Proc. 2002 Australasian Conference on Robotics and Automation Auckland, 27-29 November 2002

Design of a Web-Based Tele-Programming Interface for Mobile Robots

David Austin

Robotic Systems Laboratory,
Research School of Information Sciences and Engineering,
Australian National University, ACT 0200, Australia.
david@syseng.anu.edu.au
robot.anu.edu.au

Abstract

Owning and maintaining a mobile robot is fairly expensive and is beyond the reach of the vast majority of people. This paper presents the design of an interface which allows remote users to develop and execute programs which control the mobile robot at the Australian National University. In addition to allowing greater public access to a mobile robotics, this project promotes scientific exchange by allowing mobile robotics researchers worldwide to experiment with the ANU robot. This paper gives an overview of the long-term autonomous operation system, which is a basic requirement, and then presents the web tele-programming interface design and initial steps towards implementation.

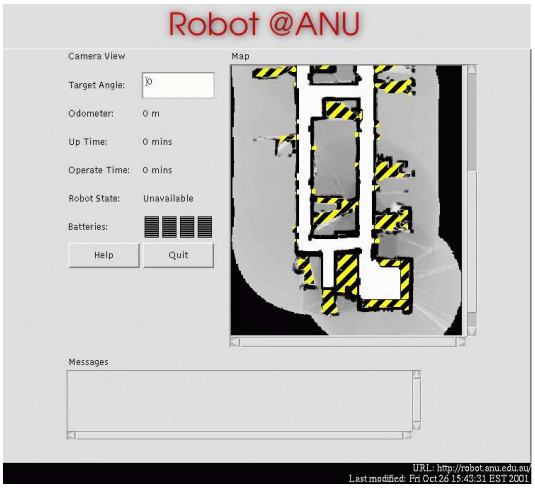

Figure 1: ANU Teleoperation Interface

1 Introduction

As part of recent research at the Australian National University, we have been developing a mobile robot capable of long-term continuous and fully autonomous operation in an unstructured office environment. The robot has a robust software system with redundant modules for critical functions (e.g. localisation) and a recharging station with which it can dock and autonomously recharge itself. This mobile robot provides a significant new opportunities which have not been commonly available previously. So far, we have used a web-based interface to make the robot available for teleoperation whenever the robot is not used for research (see Figure 1). However, teleoperation is a limited form of interaction, and rapidly becomes boring. To provide another, richer form of interface we are developing a web-based interface which allows remote users to write programs to control our robot: *tele-programming*.

The tele-programming interface makes it much easier for people interested in robotics to start out and get a feeling for the particular excitement and challenges of the field. We see the tele-programming interface as a potential focal point for mobile robot software development and experimentation. The web-based tele-programming interface also has possibilities for use in the classroom, allowing students to use a mobile robot without the requirement that each school purchase, set up and maintain a robot. Finally, the tele-programming interface can be used for research experiments and is particularly interesting for testing the robustness of systems to changes in environment and sensor placement.

This paper presents a new web-based tele-programming interface for the Australian National University XR4000 mobile robot. Section 2 presents an overview of the mobile robot system, including the recharging and redundant localisation modules. Then the tele-programming interface design and implementation is presented in Section 3, with security details discussed in Section 4. Section 4.1 presents an example of the use of the interface and finally, we conclude with a discussion in Section 5.

Figure 2: Recharging station

2 System Overview

The system consists of a Nomadic Technologies XR4000 mobile robot, equipped with two 800MHz Pentium III CPUs, 48 Polaroid sonars, a SICK laser scanner, a panoramic camera, an active stereo vision head [?], and a web camera. The robot is controlled by a modular software system, much of which has been released as open source software[Austin, 2002]. The most important aspect of the system for this paper is that it allows the robot to operate continuously and fully autonomously. Two key components that enable this are the autonomous recharging system and the redundant localisation modules.

2.1 Recharging

One of the important aspects for autonomous operation of a mobile robot is its power source. Most mobile robots today use batteries to provide power both for motion and for computation. Therefore, the management of the energy supply of the robot is a crucial function for long-term operations. The robot must monitor the state of the batteries and periodically recharge them. There are many possible ways to implement recharging. Figure 2 shows the recharging station that has been built at the ANU (for less than $100). The recharging system for our robot has been implemented as follows: at the top there is an infrared beacon which is used to locate the recharger from a distance; once the robot is fairly close to the recharger, the robot servos using the laser scanner and the reference grid in the middle of the recharging station; finally, the robot docks with the power plug just below the grid (see Figure 2).

