

Robust, Long Term Navigation of a Mobile Robot

David Austin and Kirill Kouzoubov

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

Abstract

In order for the field of robotics to become widely accepted, it is necessary to address the ongoing problem of reliability. For too long now, the robotics community has neglected robustness and reliability, focusing instead on theoretical advances. This paper presents work undertaken at the Australian National University to develop a mobile robot which operates 24 hours a day, 7 days a week without human intervention. The design of our system is presented, with emphasis on the aspects which improve the robustness of the system. In addition, the results from our experiments are presented which illustrate the reliability levels that we have achieved and highlight the areas where further work is required.

1 Introduction

Mobile robotics is today reaching the point where deployment into real-world situations seems possible. Enabling technologies such as path planning, localisation and obstacle avoidance have all been proven in laboratory situations. However, long-term experiments with mobile robots are still quite rare. This paper describes the mobile robot and the framework for long-term experiments that is being established at the Australian National University.

To date, only a few researchers have presented their experiences with and approaches to long-term mobile robotics. The robot Xavier [1] has been in operation for a number of years, available for teleoperation on the web. However, Xavier possesses no automatic recharging system and therefore was not available continuously. Since the batteries were charged manually, some supervision of Xavier's operation is required. Two generations

of museum tour guides developed by Burgard *et al.* [2] and Thrun *et al.* [3, 4] both demonstrated longer term operations, online for a total of 32 hours 18 minutes and 94 hours and 23 minutes, respectively. Again, these systems required manual battery charging and so had a degree of supervision. In Japan, Yuta and Hada [5, 6] have started a project to develop an automatic recharging system but have yet to demonstrate integrated localisation and navigation capabilities. Our goal is to develop a system that can run 24 hours a day, 7 days a week for up to a year with no supervision. This goal requires modification of the hardware of the robot, development of a software system suited for long-term operation as well as reliable sensing and control algorithms.

This paper presents our initial steps towards the development of a reliable mobile robot system which can operate autonomously for long periods in real-world environments. Section 2 presents an overview of the hardware and software components of our robot. In Section 3, we discuss the problems of batteries, power management and recharging. Section 4 outlines our localisation system and Section 5 describes the navigation algorithm. In Section 8, we present the results of our long term experiments.

2 System Overview

The hardware and software systems required to build and run a mobile robot are extremely varied and complex. Clearly, we, the robotics community, must develop methods to manage this complexity or we will be unable to create reliable and scalable robot systems. Experimentation on real-world robot systems is required to gain the experience and insight to address the issue of complexity. Our robot runs DROS[7], an open source mobile

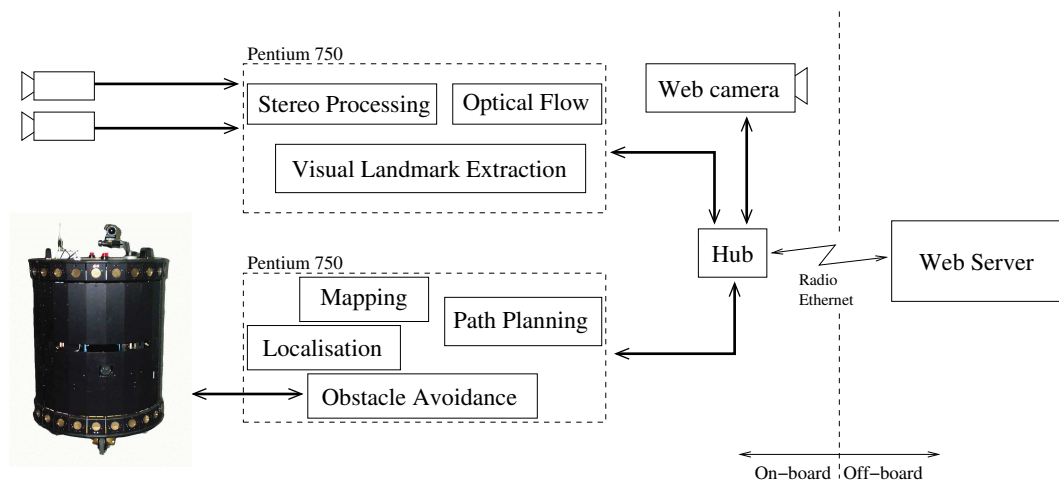


Figure 1: Overview of the hardware and software components that are used for the mobile robot

