The WURDE Robotics Middleware and RIDE Multi-Robot Tele-Operation Interface

Frederick Heckel, Tim Blakely, Michael Dixon, Chris Wilson, and William D. Smart
Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045
One Brookings Drive
St. Louis, MO 63130
United States
{fwph,tmb1,msd2,ccw1,wds}@wustl.edu

Abstract

We have developed highly modular middleware for robotics programming and an interface for multi-robot teleoperation. WURDE provides abstractions for the communications, applications, and systems levels of robotic system development, which helps to isolate the developer from details not essential to the immediate task. RIDE is a control interface inspired by real time strategy games for tasking multiple robots at the same time, which increases the situational awareness of the operator and allows a single person to control many more robots than with single-robot interfaces. In this paper, we describe WURDE and RIDE and discuss how they were used in the 2006 AAAI Mobile Robot Competition and Exhibition.

Introduction

One of the major problems in robotics research is still the difficulty of developing software and adapting existing software for use on systems which may be highly customized. A great deal of effort is spent writing new implementations of existing algorithms– sometimes even in the same lab. Over the course of a few years, the same algorithm may be rewritten multiple times due to change of staff, new robots, major system overhauls, or software problems. Building middleware that is non-invasive (enabling simple adaptation of existing software), modular (allowing different algorithms and simulators to be tested easily), communication-agnostic (freeing the developer from dealing with low-level communications issues, and easy to use is currently a major priority in the community. We have developed WURDE, the Washington University Robotics Development Environment, to address each of these issues for our lab.

Using WURDE, we have developed a new robot tasking and control interface. RIDE, the Robot Interactive Display Environment, is a real time strategy (RTS) game inspired control interface. We have taken advantage of the common RTS style of gaming to build an interface which allows a single user to control many robots at once. This increase the overall situational awareness of the user and decreases the burden of control. In this paper, we describe both WURDE and RIDE, and discuss how they were used in our entry for the 2006 AAAI Mobile Robot Competition and Exhibition.

Background

Robot Architectures

Several robotics middleware packages have emerged over recent years. One of the most widely-used is Player (Gerkey *et al.* 2001). Player serves as an interface to many different types of robot devices, providing drivers that interface with many different pieces of hardware. In addition, the Stage and Gazebo simulators are available to use with programs written with Player. Player is C-based and tied to a TCP/IP-based communication protocol. It also lacks a module management system.

The MARIE (Côté *et al.* 2006) middleware provides tools which allow the adaptation of different communications protocols and applications. It is a very flexible system; our software differs by placing a very strong focus on ease of use, at a potential loss of flexibility.

The CARMEN (Carnegie Mellon University 2006a) package from Carnegie-Mellon University provides software to interface with several devices and provide access to a a number of important algorithms. CARMEN is a very complete package, but is tied to the CMU IPC communication system. It is also a C-based system. WURDE takes advantage of the object-oriented features of C++ while abstracting the communication protocol to reduce dependence on specific communication mechanisms. WURDE is also designd primarily as an environment for constructing and adapting new modules, and has a module monitoring and management system.

ADE (Kramer & Scheutz 2006) from Notre Dame is a java-based system. Apart from the language, the chief differences between ADE and WURDE are that ADE uses Java Remote Method Invocation, while WURDE uses a message passing protocol as its distributed computation mechanism. ADE also has a security model for controlling access to modules, something that WURDE currently lacks. The major difference is that WURDE primarily uses asynchronous communication and wraps the communication mechanism to simplify the process of adding additional communication protocols.

Real Time Strategy Interfaces

There is some previous work on RTS-style interfaces for robots. Jones and Snyder (Jones & Snyder 2001), describe

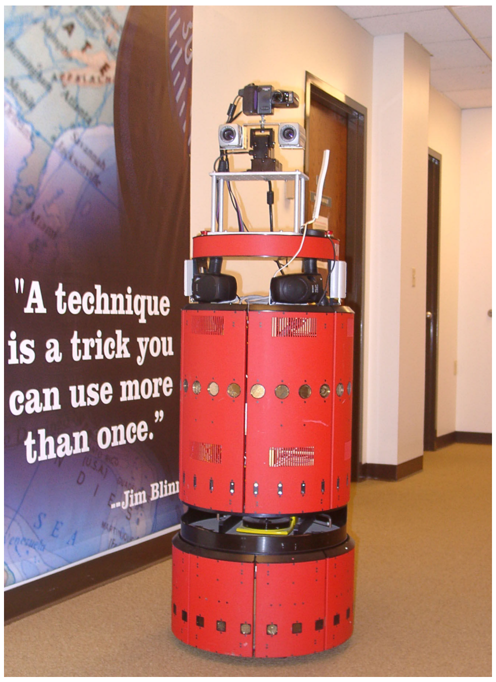


Figure 1: Lewis, our iRobot B21r research robot.

a system that is very similar to ours, although it is designed for the control of a small number of free-flying space robots. Our system differs from theirs in our extensive use of sensor and data visualizations and in our use of a sliding autonomy system that allows the robots to request assistance.