We have added a battery balancing circuit [Austin and Fletcher, 2001] because charging the batteries in series led to diverging battery voltages (this is a common problem with Nomadic's XR4000s).

2.2 Redundant Localisation

Localisation of a mobile robots in indoor environments has been studied for some time. Our system uses a Kalman filter localisation method (based on [Jensfelt and Christensen, 2001]) as the primary localiser because of its low computational overhead. This method uses a line segment map of the world and determines the pose of the robot by matching line segments extracted from laser range scanner data.

In addition to the simple method above, a particle filter based global localisation method [Jensfelt *et al.*, 2000] is used initially and in the event that the robot becomes lost. This method requires considerably more computation and so is not used all of the time. Clearly, running multiple localisation algorithms in parallel will increase reliability and provide improved detection of failures.

3 Tele-Programming Interface

Clearly, for a tele-programming interface, the basic requirements are that users are able to upload source files, compile them, execute them and get feedback from the execution. The web-based interface for file management, including upload and compilation is fairly straightforward and can be implemented in PHP easily. Clearly, it is important to make the user interface as easy to use as possible, but there are no technical challenges in the implementation. The web site has been designed as shown in Figure 3. The first step for a user is to register. Once registered, users can log in to the user home page. Here there is access to their modules through the module management page, as well access to any sensor data that has been generated by execution of their modules.

The most challenging part of the tele-programming interface is the execution environment where user code is run. The execution environment must make available interfaces through which the user program can receive sensor data, as well as an interface for controlling the robot. The programming environment matches the software system used to control our robot: the DROS system [Austin, 2002], which is open source software. DROS supports modular programming with inter-module communications using TCP and Unix sockets (with further communications methods planned). The message formats are defined in a language independent way, so that it is possible to interact with the DROS system using any language. The DROS system has been implemented in C/C++ so there is strongest support for those languages. In addition, Java support is under development.

The interfaces to the robot's sensors are to be provided by separate programs running within the execution environment. These programs appear identical to