robot operating system that we have developed. An explicit goal for DROS is to simplify the tasks of modular programming. Modular programming is typical in research environments, where students work on separate aspects of the system. In addition, the hope is that multiple modules can be developed for each important robotic task to promote error recovery. DROS consists of over 100,000 lines of code, with more in development.

The system developed for our mobile robot is shown in Figure 1. The robot contains two on-board computers, both 750MHz Pentium CPUs. The lower CPU runs the navigation software, consisting of localisation, path planning and obstacle avoidance modules. In addition, the mapping process runs on the lower computer. The upper CPU has video capture cards and is used to process the incoming video data. At present, no use is made of vision information though work is ongoing on vision-based SLAM and visual-servoing for recharger docking.

3 Batteries and 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 imple-

mented as follows: at the top there is an infrared beacon which can be used to locate the recharger from a distance (this is not presently used as it is not needed); 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).

The recharging station consists of the power plug and the target for the robot's laser sensor (see Figure 2). Our XR4000 has on-board charging power supply which is connected to the AC 240V power supply by a standard IEC plug, the type used to power computers. IEC plugs are quite good in that they are proven to be safe and being a standard are cheap and easy to get, however millimeter precision is required to plug an IEC connector in to the socket. Our XR4000 robot is capable of performing sideways moves with millimeter accuracy, and the algorithm developed by Seungjun Oh [?] used this feature of the robot to perform docking successfully. However due to a firmware bug in the robot's controller the algorithm would stop working (small movements cease working) approximately 24 hours after rebooting the robot

To overcome this problem we decided to develop a docking system which does not rely on the sideways moves and only uses a car-like model for motion, with one exception: on-the-spot rotation is allowed. Initially we were hoping to achieve millimeter accuracy, but we failed in that and had to redesign the plug.

The solution we came up with is as follows: if



Figure 2: Recharging station

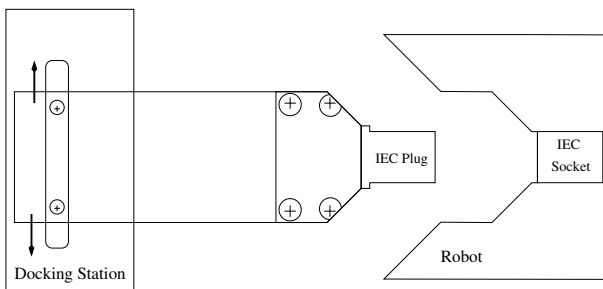


Figure 3: Plug and Socket

you can not position the robot accurately enough relative to the plug, make plug slide into the right spot instead. So we mounted the connector on a compliant platform (see Figure 3), which can move sideways 10 mm right and left from the centre position, change it's orientation by few degrees, it is also spring loaded and can be pushed towards the recharging station by 20 mm, this helps to reduce the impact during docking. This connector design allows for slight misalignment of the robot during the docking, increasing the reliability. When robot is docking to the recharging station it forces the plug in to the right position by sliding it left or right from the centre.

The decision to use this movable platform rather than designing a new type of the connector was mainly due to safety concerns. As there is high voltage involved the plug would have to be human-proof. It seemed that designing a platform that can be aligned during docking was an easier and safer solution in our case.

The target we use to identify the recharging station is a grid which generates a highly distinct pattern when "viewed" by the laser range sensor. We have reused the grid developed by Seungjun Oh [?] It consists of the backplate, 50 cm in length, and four 3 cm wide strips positioned 8 cm in front of the backplate. The strips are equally spaced with 3 cm gaps between them. The central gap is located right above the centre position of the plug. All strips and the backplate have matte surface to reduce reflection and hence improve range sensor accuracy.

