Software architecture and simulation tools for autonomous mobile robots [1]

G.D. van Albada, J.M. Lagerberg, B.J.A. Kröse
Department of Computer Systems
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam

Abstract

In the development of software for experimental autonomous robotic vehicles various problems have to be solved.

Firstly, as the systems are experimental, frequent changes in the sensor configuration, the computer hardware and the control software must be easily incorporated. This requires a modular software design. Secondly, it is frequently necessary to test new software, new algorithms and even entirely new control paradigms. This is facilitated by the use of suitable simulation software. Thirdly, the software developed for one vehicle should be easily portable to vehicles of a somewhat different mechanical design. A common control level for all vehicles makes this possible.

In this paper we will address the solutions that we have adopted within the ESPRIT II project MARIE[2], starting with a presentation of the low level control structure adopted for our vehicle. The emphasis of our paper will be on the extensive simulation system that has been developed to support our software development efforts. Important aspects of this simulation system are the simulation of the response of the vehicle to the commands generated by the software, the simulation of the response of the sensors as the vehicle moves through the environment, and the user friendly graphical interface.

The simulation software allows the user to construct a complex environment and to find the output that would be produced by various types of sensors, such as ultrasonic range finders, as the robot drives through this environment. The graphical interface not only shows the progress of the vehicle, but also indicates how obstacles are seen by the sensor system.

[1] This paper describes work done for the ESPRIT II project 2043: "Mobile Autonomous Robot in Industrial Environment" (MARIE).
It was published in "Robotic Systems," Vol 10, "Advanced techniques and applications," Spyros G. Tzafestas (editor), Kluwer Academic Publishers, p. 495-503 (1992)
Presented at Euriscon '91 (Corfu)

[2] The other partners participating in the MARIE project are VOLMAC (prime), Robert Bosch GmbH, Framatome, Framentec, Hitec, IAI, Indecon, University of Strathclyde.

Introduction

One of the central themes in the research effort at our institute is the achievement of autonomous behaviour by robotic systems. We study both stationary robot arms and robot vehicles, both in the context of nationally funded projects and ESPRIT projects. In this paper we describe aspects of the software architecture and the simulation tools that we utilize in building an autonomous mobile robot. This approach was developed mainly in cooperation with other partners in the ESPRIT project MARIE.

The objective of the work described here is the achievement of a software environment and design philosophy that facilitate the rapid design, implementation and testing of sensor data processing and of high and low level control modules for mobile robots.

The first issue that we have to address is that of software flexibility. Various approaches to the problems of task execution, path planning, navigation and collision avoidance must be tested without completely redesigning or rebuilding the control software for the cart. This problem is generally solved by using a modular design, with well defined interfaces. The data representations within each module are hidden from the other modules.

The second issue is that of portability. To this date almost all mobile robots have a unique hardware and software architecture. Yet, it is desirable to build the control software in such a manner that it can easily be ported between various vehicles. The solution in this case is the design and implementation of an appropriate generic control layer or virtual machine having an "instruction set" that is independent of the underlying hardware.

The third issue concerns the initial testing of the software for the cart and predictions of the effects of modifications in the software and hardware. Furthermore, the effects of system failures etc. must be easily and safely testable. A suitable simulation environment greatly facilitates these procedures.

In this paper we will first describe the computing environment that we use. Next we will touch upon the general structure of our control software and describe the manner in which we strive to ensure modularity and portability. Subsequently, we will describe the simulation software packages that have been developed and our experience with the actual implementation and use of the software.

The computing environment

Both within the MARIE project and the Department of Computer Systems of the University of Amsterdam we strive for standardization of our computing equipment and operating systems. We use UNIX[3] workstations, mostly SUN Sparc stations, and a variety of dedicated experimental systems, linked together through ethernet.

The experimental systems are used for image acquisition, image processing, and robot control. On the hardware level, where possible, we have opted for VME and MC680x0 based processor boards as a standard, viz. Force 30ZBE. On the software level, we have

[3] UNIX is a registered trademark of AT&T Bell Laboratories.

chosen for standard C and a widely used real-time operating system, viz. VxWorks[4]. VxWorks is in many ways compatible with our UNIX environment, e.g. UNIX files can be accessed directly and Internet type sockets can be used in much the same way as in UNIX, including the use of select statements.