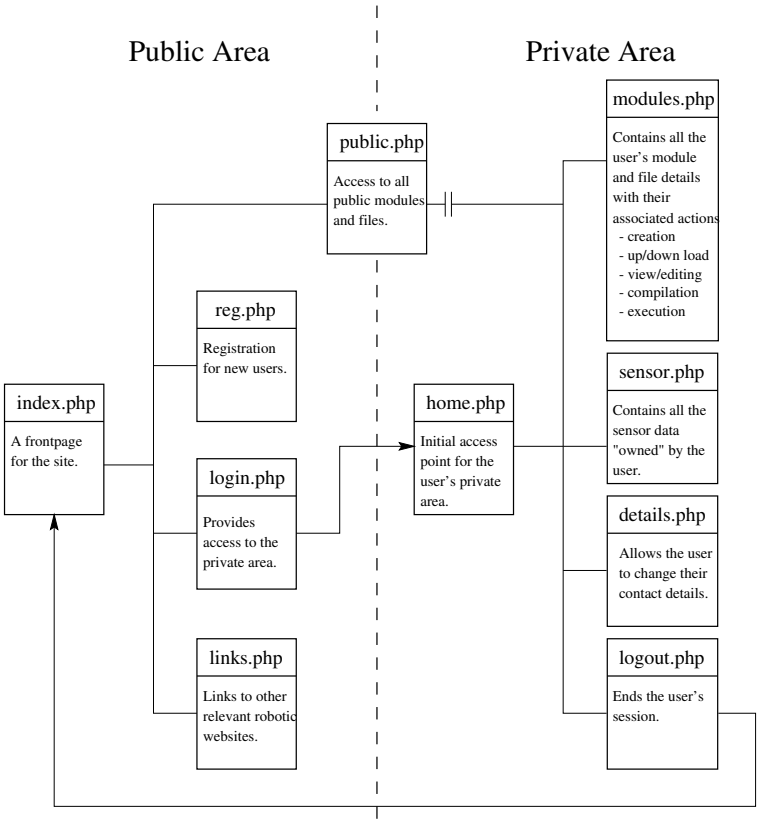

Figure 3: Web Site Design

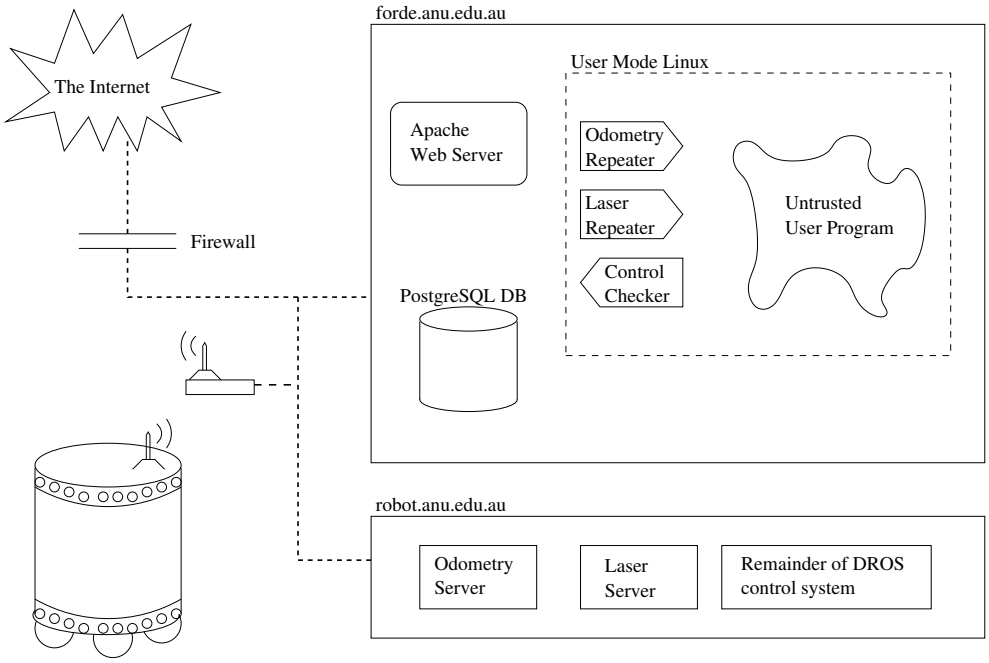

Figure 4: Execution Environment for Untrusted User Programs

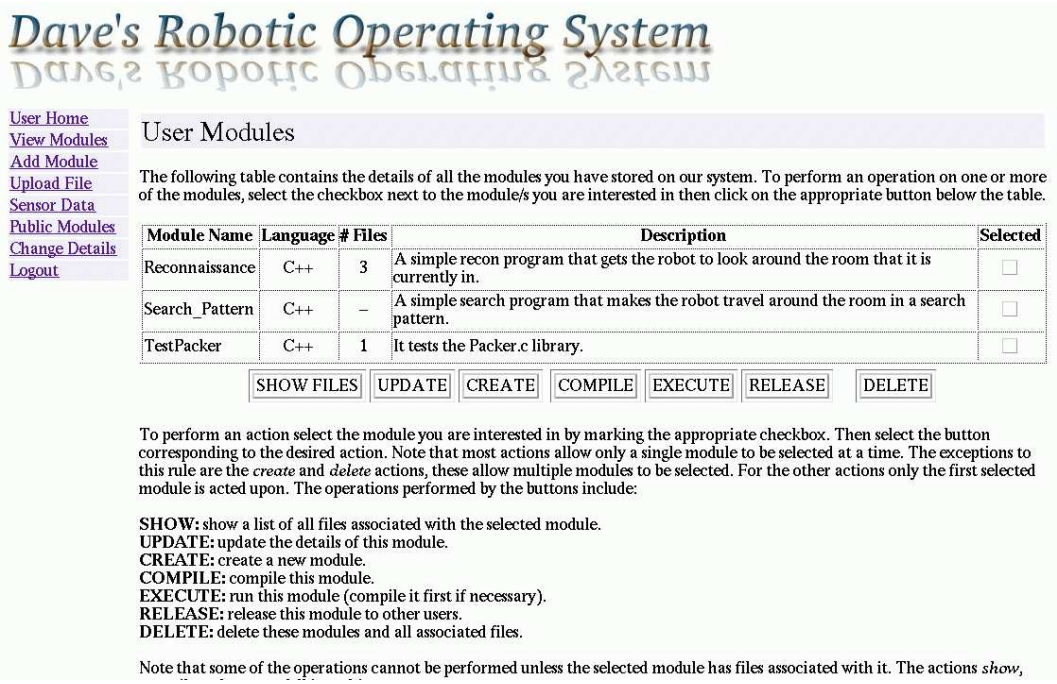

Figure 5: User Module Management Page

the servers on the actual robot but will provide a level of insulation from the robot (discussed further in the next section). The control interface will be at a high level where user programs issue commands to move to a given target point. Target points can easily be verified against a set of allowable target points (described by polygons in this case). In the future, we plan to allow low-level control of the robot, by permitting user programs to issue velocity commands. The velocity commands will be checked prior to being issued to the robot, to ensure that collision will not occur. The code to do this already exists as part of the Dynamic Window Approach [Fox *et al.*, 1997] collision avoidance algorithm that is being used for local obstacle avoidance. During program execution, the user can monitor the progress of the robot using the teleoperation interface shown in Figure 1.