Parasuraman, Galster, and Miller describe a task-level control interface called Playbook (Parasuraman, Gastler, & Miller 2003), and evaluate its effectiveness on a simulated unmanned vehicle control task. Subjects controlled six simulated vehicles under a range of conditions. The interface was a two-dimensional representation of the world, where the only sensor visualization was a representation of the robot's field of view.

A number of robot simulations take advantage of first-person computer games technology (for example,the USAR simulation (Wang, Lewis, & Gennari 2003)), but typically do not take advantage of the associated interfaces. The game-based simulator described by Faust, Simon, and Smart (2006) allows humans to directly control human avatars, using a first-person interface, but this is still essentially a direct teleoperation interface.

The WURDE Middleware

The WURDE (Washington University Robotics Development Environment) middleware is a highly modular, abstracted library and set of utilities to simplify robotics development. WURDE does not require the use of any particular robot software architecture, but does assume that software will be written as a number of small, interconnected applications. The modular structure allows the researcher to easily test different algorithms and configurations of the

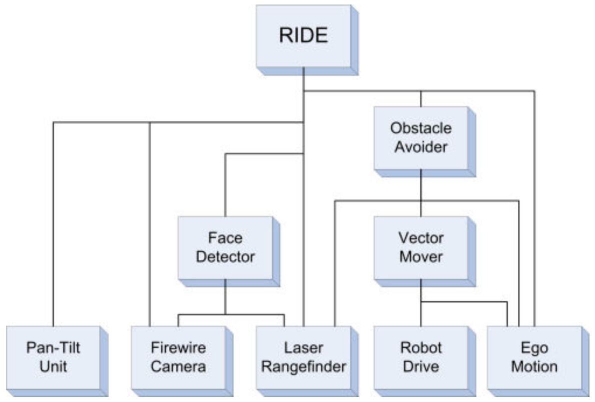


Figure 2: The WURDE architecture.

robot. In addition, it is very simple to switch between different robots and simulations using this method, since the low-level interfaces to specific hardware (or simulated hardware) are contained in a small number of modules that are easy to replace.

Our primary goal in designing WURDE was to develop middleware for robotics that is extremely simple to use, especially for those new to programming mobile robots. It is far too common to spend large amounts of time adapting programs written for other robots or writing common boilerplate code. Robotics development is hard enough without dealing with low-level software and communication issues that can be abstracted, so we aim to minimize these aspects of development so that researchers can focus on *research*. WURDE is designed to make it simple to write new communication adaptors, so it is not tied to a particular protocol. At present, we are using Carnegie Mellon University IPC (Carnegie Mellon University 2006b) for our primary communication adaptor.

Abstraction

WURDE uses four layers of abstraction: Communications, Interfaces, Applications, and Systems. Each layer is cleanly separated from the other layers.

The Communications layer defines basic types and methods for moving data in a *communications adaptor*. The adaptor also provides facilities for auto-generating code which is specific to different types of data (similar to CORBA), or using another method such as templating for the same purpose. Apart from basic types, such as Points, Time, and Vectors, the communications layer does not specifically define message types for data. This makes it easy to add new message types without having to modify any communications adaptor code.

The Interface layer describes the data which each type of robot capability actually requires. For an obstacle avoider, this may be the current state of the robot (stuck, moving, idle), and some commands (move to location, set max speed, set avoidance radius). These interfaces are described using XML which is then used to generate code which can be called from the Applications layer as well as any code which

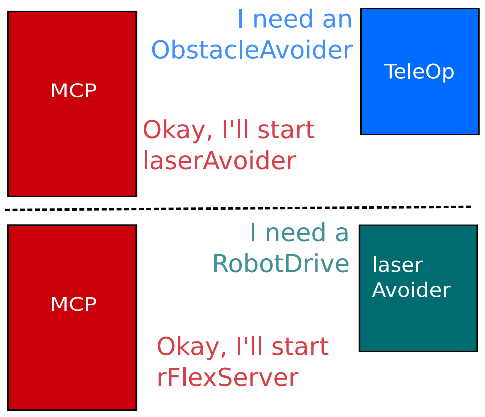

I need an ObstacleAvoider
TeleOp
MCP
Okay, I'll start laserAvoider
I need a RobotDrive
laser Avoider
MCP
Okay, I'll start rFlexServer

Figure 3: Module startup with MCP.

is necessary to work with the communications adaptor (such as marshalling data). These interfaces are independent of the communications layer.