The docking process consists of several steps:

1. **Locate the recharging station.**
2. **Plan Path.** The path is an arc starting at the current location and ending 10 cm in front of the plug, such that tangent at the end point is parallel to the plug.
3. **Initial Rotate.** Rotate on the spot, so that robot is on the tangent to the path planned.
4. **Follow the Arc.** Move along the arc, constantly reassessing your location relative to the recharging station using output of the particle filter.
5. **Docking.** If safe move straight forward with a slow velocity (10 mm/s).

We use laser range sensor data fed to the particle filter algorithm [?] to constantly reassess the

robot's position and orientation relative to the recharging station. Laser data arrives on average three times a second, which is a fairly low frequency, so odometry readings are used in between the laser scans.

We have added a battery balancing circuit [8] because charging the batteries in series led to diverging battery voltages (this is a common problem with Nomadics XR4000's). In addition, we have replaced all of the power wiring to the CPUs. We believe that inadequate power wiring was causing the CPUs to hang intermittently.

4 Localisation

Localisation of a mobile robot in indoor environments has been studied for some time. Our system uses a Kalman filter localisation method (based on [9]) 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 [10] 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 further localisation algorithms in parallel will further improve the reliability and improve detection of failures. However, the biggest issue in the localisation is that the environment changes over time. These changes cause the map to become less accurate and the error rate of localisation to increase. This area is the subject of further research.

5 Navigation

The navigation system used is based on the Dynamic Window Approach [11]. Sensor data from the laser range scanner and the sonars is integrated in an occupancy grid and then the robot path is planned in the grid to optimise an objective function. The objective function assesses the heading towards the goal, the velocity of the robot and the distance of the trajectory from collisions. An occupancy grid is used at the low level to allow the integration of multiple sensors, clearly important for robust operation. At the top level, the path is planned in a topological, graph-based map. The navigation system detects when the robot gets stuck and re-plans alternate routes. Statistics

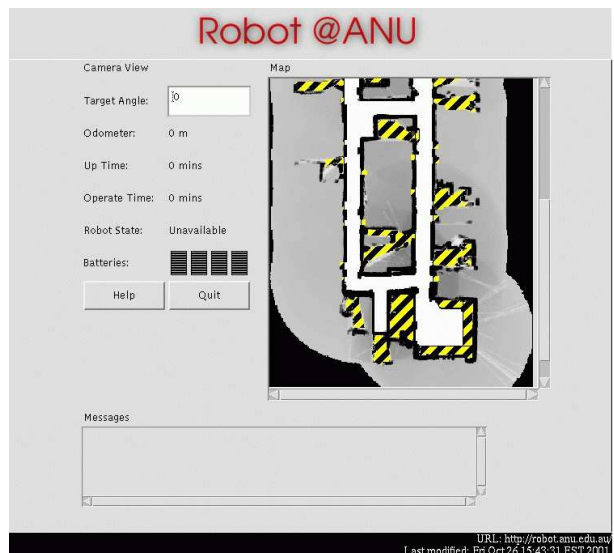


Figure 4: Web-based teleoperation interface

for each arc of the graph are maintained indicating the probability of traversal. These statistics are used in the path planning process.

6 Teleoperation Interface

One of the requirements for long-term experimentation with a mobile robot is a task or series of tasks to conduct which will take considerable time. An option is to just generate random points within our building and send the robot to them. Instead, we have chosen to make the robot available on the web for remote users. Figure 4 shows the user interface that is presented. Remote users can click on a map of the building, sending the robot to those locations.

