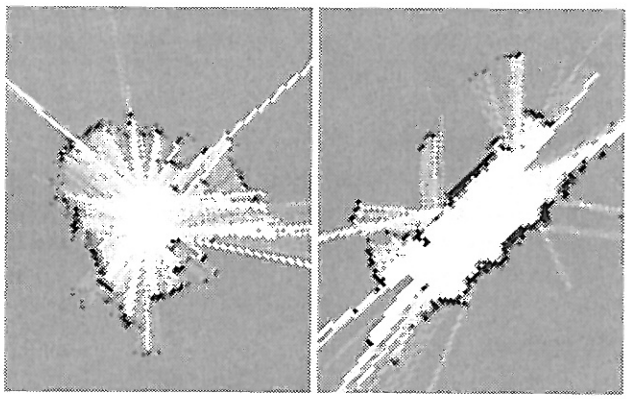

Figure 3. MapServer maps: room (left), corridor (right)

MotionController

The MotionController executes and monitors motion commands. It generates position and rate setpoints (translation and rotation) for the robot's low-level servo controller. Our current MotionController supports the motion command set shown in Table 4. Each command is implemented as a Saphira behavior. This allows all vehicle motion to be safeguarded (i.e., through command fusion with Safeguarder commands). Because Pioneer robots can turn in place, we use a "turn then move" motion strategy.

The MotionController continuously monitors the progress and status of each executing motion behavior. Whenever it detects lack of progress (e.g., due to safeguarding) or failure, the MotionController outputs a message to any connected operator interface.

Table 4. Motion control commands

command	control variable
translate	distance, rate
rotate	heading (relative/absolute), rate
vector	heading (absolute) + translate rate
pose	2D (x, y), 3D (x, y, heading), path

Safeguarder

The Safeguarder maintains vehicle safety. To avoid collisions, it scans the MapMaker's occupancy grid for obstacles. Our approach is similar to the first stage of [15], but instead of histogramming obstacle density, we compute distance to obstacles in the direction of motion. Whenever the robot approaches an obstacle, the Safeguarder reduces translation speed. If an obstacle reaches the standoff distance, the Safeguarder forces the robot to stop.

The Safeguarder prevents rollovers by monitoring vehicle attitude. Whenever roll or pitch exceeds a specified threshold for more than a short period, the Safeguarder forces the robot to stop. It also constantly monitors system health and takes action if it detects problems. In particular, the Safeguarder prevents vehicle motion if it detects low power, high temperature, excessive motor stall (indicative of drive obstruction), or hardware controller failure.

Sensor Modules

A Sensor Module interacts with a single sensor or a group of related sensors. Each module works like an operating system driver: it communicates with the device using sensor-specific protocols, processes the sensor data, then publishes the results for other controller modules to use. There are currently five Sensor Modules in the controller: GPS, Health, Odometer, Sonar, and TCM2.

The GPS module acquires GPS position fixes and transforms the data from geodesic coordinates to the regional Cartesian user grid. If data is unavailable, or of poor quality (e.g., high DOP), the GPS module outputs a warning.

The Health module monitors vehicle health sensors. At this time, these are power (battery voltage), temperature (environment), and watchdog (hardware controller).

The Odometer module processes wheel encoder data. It computes differential position and velocity based on encoder changes.

The Sonar module controls a ring of ultrasonic sonar. It is used to enable/disable transducers and to configure polling order (to minimize crosstalk). The Sonar module processes range data by applying a cut-off filter (ranges greater than a cut-off are discarded) and then transforming the range to a position (world frame).

The TCM2 module acquires orientation (roll, pitch, compass heading) data from the TCM2. To reduce noise, the module smooths the data using an exponential filter.

UIGateway

The UIGateway is a proxy server for user interfaces. It provides access to controller services (e.g., motion control) while hiding the controller's complexity. The UIGateway uses a simple message protocol which works well even over low-bandwidth connections. The protocol is text-based (which speeds integration of diverse interfaces) and synchronous (to reduce latency and to improve safety).

Whenever an interface is connected, the UIGateway continually monitors the connection. If it detects a communication problem (e.g., network outage) or that the interface is no longer responding, the UIGateway immediately stops the robot and closes the connection. Thus, the UIGateway ensures that operator commands are only executed while the interface is active and functioning.

4 Results

To date, we have used our safeguarded teleoperation controller with three operator interfaces having very different characteristics[6]. WebDriver operates using the World Wide Web. We conducted informal, indoor user tests and found that safeguarding improves the driving experience for novices. In particular, novices reported that safeguarding reduces their fear of "breaking something", especially when exploring unfamiliar rooms and corridors.

GestureDriver is based on visual gesturing. Hand motions are tracked with a color stereo vision system and mapped into motion commands. Since the primary interaction mode maps hand gestures (which are often noisy) directly to vehicle rates, safeguarding is needed to prevent collision and rollover during operator training.