The Applications layer is composed of a common API, providing a standard programming interface for controlling different aspects of individual applications, communications endpoints, and various useful utilities. Developers can implement new algorithms at this level, without being concerned about how data moves. Not only is the underlying communications protocol handled by the communications layer, but the process of finding the correct sources and sinks for data is handled by the systems layer. Developers implement the algorithm and specify what types of data are needed (i.e., an obstacle avoider requires a robot drive, a range sensor, and odometry), but not where it comes from.

Finally, the systems layer abstracts the details of individual applications away: the systems developer can specify how different applications connect to one another to provide required sources of information, but does not need to modify any source code. The connections are currently specified using XML configuration files. Current work on this layer is focused on building the application connections automatically using an optimization approach. One of the major features our systems layer provides is the ability to initialize a large number of robot applications without the need to write a startup script.

Modules

WURDE is designed to work with many different "modules" running as independent processes on one or multiple systems. Modules communicate asynchronously via the underlying communications protocol. The communications endpoints are connected by the MCP (Master Control Program) based on a layout described using an XML file and using information automatically harvested from the modules when the MCP is initialized. When a new module is started by the user, it first communicates with the MCP to determine where it should find the information it needs, and then enters the standard run loop. The MCP meanwhile initializes any additional modules that are required (see figure 3). These modules repeat the process until all requirements are fulfilled.

Figure 4: Microsoft's Age of Empires.

This mechanism allows the researcher to swap low-level modules while leaving high-level modules unchanged. We take advantage of this with a number of modules built to work with the iRobot B21r and ATRV Jr platforms, and the Player/Stage simulation environment(Gerkey *et al.* 2001). In addition, we have built a vision module to interface with Firewire cameras and the Intel OpenCV vision software (Intel Corporation 2006).

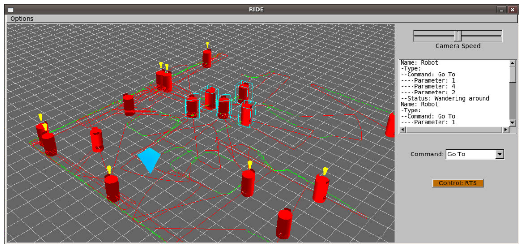
The RIDE Interface

Another module built with WURDE is the RIDE interface, our multi-robot tasking and control program. RIDE takes advantage of paradigms developed the computer gaming world. For many years now, there have been computer games which require the user to control a large number of diverse agents in a complex environment. One of the most popular styles of games in recent years has been the real-time strategy game.

Figure 4 shows a screenshot from Age of Empires II, a typical real-time strategy game(Microsoft Games 1999). The user can select one or more units on a central iconic, isometric view of the world. Units, terrain, and other features are shown in this part of the window. Once units have been selected, a list of possible tasks for the units is displayed below this window, depending on the type of units selected. Clicking on one of these tasks assigns it to all currently selected units. Other details about the units are also displayed below the main world view.

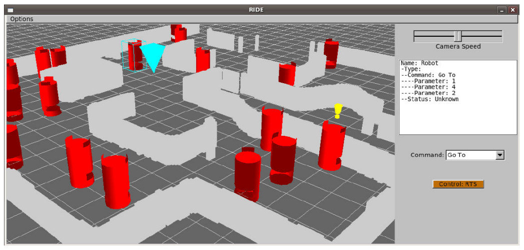
Figure 5 shows RIDE, the Robot Interactive Display Environment, our supervisory control interface for large teams of mobile robots.[1] The main window shows an iconic view of the world, just as in RTS games. Additional GUI elements allow control over the camera position, display information about the selected robots, and allow selected robots to be tasked by the operator, as in RTS games. Robots can also request help; when this happens, a yellow exclamation mark is placed over the robot. The human operator can then quickly see which robots may need to be re-tasked due to a problem.

[1]Although all of the functionality of the interface is in place, it is not yet "production quality". We are currently working on improving the look-and-feel of the GUI.



```
Name: Robot
-Type:
--Command: Go To
----Parameter: 1
----Parameter: 4
----Parameter: 2
--Status: Wandering around
Name: Robot
-Type:
--Command: Go To
----Parameter: 1
```

Figure 5: RIDE.



```
Name: Robot
-Type:
--Command: Go To
----Parameter: 1
----Parameter: 4
----Parameter: 2
--Status: Unknown
```

Figure 6: RIDE with occupancy grid.