7 Teleprogramming Interface

In addition to the teleoperation interface, we are in the process of implementing a web-based interface to allow remote users to write programs to control our robot. The teleprogramming interface allows users to upload source code files, compile them into a module and execute the module to control the robot. The programming environment is the same open-source DROS system [7] used for the rest of our robot control system. The user program cannot be trusted and is executed in a sandbox environment with a User Mode Linux virtual machine. The user program can access the sensor data from the robot via bandwidth limited connections. The control commands from the user program are sent to a checking module which en-

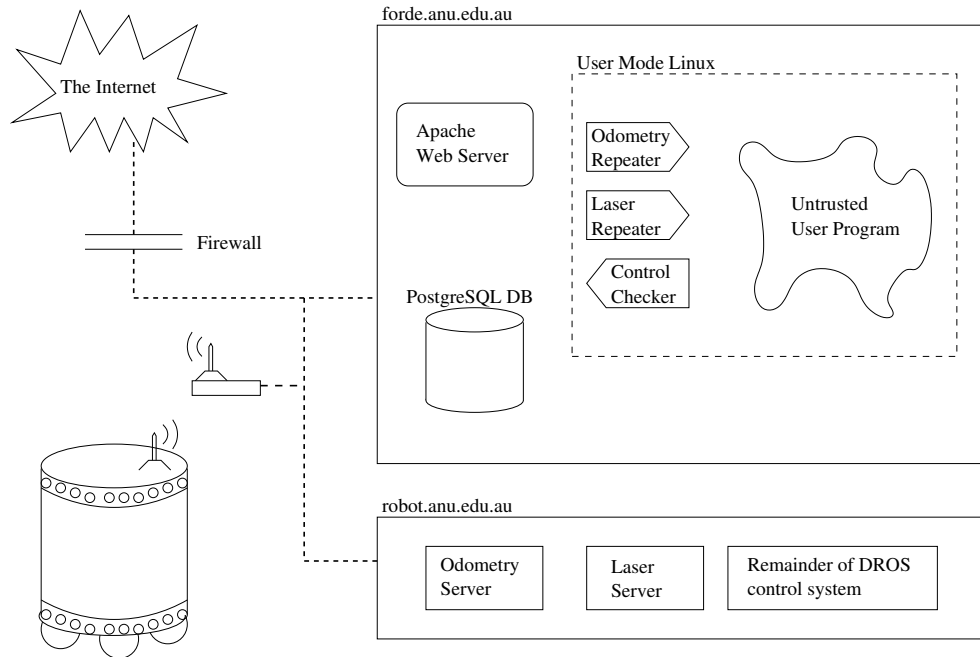


Figure 5: Execution Environment for Untrusted User Programs

sure that the commands are safe before forwarding them to the robot. Figures 6 and 7 show examples of compilation and execution, respectively.

8 Results

In an experiment with the previous software system, the robot logged 140 hours of operation and traveled more than 25km. However, as can be seen in Figure 8, the robot required some supervision and manual correction of errors. The first, and most significant problem was that the old recharging system used a simple IEC plug with no compliance. This required 1-2mm positioning accuracy for successful docking. Unfortunately, due to a firmware bug in the motor control microcontroller, small position-based motion commands fail after 24 hours. It was found that this could be delayed to 48 hours by increasing the velocity for small motions. However, this solution is clearly undesirable, and so we implemented a new recharging system with a compliant plug, as described in Section 3 above.

The second significant type of failure was localisation failure. Clearly, failure of this type is very significant since the robot cannot find its way back to the recharging station if it is lost. Our solution to this problem is to implement redundant localisation methods, and in particular, to implement a global localisation method that can recover in the

event that the robot gets lost. This has yet to be tested in a long-term experiment.

The “Bad Goal Position” failure shown in Figure 8 is an example of what happens when software does not include error checking for failure cases. In this case, a user of the teleoperation interface was able to send the robot to a point that it could not reach. The code for handling this case had not been fully debugged.