4 Security

Clearly, security is an important consideration for the tele-programming interface since we are basically allowing users to upload any (untrusted) code and execute it on our computer. Security is provided by running the untrusted code within a virtual machine or *sandbox*: a Linux system executing as a user process of another Linux system. This software is called User-Mode Linux[Dike, 2002]. User-Mode Linux is configured so that the untrusted code cannot make network connections outside of the sandbox. Bandwidth-limited connections to the robot's sensors are made available within the user-mode Linux and rigorous safety checking is applied the control commands coming from the untrusted code, before sending them to the robot. It is important to limit the bandwidth available to user programs to prevent denial-of-service type attacks which could saturate the radio link to the robot.

4.1 Teleprogramming Example

1. **Log in** The teleprogramming interface requires that users log in first, so that their source code files and sensor data can be managed.
2. **Module Management** Figure 5 shows the module management page, with multiple options available for the management of user programs (or modules).

   Figure 6 shows an example user program. This program connects to the laser server (lines 21, 22 and 26) requesting laser data at $1Hz$. A callback function is added to be called whenever new data arrives on line 24. Finally, the event loop is entered on line 31. The event loop waits for an event (in this case new laser data) and calls handlers, in this case the `notify` function on lines 9-17. The `notify` function prints out the laser range in front of the robot (reading number 180). Once this has happened 10 times, the event loop is terminated with the `FinishEventLoop` call on line 16.
3. **Module Compilation** Sample output for compilation (actually compilation failure) is shown in Figure 7.
4. **Module Execution** Figure 8 shows the results of running the program shown in Figure 6 using the teleprogramming interface.

```
#include <iostream.h>
#include "LaserProxy.h"
#include "GCEventLoop.h"
#include "ComponentBase.h"
#include "DROSDefines.h"

int count = 0;

void notify(LaserProxy *lp, long id,
         int changes, GCEventLoop *loop) {
  cout << "Front range = " << lp->r(180) << endl;
  proxy->ClearChanges(id);

  count++;
  if (count >= 10)
    loop->FinishEventLoop();
}

int main() {
  GCEventLoop loop;
  LaserProxy lp("LaserServer", LASER_SERVER_LASER_SCAN,
                1.0 /* seconds */);
  loop.AddToComponentList(&lp);

  lp.AddNotify((NotifyCallbackType) notify,
               LASER_PROXY_NEW_DATA, &loop);

  if (loop.EstablishConnections() < 0) {
    cerr << "Failed to EstablishConnections\n";
    return -1;
  }

  loop.EventLoop();
  return 0;
}
```

Figure 6: LaserFrontDist: Example program which processes laser data.

5. Sensor Data Retrieval

Not shown are options for editing a file using a HTML form (for simple edits) or forms for uploading new copies of files (e.g. after major edits, using favourite text editor). Also note that the teleoperation interface shown in Figure 1 is used to monitor the execution of programs on the robot.

5 Discussion

This
paper has presented a web-based tele-programming interface for mobile robots. The major components are an autonomous, continuously operating mobile robot, a PHP website with database, and a secure execution environment. With these components, an interface is created which allows remote programming of the ANU XR4000 mobile robot. We see great potential for this interface as an educational resource and a focal point for development of robot software.

Future work includes improvements to the user interface, based on user feedback. Also planned is a low-level interface to allow user programs to issue velocity commands to the robot. Clearly, this low-level interface must perform more checks, based on a range of different sensors.

Acknowledgements

The author would like to acknowledge and thank Ian Sainsbery for his assistance in developing the system.

References

[Austin and Fletcher, 2001] David J. Austin and Luke Fletcher. Modifications made to the anu xr4000. http://forde.anu.edu.au/~david/Nomadics/XR4000/modifications.html, 2001.

[Austin, 2002] David Austin. Dave's robotic operating system, 2002. http://www.dros.com/.

[Dike, 2002] Jeff Dike. The user-mode linux kernel home page, 2002. http://user-mode-linux.sourceforge.net/.

[Fox *et al.*, 1997] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1), 1997.