The computing hardware for the Marie vehicle is a case in point. It is based on a VME bus with one or more general purpose processors and one or more dedicated processors. We currently use a Force 30ZBE with VxWorks as the main processor, a MC68000 based system for the ultrasonic sensor system and dedicated PID controller hardware (four NS LM628 on a Philips PG3679 board – the "Philips Motion Controller" or PMC) for the low level control. A (removable) Ethernet connection provides the communication between the VME bus system and the SUN network.

Software design for the MARIE vehicle

As stated before, we attempt to realize as much as possible a modular software structure for our robot. The philosophy of our approach is to develop control and sensor data processing modules as stand-alone processes, which communicate via standardized communication channels, viz. sockets. In specifying the modules, we strive for a structure that makes the software easily portable to other mobile robots, having a different hardware and different sensor types. This puts requirements on the level of abstraction at the interfaces, which must be as high as possible. I.e., going up from the lower software levels to the higher levels, we strive to go from the specific to the general as quickly as possible.

Part of the structure of the currently implemented control software is illustrated in Figure 1.

As we go up from the PMC driver module, which implements the driver for the hardware PID controller, to the virtual cart, we can clearly demonstrate this increase in abstraction level.

The PMC driver module must know about the representation used in the NS LM628 for all the PID control parameters. This involves knowledge about clock frequencies, scaling factors etc. The interface for the PMC driver module has been designed such that calling modules can specify filter parameters, velocities and such in terms of encoder ticks (this cannot be helped here), seconds and an output to the motors/actuators normalized to the range from -1.0 to +1.0.

The next higher level, the virtual cart, will be discussed more extensively below. It completely hides the existence of the PID controller hardware and many other specific properties of the cart from the higher levels and accepts path specifications in SI units. Thus we have arrived in just two steps at an interface that can be used for virtually any cart.

[4]VxWorks is a trademark of Wind River Systems, Inc.

The Virtual Cart

The virtual cart is a software layer that provides the developer of high-level software with a simple and consistent call interface that is, as far as possible, independent of the underlying hardware. It also provides certain basic safety features. The concept of a virtual cart plays an important role in achieving portability and data abstraction.

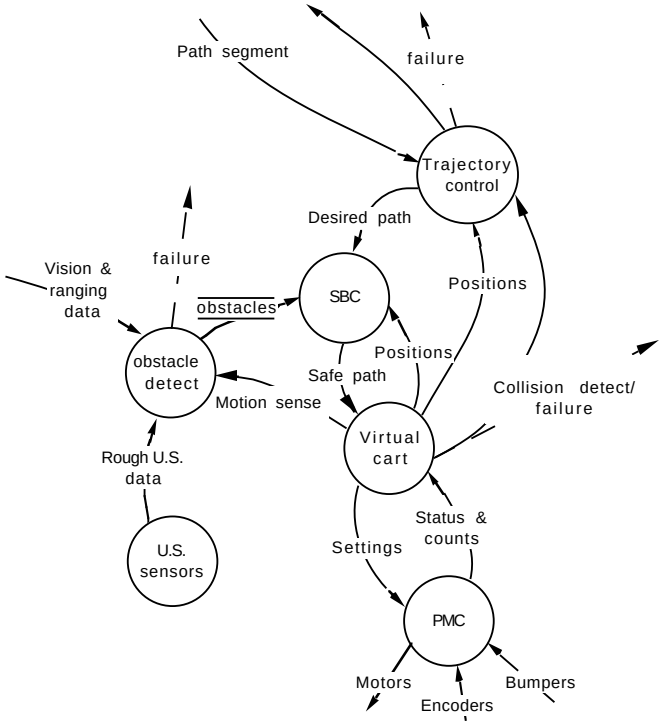

Figure 1. Part of the low-level software structure for the MARIE vehicle. The PMC module is the Philips Motion Controller board plus driver, described in this paper; the SBC module implements a sensor based collision avoidance strategy. The trajectory control and the SBC module are currently being tested in simulation in an environment provided by CARSIM and ASSIM.