9 Discussion

From our efforts to build a robust, autonomous mobile robot the following conclusions can be drawn:

- There can never be too much error checking. When it comes to reliability, software systems scale poorly. It is necessary to spend considerable effort on handling all possible failure modes so that each software module is as robust as possible.
- Wherever possible, tasks should be engineered to make them easier. Our original solution to the recharger docking problem, while elegant required too much accuracy from the robot and hence was not robust. Modification of the recharging station is relatively easy and should be done to simplify the docking task. However, it is usually not ac-

Dave's Robotic Operating System

[User Home](#)
[View Modules](#)
[Add Module](#)
[Upload File](#)
[Sensor Data](#)
[Public Modules](#)
[Change Details](#)
[Logout](#)

Compile Modules

```
recon_Support.c: In function 'int save_file(char *, unsigned char *)':  
recon_Support.c:19: implicit declaration of function 'int write(...)'  
Mapper.c:4: parse error before `)'  
Mapper.c: In function 'void create_map(...)':  
Mapper.c:10: warning: int format, double arg (arg 3)
```

```
Compile: recon_Support.c (failed)  
Compile: recon.c (compiled)  
Compile: Mapper.c (failed)
```

Compilation of module: Reconnaissance (*failed*)

Copyright 2002 David Austin

Figure 6: Module Compilation

Dave's Robotic Operating System

[User Home](#)
[View Modules](#)
[Add Module](#)
[Upload File](#)
[Sensor Data](#)
[Public Modules](#)
[Change Details](#)
[Logout](#)

Execute module

Output of module: Reconnaissance

```
Reconisance>> ---- Reconnaissance ----  
Reconisance>>  
Reconisance>> Performing scan.....complete  
Reconisance>> Constructing model...complete  
Reconisance>> Saving data.....complete  
Reconisance>>  
Reconisance>> Data file: recon01.dat
```

Execution of module: Reconnaissance *successful*.

Copyright 2002 David Austin

Figure 7: Module Execution

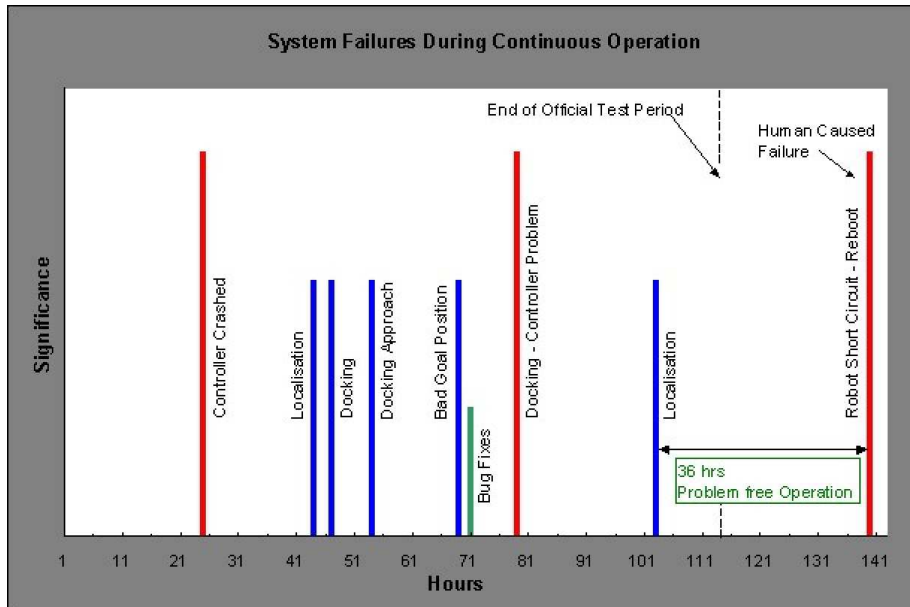


Figure 8: Results of long-term experiments

ceptable to engineer the environment to simplify the localisation task since this would require significant changes and effort.

