An Application for a Distributed Computer Architecture - Realtime Data Processing in an Autonomous Mobile Robot

Ralf Hinkel, Thomas Knieriemen, Ewald von Puttkamer

Computer Science Department, University of Kaiserslautern
Erwin Schrödinger Strasse, D-6750 Kaiserslautern, West Germany

Abstract

This paper describes a data processing structure for real-time systems with sensory feedback and a distributed computer architecture which supports it. The computer system is implemented within an autonomous mobile robot and is based on a set of independent processing modules, each of which consists of a processing core, a transport unit and a common memory between them. Communication between modules is done in two ways: For command transportation, a direct point-to-point connection is used and transport of raw and processed sensor data is done by broadcasting. Transportation is controlled by an adjustable time table, which is maintained by the supervisor. If he detects that a module does not process the given task in time, he slows down driving speed and adjusts the time table. This facility allows to adapt the action speed to the current processing capacity and finally to the complexity of the environment.

1. Introduction

Autonomous mobile robots (AMR) are systems which interpret, plan and execute a given task by their on board computer system. The ability to operate in an unknown or partly known environment without human intervention is essential for an AMR to be considered fully autonomous. As a function of their locomotion system they are classified in underwater, water, land, air and space robots. They use a complicated perception system, which allows them to construct a world model of their environment. Because of the inaccuracy of the perception system this model may be incomplete and some parts may be inconsistent, but ultimately it must be consistent in such a way that the robot can record its movements, determine its position and recognize obstacles in a reliable manner. Besides the sensor data processing, several planning and navigational capabilities like obstacle avoidance, exploration of unknown environment or path planning in known environment must be provided.

General System Architecture

The general data processing structure of real-time systems with sensory feedback may be considered to consist of two different structures (figure 1). First, a vertical control structure represents the general command flow among the different levels of control and second, a horizontal data processing structure represents the data stream among the components, which process the low-level sensor data into a more and more abstract description of the environment. The command flow is derived from a given task and controlled by the results of the data stream. Influenced by the actions of the system, which are the results of command execution, the actual environment is changing. Finally the reaction of the environment is recognized by the perception system, so that a closed loop is constructed.

![Figure 1: General data processing structure of real-time systems with sensory feedback](image-url)
Previous Work
The main focus on the current AMR research is directed to control structures, path planning, navigation, sensor techniques and data processing, but not to the computer architecture which is the base of the control architecture, especially of real-time systems. One of the earliest systems, the JPL robot [1], uses a distributed system with 3 computers and a hierarchy of separate concurrent processes, which are distributed among them.

Another more recent system, the CMU Rover [2,3] uses a distributed computer system for sensor preprocessing and motion control. The planning system runs outside on a VAX, which is connected via a radio link and a dedicated processor to the computer system of the robot. The distributed system, which is described by A. Elfes [4] and which is loosely coupled via a low speed serial link, uses the concept of cooperative experts. The asynchronous processes, which are located in the various processors, communicate with a central blackboard via its processor, which is the center of the star network used. Actual access to the blackboard is only gained through a monitor, to insure the integrity of the posted data. The blackboard mechanism has been used in several research areas, e.g. in speech understanding [5] and tracking of objects [6].

A distributed blackboard is used by Harmon in the Ground Surveillance Robot [7,8,9]. This System uses a set of subsystems, each of which consists of a tightly coupled multiprocessor group which communicates via shared memory. The blackboard itself actually resides in the common memory and is replicated in each subsystem, which is coupled to the LAN through an intelligent communication interface. This makes each system think that it is sharing blackboard memory with all other subsystems. Because the LAN interface is expensive hardware, the subsystems are implemented as normal multiprocessor architectures with all their known problems.

This paper describes a distributed computer architecture which is based on a set of independent processing modules, each of which consists of a processing core, a transport unit and a common memory between them. After introducing the physical robot and the control architecture, the software concept of the communication system and finally the implemented computer architecture is described.

2. MOBOT-III an Autonomous Mobile Robot

The proposed operating environment for the autonomous mobile robot MOBOT-III is the connecting area (entrance halls, passages, corridors, etc.) of big buildings like hospitals, universities, hotels, headquarters or ministries. In general, this area is well structured, less cluttered and with fewer changes of the local arrangement, but without any requirement for a static environment. In particular, there are main types of dynamic objects: moving objects (persons), temporary obstacles such as luggage at the registration counter or parked objects and objects with changing state like opened and closed doors. In order to reach other floors, the system should be able to use elevators and open doors automatically. Typical applications for such an indoor service robot would be any kind of transportation with point-to-point tasks, covering of the entire working area for floor cleaning tasks or guard functions and security inspection.

Physical Aspects
The physical structure of MOBOT-III is shown in figure 2. It is a three-wheeled vehicle with one driven and steered front wheel and two passive rear wheels, equipped with shaft encoders for odometry functions and internal position estimation. The empty weight of MOBOT-III is approximately 60 kg and the overall dimensions are 70 cm length, 52 cm width and 65 cm high. The vehicle uses two 12 Volt batteries with a capacity of 2 kWh for power supply. Maximum speed is about 2 m/sec and the average working speed is around 1 m/sec.