In designing the virtual cart interface, it was necessary to choose a suitable control paradigm. Basically, three different types of wheeled vehicles are in use (disregarding a plethora of special purpose vehicles). The most versatile, but probably least common of these is the vehicle that can move in any direction in any orientation. The other two types have one fewer degree of freedom. They are those with one or more drive wheels providing propulsion, and one or more steered wheels, determining the path curvature and those where the driving and steering are combined. The former ("bicycles") cannot in general

turn in place, the latter ("wheelchairs") can. The most appropriate set of parameters to describe the behaviour of bicycles are speed and path curvature, for wheelchairs speed and turning speed. These differences are only important at low speeds and high path curvatures, where a singularity occurs in the conversion, otherwise both models are compatible.

We elected to implement both the bicycle and the wheelchair paradigms for the virtual cart. The virtual cart layer has been implemented and tested for a four wheeled vehicle that has two separately powered rear wheels and two steered front wheels coupled through a traditional trapezoidal system and controlled by a single electric motor. The cart thus has one more controlled axis than it has degrees of freedom. The virtual cart layer not only hides this redundancy from the user, but also hides details such as the wheelbase, encoder resolution etc. Commands to the virtual cart (bicycle paradigm) specify a path length relative to the current position of the cart, velocity, acceleration, path curvature and time derivative of the curvature all in SI units. Each new path specification immediately replaces the current specification.

Besides providing the desired abstractions, the virtual cart layer does more. Firstly, it significantly decreases the frequency at which higher level modules need to be executed. It does this by ensuring that a moderately complex path segment can be driven with good accuracy and by ensuring that those sensors that signal conditions that need to be reacted to on very short time scales are monitored, e.g. collision detectors attached to the bumpers of the cart. It also integrates the actual path as derived from the encoder readings to obtain estimated Cartesian positions (dead-reckoning).

Another important service of the virtual cart is that it provides some basic safety measures. The most evident measure is the monitoring of the collision detectors. If a collision is detected on one of the bumpers, the cart is prevented from driving further in that direction until the condition is reset by a higher software level and an exception is signalled. Similarly, if the desired and actual position of the cart differ too much, a stop is forced and an exception is signalled. A third measure is more subtle, and more easily circumvented - the higher level software must always specify a path length. This ensures that the cart will be stopped reasonably soon even if the higher level software crashes.

Development Environment

The software for the cart is developed and tested in simulation in a UNIX environment. UNIX is used because it provides a rich set of software development tools and is widely available. All the code has been written in C.

In the development stage new software modules are always tested in the simulation environment, after which they are implemented on the vehicle for final testing.

To support the development of software for experimental autonomous robots simulation is almost essential. New algorithms and new control paradigms can be developed and tested in a simulation environment before the hardware is even available. Debugging and testing of algorithms on a real machine is very time consuming, inefficient and in some situations even might be considered as dangerous, while off-line experiments can save a lot of time and allow the generation of measurement data in a very easy and flexible way. A simulation environment can give access to parameters which are not available in the real time application, like for example the position and orientation of the robot, or the voltages sent to the motors. Furthermore, a simulated environment can often be modified more easily than the real environment.

Yet, on the other hand, it must be realized that simulation can never entirely replace testing on the real system. Modelling of the exact mechanical behaviour of the cart is difficult at best, as is the simulation of the exact timing behaviour of a collection of processes running in a real-time environment. Also, the requirements of a simulation environment in the area of initialization, resets and user interfaces necessarily differ somewhat from the final application environment, often necessitating some code modifications when porting the tested software between the two environments. A simulation tool always implies a trade-off between realism and cost.

As our simulation tools are very much development tools indeed, they tend to be modified and extended as the development work on the vehicle's software progresses. Therefore, the work described in this paper must be seen as a snapshot of the status of a very useful and very intensively used set of tools but not as a final product.

The simulations are implemented on a graphical, UNIX-based workstation. We have two sets of simulation packages that we are currently integrating into a single environment. Some interaction is possible already. The first package - CARSIM - provides a simulation of the cart plus the low level control software. The second - ASSIM - provides sensor-simulation capabilities for a cart moving in a given environment.

The usefulness of simulation in the development of autonomous mobile robots systems has been reported by other authors; descriptions of other simulation packages for mobile robots can be found, e.g. in [1], [2], [3] and [4].

CARSIM

The simulation package for the vehicle consists of various sets of routines. One set of routines provides the actual simulation of the cart and the interface to the control software as provided by the Philips Motion Controller (PMC) driver. A second set of routines implements the simulation of the virtual cart. Furthermore two different user-interfaces are provided.