[Jensfelt and Christensen, 2001] Patric Jensfelt and Henrik I. Christensen. Pose tracking using laser scanning and minimalistic environmental models. *IEEE Transactions on Robotics and Automation*, 17(2):138–147, April 2001.

[Jensfelt *et al.*, 2000] Patric Jensfelt, David J. Austin, Olle Wijk, and Magnus Andersson. Feature based condensation for mobile robot localization. In *IEEE Intl. Conf. on Robotics and Automation*, 2000.

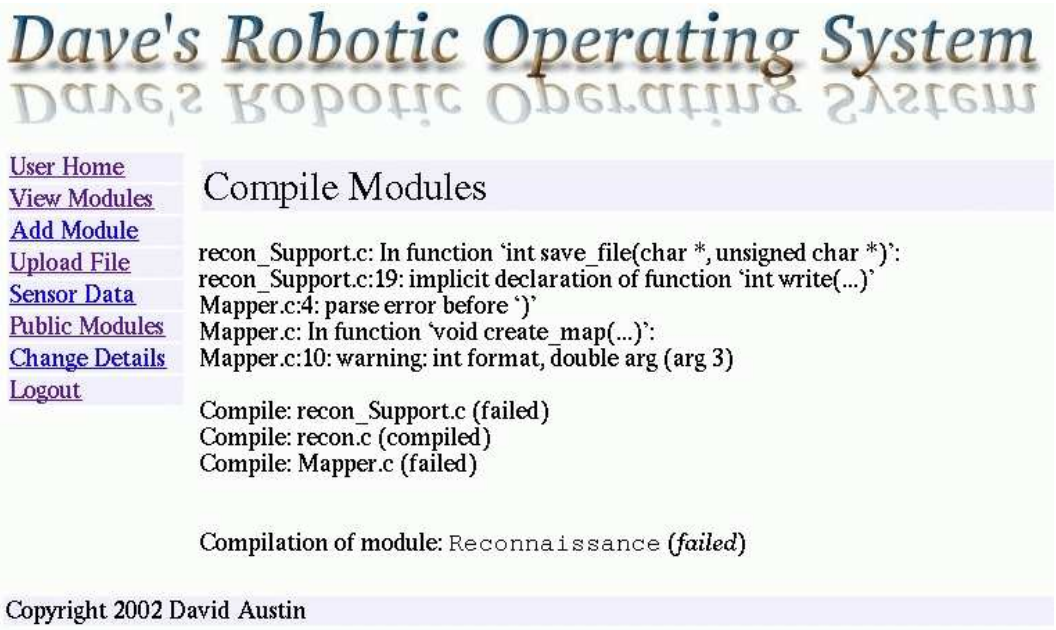

Figure 7: Module Compilation

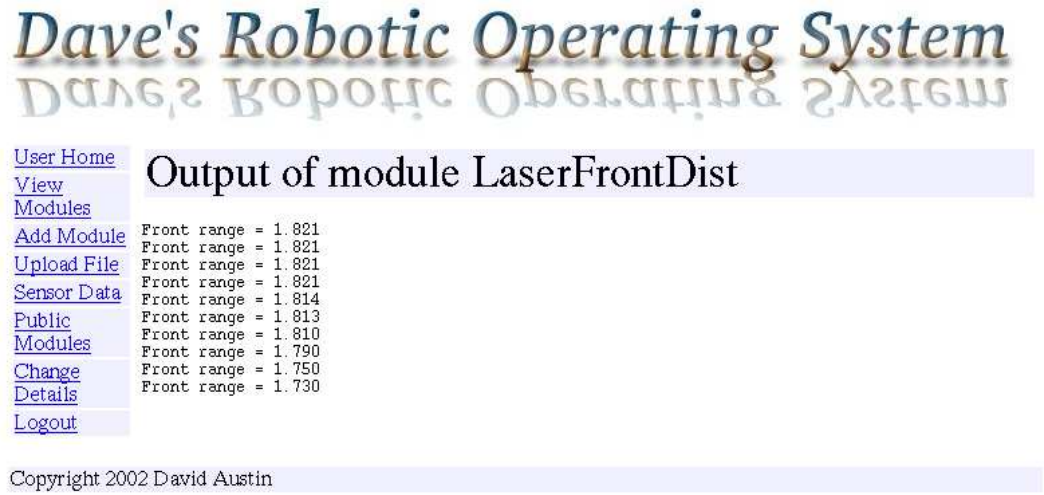

Figure 8: Results of LaserFrontDist Module Execution