- Critical functions for the system should be made redundant, as far as possible. This clearly applies to localisation, and all reliable mobile robots should have at least two localisation schemes, preferably with multiple sensors and multiple sensor modalities. With this in mind, we are investigating vision-based localisation because our present schemes both depend heavily on the SICK laser scanner. For safety, obstacle avoidance should also be implemented with redundant modules.

One of the most frustrating aspects of developing a robust mobile robot is discovering defects in supplied, closed systems for which a work-around must be found. In this regard, mobile robot and sensor manufacturers still have much to learn.

1. KISS: Keep It Simple Stupid. A fully functional mobile robot system will be a complex system. Manufacturers should strive to avoid hardware complexity. The XR4000 has 6 micro-controllers, 2 CPUs, plus any additional sensors.
2. Manufacturers should not attempt to build an API on top of their system. Instead expose

(and fully document) the communication interface. We have spent considerable effort reverse-engineering the XR4000 server[12] to get access to the robot at a lower level, removing one layer of delay (which added no useful functionality). Writing a robot software system is a very difficult task with real-time constraints, multiple sensors and multiple modules executing simultaneously.

3. Time-stamping of sensory information is crucial. It is necessary to be able to compute timestamps in host CPU time.
4. Communications protocols should be robust, capable of rejecting errors and resuming communication. The SICK laser scanner protocol is an example of a bad implementation. The start of a packet is marked with a 0x02 byte. However, this byte is allowed to occur in the remainder of the packet, so that when a communication error occurs, it is non-trivial to get synchronised again. Furthermore, the synchronisation can take several packets, or over a second at standard data rates. Escaping of 0x02 bytes in the packet body would avoid this problem and make the sensor much easier to deal with.
5. Power supplies should be designed and tested for the task. Mobile robot control systems will often consume large amounts of CPU

power (and hence large amounts of electrical power). Manufacturers need to account for this. Furthermore, research robots need additional capacity for the sensors and CPUs

Nomadic Technologies may be a particularly bad example¹ with most of the above problems, however, our experience with other robots and sensors is also far from perfect.

Acknowledgements

The author would like to acknowledge and thank Kirill Kouzoubov, Ian Sainsbery, Peter Brown, Haydn Lowe, Colin Thomsen, and Seungjun Oh for their assistance in developing the system.

References

- [1] R. G. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, J. O'Sullivan, and M. M. Veloso, "Xavier: Experience with a layered robot architecture." To appear in the ACM magazine *Intelligence*, edited by Manuela M. Veloso.
- [2] W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Proc. of the National Conference on Artificial Intelligence*, 1998.
- [3] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Minerva: A second generation mobile tour-guide robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.
- [4] W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial Intelligence*, vol. 114, no. 1-2, 2000.
- [5] S. Yuta and Y. Hada, "Long term activity of the autonomous robot - proposal of a benchmark problem for the autonomy," in *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, pp. 1871–1878, Oct. 1998.
- [6] Y. Hada and S. Yuta, "Robust navigation and battery re-charging system for long term activity of autonomous mobile robot," in *The 9th International Conference on Advanced Robotics('99 ICAR)*, pp. 297–302, Oct. 1999.
- [7] D. Austin, "Dave's robotic operating system." <http://www.dros.com/>.
- [8] D. J. Austin and L. Fletcher, "Modifications made to the anu xr4000." <http://forde.anu.edu.au/~david/Nomadics/XR4000/modifications.html>, 2001.
- [9] P. Jensfelt and H. I. Christensen, "Pose tracking using laser scanning and minimalistic environmental models," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 138–147, Apr. 2001.
- [10] P. Jensfelt, D. J. Austin, O. Wijk, and M. Andersson, "Feature based condensation for mobile robot localization," in *IEEE Intl. Conf. on Robotics and Automation*, 2000.
- [11] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, 1997.
- [12] D. Austin, "Drobot - nomadics robot software and hardware support." <http://drobot.sourceforge.net/>.

¹Certainly, market forces seem to have acted in the case of Nomadic Technologies!