Besides providing essentially the same call interface as the actual PMC driver routines, the simulator routines in the first set also provide mechanisms that allow the state of the simulated vehicle to be interrogated and a routine to reset the simulation to its initial state.

The lowest level of the simulation software consists of computations of the dynamic and kinematic behaviour of the MARIE vehicle which has two independently driven rear wheels and two coupled, steerable front wheels controlled by one motor. In this part of the simulation the control outputs of the next level are translated, through a calculation of the mechanical response of the DC motors, to simulated sensor outputs (the axis encoders of

the three motors) which are used to control the different motors. The dynamical model[5] used for the MARIE cart takes into account some non-linearities as they occur in a real cart, such as a static friction, but disregards other complexities, such as the change in effective mass as the steer is turned, the free play in the rear wheels and the elasticity of the cart. Some care has been taken to ensure that the simulated cart can behave differently from the control model in the virtual cart layer, but in a physically sensible way. E.g. various dimensions can be chosen slightly different, allowing the effect of such discrepancies to become apparent. The equations are solved to second order accuracy where possible. The correspondence between the simulated cart and the real cart is such that it is found that the control parameters that work well for the simulated cart also allow a satisfactory control of the real cart.

The cart simulator also generates information about the absolute position of the cart, the average control outputs to the vehicle motors and several other state variables that are not normally accessible in a real cart. These value can be interrogated by e.g. the user interface and the sensor simulation software. A separate monitoring program is available that can plot the attained absolute positions and generate a log file of various state quantities.

A second set of routines extends the interface level for the vehicle simulation up to the level of the virtual cart. Extending the simulation to this level provides a uniform call interface, valid for all vehicles on which the virtual cart layer has been implemented. These routines differ only in certain control aspects from the real-time version.

Two sets of user-interface routines are provided. The first provides only some very basic capabilities to call all the various routines in the PMC driver and the virtual cart, but provides extensive status reporting facilities and thus allows the user to examine the effects of various calls in the various simulated control layers in detail. The primary purpose of this user-interface is to provide detailed testing and debugging facilities.

The second user-interface is more suited to the study of the interaction of the simulated vehicle plus control software with higher level control routines. It provides extensive graphical display facilities and user interaction. It also provides an interface to a sensor simulation package "ASSIM". The combination of this user-interface with the vehicle simulator and various layers of control software is referred to as "CARSIM".

ASSIM

The "Amsterdam Sensor Simulator"[6] is a software package which simulates sensor data from various sensors mounted on a (simulated) mobile robot. One or more robots can be positioned in a 2D environment. When an event in the environment occurs (the robot on which the sensors are mounted, or one of the other robots moves), the simulator recalculates all sensor data. All sensors that are mounted on a robot are thus kept on their correct simulated output continuously.

The current version contains a simulation of a laser range finder, an ultrasonic range finder, an ideal range sensor and a collision detector. For navigation on a grid, a grid-line detector and a transponder detector are implemented.

The environment is represented as a set of 2D polygons, each with a starting height and end height. Surface properties (as for example reflectivity for sound or for electromagnetic waves) can be set by the user. Data about the environment as well as data about the robots (shape etc.) can be stored in a simple data base.

For the generation of the sensor data, a model of the (physical) properties of the sensor is used. For the ultrasonic sensor we have taken into account the intensity profile of the transmitter, the attenuation by beam divergence, absorption in the air and by reflection, specular reflections and (spurious) signals because of multiple reflections.

The simulation package can be used stand-alone, where the user has the choice of three different user interfaces:

a) When running the simulator on a Sun workstation, the graphical user interface can be used. Different maps of the environment can be loaded using the buttons. Robots can be moved either by using the mouse or by entering the desired position and orientation in a special window. Sensors can be mounted or unmounted with buttons. Every sensor has a graphical representation of the data in a separate window.

b) In the non-graphical mode, a command line interpreter can be used to communicate with the program.

c) The user can write his own initial setup and sensor configuration and create routines for particular reactions on mouse or keyboard events.

A more extensive, but slightly outdated description of ASSIM can be found in a paper by Kröse [5].

Currently ASSIM is integrated with CARSIM in a single simulation environment. The environment and sensor configuration are still read from the database, but the position and orientation of the vehicle is provided by CARSIM. The simulated sensor data is available for other modules, and can also be represented graphically in a window on the screen. In this set-up, sensor based control modules (SBC in Fig. 1) can be tested in simulation before implementation on the actual vehicle.