PdaDriver (Figure 4) is our most recent interface and runs on a Casio Cassiopeia PDA. It provides a variety of driving modes and supports collaborative control (which enables the robot to query the operator for information and advice). We have used the PdaDriver for remote driving on paved roads, on benign natural terrain, and indoors. In all environments, we have found that the controller works well: waypoint-based driving is efficient, safeguarding prevents collisions (with fixed/moving obstacles), and audio enables interaction with humans in the environment.

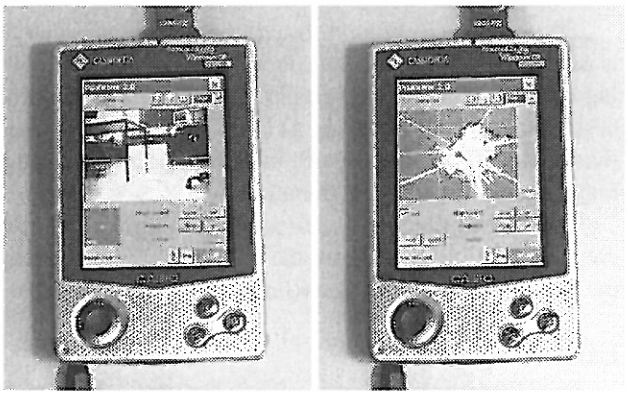

Figure 4. PdaDriver: image mode (left), map mode (right)

5 Conclusion

We have developed a teleoperation controller which supports remote driving in unknown, unstructured environments. Our controller differs from other teleoperation control systems because it satisfies the needs of a broad range of operator interfaces and control modes. In addition, the controller provides continuous safeguarding to maintain vehicle safety regardless of control mode, operator input, and environmental hazards.

Although the controller satisfies the needs of our current interfaces, further development would make it more robust. Better localization would aid navigation and motion control. Additional range sensors (stereo vision, ladar) would improve map building and provide better collision avoidance. Finally, wheel-slip and wheel-blocked sensors would enhance safeguarding.

Acknowledgements

We would like to thank Kurt Konolige for providing Saphira source code and his tireless support. This work was partially supported by a grant from SAIC and the DARPA ITO MARS program.

References

[1] Borenstein, J. and Koren, Y., "Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance", IEEE Journal of Robotics and Automation, 7(4), 1991.

[2] Deschler, M., "TCM2 Sensor Development", Technical Report, VRAI Group, EPFL, 1998.

[3] Dutoit, T., et. al., "The MBROLA Project: Towards a Set of High-Quality Speech Synthesizers", ICSLP, 1996.

[4] Fong, T., FPC: Fourth Planet Communicator, Fourth Planet, Inc., Los Altos, CA, 1998.

[5] Fong, T., and Thorpe, C., "Vehicle Teleoperation Interfaces", Autonomous Robots, Vol 11(1), 2001.

[6] Fong, T., Thorpe, C., and Baur, C., "Advanced Interfaces for Vehicle Teleoperation: Collaborative Control, Sensor Fusion Displays, and Remote Driving Tools", Autonomous Robots, Vol 11(1), 2001.

[7] Gowdy.J, "A Qualitative Comparison of Interprocess Communications Toolkits for Robotics", CMU-RI-TR-00-16, Carnegie Mellon University, 2000.

[8] Graves, A. and Czarnecki, C., "A Generic Control Architecture for Telerobotics", UMCS-99-3-1, University of Manchester, 1999.

[9] Hasemann, J-M., "Robot Control Architectures: Application, Requirements, Approaches, and Technologies", SPIE Intelligent Robots and Manufacturing Systems, Philadelphia, PA, 1995.

[10] Konolige, K. and Myers, K., "The Saphira Architecture for Autonomous Mobile Robots", in AI and Mobile Robots, (Bonasso, R. and Murphy, R., eds.), MIT Press, Cambridge, MA, 1997.

[11] Krotkov, E., et. al., "Safeguarded Teleoperation for Lunar Rovers: From Human Factors to Field Trials", IEEE Planetary Rover Tech. and Sys. Workshop, 1996.

[12] Lin, I. et. al., "An Advanced Telerobotic Control System for a Mobile Robot with Multisensor Feedback", IAS-4, IOS Press, 1995.

[13] Maslowski, A., et. al., "Autonomous Mobile Robot Controller for Teleoperation System", ISMCR, Prague, Czech Republic, 1998.

[14] Sheridan, T., *Telerobotics, Automation, and Human Supervisory Control*, MIT Press, Cambridge, MA, 1992.

[15] Ulrich, I., and Borenstein, J., "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots", IEEE ICRA, Leuven, Belgium, May 1998.