One example of sensor visualizations can also be see in figure 5. The sensor displayed in this case is a SICK laser rangefinder with 180 radial distance measurements. In this display, if the difference between adjacent readings is small, they are joined by a green line, and otherwise a red line. This helps to highlight objects and walls, making it much easier for a human operator to interpret the sensor readings. This was a very simple but extremely effective innovation which helped improve operator understanding of the situation.

RIDE also displays sensor information and can render occupancy grids as walls in the world, as seen in figure 6. In the future, we will be able to texture map the current camera image from the robot onto the walls; this code is still in an early stage and not shown here. We plan to run user studies soon to confirm the effectiveness of RIDE and determine how to improve the interface.

AAAI 2006

In July 2006, we brought our robot to the AAAI Mobile Robot Competition and Exhibition in Boston, MA. This was our first "field test" of WURDE, and we encountered several challenges and unexpected issues over the course of the conference. Fortunately, we were able to work around them to demonstrate WURDE and RIDE, thanks in part to the structure of our software architecture. Our largest problem was with the obstacle avoider, which experienced some bugs leading to incorrect behavior. Due to the modularity of our architecture, we were able to split our team so that some people focused on debugging the obstacle avoider while others checked RIDE to make sure it was working correctly.

Certain aspects of RIDE turned out to be very successful. The laser display technique described earlier allowed robot operator to easily drive the robot in areas of the conference center where the operator had not been himself. Even without employing any mapping routines, and only using sensor displays, some operators were able to navigate the robot a significant distance and then return to the robot work area.

None of the difficulties we experienced were major problems and we were able to meet our goals to demonstrate WURDE and RIDE. While WURDE was relatively invisible (indeed, a correctly functioning architecture tends to go unnoticed) we were able to show off RIDE to various people at the conference using both our robot, Lewis, and in simulation using Stage. We received some very positive feedback from this process, and some suggestions to help improve the interface.

Conclusion

At the AAAI 2006 Mobile Robot Competition and Exhibition, we demonstrated our new robotics middleware and RTS-based control interface. We are continuing to develop these software packages and plan to release our source code in January 2007. Our middleware allows rapid development of robot applications by creating clean levels of abstraction so that the developer is not bogged down with irrelevent details at any stage of the development process. It is simple enough to allow inexperienced programmers to quickly grasp the basics while sacrificing very little flexibility.

The RIDE control interface allows a single user to control many robots easily, using a paradigm taken from the computer gaming world. Although we successfully demonstrated RIDE at AAAI 2006, there is still much to be done. The major job, currently underway, is improving the usability of the interface and adding more sensor visualizations, including learned maps. We are also adding more tasks to the the repertoire as we develop more capabilities in the WURDE framework.

References

Carnegie Mellon University. 2006a. Carmen Robot Navigation Toolkit. Web site: http://carmen.sourceforge.net/.

Carnegie Mellon University. 2006b. Inter Process Communication. Web site: http://www.cs.cmu.edu/ IPC/.

Côté, C.; Brosseau, Y.; Létourneau, D.; Raïevsky, C.; and Michaud, F. 2006. Robotic software integration using marie. *International Journal of Advanced Robotic Systems* 3(1):55–60.

Faust, J.; Simon, C.; and Smart, W. D. 2006. A video game-based mobile robot simulation environment. In *Proceedings of the International Conference on Robots and Systems (IROS)*.

Gerkey, B. P.; Vaughan, R. T.; Stoy, K.; Howard, A.; Sukhatmem, G. S.; and Matarić, M. J. 2001. Most valuable player: A robot device server for distributed control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1226–1231.

Intel Corporation. 2006. The Open Source Computer Vision Library. Web site: http://www.intel.com/technology/computing/opencv/.

Jones, H., and Snyder, M. 2001. Supervisory control of multiple robots based on a real-time strategy game interaction paradigm. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, 383–388.

Kramer, J., and Scheutz, M. 2006. Ade: A framework for robust complex robotic architectures. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*.

Microsoft Games. 1999. Age of Empires II: Age of Kings. Web site: http://www.microsoft.com/games/age2/.

Parasuraman, R.; Gastler, S.; and Miller, C. 2003. Human control of multiple robots in the RoboFlag simulation environment. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, 3232–3237.

Wang, J.; Lewis, M.; and Gennari, J. 2003. USAR: A game based simulation for teleoperation. In *Proceedings of the 47th Annual Meeting if the Human Factors and Ergonomics Society*.