Results

The modular software structure and extensive simulation tools have been tremendously useful in designing and testing our software, and also in the evaluation of new concepts. The availability of the virtual cart paradigm has greatly facilitated the porting of the trajectory controller software from the simulation environment to two different carts, one at the University of Amsterdam, the other at Robert Bosch GmbH.

After the initial design of the PMC driver interface, the simulation software for this driver was built first. Building this simulator was quite helpful in verifying the validity of the

---

[5] This model was for a large part designed and built by G.M. van der Molen of the University of Strathclyde.

[6] ASSIM was originally developed for the SPIN project 0710.133: "Planning methods and simulation for a semi--automatic vehicle," in collaboration with Industrial Contractors Holland BV.

interface design and also in improving our understanding of the actual operation of the real system. Part of the simulation routines, especially unit conversions, was directly useful for the actual driver.

The next step was the implementation of the simulation routines for the cart and the basic user interface. The first simulated version of the virtual cart was built and tested in this environment.

Work on the graphic interface and an early, simplified version of the cart simulator and control software (not incorporating either the PMC or the virtual cart) proceeded simultaneously. After the first tests on the virtual cart, a merged version was constructed.

The current simulation package still suffers somewhat from the different origins of its components. We are currently working on integration of ASSIM and CARSIM, while simultaneously providing facilities to implement the various control layers as separate UNIX processes, using slightly modified call libraries. One of the major issues in this case is the implementation of a suitable "event" or time manager to simulate the progress of time for all the simulated processes in a manner consistent with the inherently concurrent nature of the processes in the real-time system.

In porting our software to the real vehicle, we found that, as the level of abstraction increased, the differences between the simulated and the real software became progressively smaller. The PMC driver and its simulator have some common code, but differ greatly in most respects. The algorithmic structure of the virtual cart is essentially the same for the simulation and the real vehicle and most of the code is shared. It was necessary, however, to modify the control structure of the virtual cart, in order to ensure that it could meet the real-time requirements of the actual cart. Initial experience with the trajectory controller shows that the differences between the simulated and real version are even smaller.

We also found that the most damaging remaining errors occurred in the PMC driver software, which had not been tested in simulation. The virtual cart, though it had been tested with the simulator of the Amsterdam cart, and was first tested on the Bosch vehicle, did contain fewer damaging errors, as most errors had been found in simulation.

Acknowledgements

Most of the work described in this paper was done for the ESPRIT II project 2043 "MARIE". The authors acknowledge the support of the ESPRIT programme of the European Communities and the constructive interaction with other partners in the consortium. In particular, we wish to thank G.M. van der Molen of the University of Strathclyde for his contributions to CARSIM and M. Bergman of the University of Amsterdam for his work on the graphical interface.

References

[1] J. Meyer, "An emulation system for programmable sensory robots," IBM J. Res. Develop. (25) 6, 1981, 955-962

[2] J. Raczkowsky, K.H. Mittenbuehler, "Simulation of cameras in robot applications," IEEE Computer Graphics and Applications, January 1989, 16-25

[3] P. Adolphs, P.Léonard, J. Amelung, M. Augustyniak, A. Bletz, "SAMOS, a flexible simulation program for autonomous mobile systems," in "Intelligent Autonomous Systems 2," ed. T. Kanade, F.C.A. Groen, L.O. Hertzberger, 1989, Stichting International Congress of Intelligent Autonomous Systems, ISBN 90-800410-1-7, 630-640

[4] T. Knieriemen, E. von Puttkamer, R. Trieb, " 3d7 - A 3D simulation environment for autonomous system design," in "Intelligent Autonomous Systems 2," ed. T. Kanade, F.C.A. Groen, L.O. Hertzberger, 1989, Stichting International Congress of Intelligent Autonomous Systems, ISBN 90-800410-1-7, 434-440

[5] B.J.A. Kröse, E. Dondorp, "A sensor simulation system for mobile robots," in "Intelligent Autonomous Systems 2," ed. T. Kanade, F.C.A. Groen, L.O. Hertzberger, 1989, Stichting International Congress of Intelligent Autonomous Systems, ISBN 90-800410-1-7, 641-649