![Physical structure of MOBOT-III](image)

Figure 2: Physical structure of MOBOT-III

Multisensory System
The sensor component of MOBOT-III consists of a multi-level sensor system generating the primary input for navigation and collision-avoidance tasks. On top of the vehicle a rotating sensor unit is mounted, which produces a multi-sensory radar shots of the surroundings. One radar shot is composed of information from laser range-finders (720 points each cycle), ultrasonic and infrared sensors. An approximate obstacle detection is done by 13 ultrasonic sensors mounted around the vehicle chassis. In the case of a collision, a tactile bumpering stops the vehicle immediately.

User Interface
The communication interface is realized by an integrated but portable user panel. Its main components are a LC-Display of 620 *200 points, a joystick, five function buttons and a port for an optional keyboard. For user-friendly communication, operating can be done window-oriented with the joystick as mouse substitute. The whole organization with roll-down menus, icons, softswitches etc. is very similar to that of a Macintosh computer. Moreover there is a speech interface which allows vocalized output for special situations. The individual output can be chosen from a stored set of selected words.
Control Structure
Figure 3 shows the general command and data flow with the actual components of the control structure. It consists of a vertical command tree, constructed in a top-down manner from a global task to a set of subtasks and a horizontal data processing tree, which is processed bottom-up from primary data to composed models. The hierarchical control tree is composed of five levels of control. Only the upper three ones are influenced by the environment situation, which means that, with the results from the data tree, they generate an internal model and use this model for individual tasks.

Computer Architecture
Implementation of the control architecture is realized by a two-layered distributed computer architecture, with multiple processing modules communicating over a common serial bus. Each processing module is responsible for a particular processing, sensing or control task and is implemented on a single board which contains two µ-processors with local RAM and ROM. The computer rack contains all the main boards of the computer architecture chosen for MOBOT-III. Details of the implemented computer architecture and the control structure are given in the following sections.

Data Flow
The data stream which is produced by the sensors cannot directly influence the execution of the actions in the tree. First the data must be interpreted and concentrated to abstract structures and features of the environment. The data of the different sensors are analysed for consistency and are composed from one step to another into a more and more abstract description of the environment. This real-time mapping process is a complicated task, because the different levels of control need a different degree of abstraction in the environment modeling.

For example, a path planner only needs topological elements, like buildings, floors, rooms and corridors. It does not need the precise geometric placement of local obstacles, but the pilot needs such information to pass an obstacle in a reliable manner. This means that, not only the final output of the data processing tree is used, but also the results of intermediate stages influence the execution of the command tree.
A set of processing modules, which work parallel to and independent of each other with the given input, form the sensor data processing tree. The input of a single processing module is not limited to the results of the previous stage, but also the results from other stages may be necessary. The goal of every module is to extract features and to find a more abstract description of the received data stream.

Because the perception system produces information in static cycle times, a synchronous data flow must be established. This does not mean that the stages are clocked with the same rate, but that they have each a fixed cycle time between the data input and the output of the results. Because some stages need a sequence of successive results of a previous stage, the cycle time between the stages could differ. For example, if a processing module must find moving obstacles, it needs more than one radar shot to locate such an object.

At the point of system construction, these known cycle times for a known task allow splitting the task of the whole environment modeling into a set of processing modules, which interact in a known manner. Because of this static distribution of the task, there is no need for a task scheduler to manage the distribution at runtime.

The great advantage of such a static distributed system is the evidence of real-time ability. Another important advantage is the good testability of the processing modules, because they can be separated from the system and verified for a known set of inputs. By composing these single processing modules into the system, their internal structure and timing is not changed, so that no change in their behavior is expected.

Command Flow
In contrast to the data flow, the command flow is asynchronous, because no cycle time and common clock exists. Here a given task produces different subtasks which are again divided into subtasks. This process of dividing and transforming actions into commands is controlled by the environment model and the previous state of the system. Because of the great amount of possible actions and different environment situations, this processing is not completely computable in its reaction time. In the lower levels and in the modules close to the hardware, the response time is known, but higher levels and especially the planning levels show a great variability in their execution time.

This stands in opposition to the demanded real-time ability, but this can be cancelled out in such a manner that the
lower levels intercept the critical situations and give the upper levels more time to react to a new situation. For example: if an obstacle is detected by the laser radar, the navigator needs some radar sequences to decide whether the obstacle is stationary or a moving one. If this calculation takes too much time a crash will result. On the other hand, the pilot only needs one radar shot and a simple computation to determine whether an obstacle is close to the vehicle or not. This means that, if the navigator does not react in time, the pilot will slow down the driving speed and ultimately stop the vehicle.

Actual control structure
The actual control structure of the MOBOT-III system is shown in figure 5. Besides the processing modules, the command and data flow with typical cycle times is described. The hierarchical control tree is composed of five levels of control. Only the upper three are influenced by the environment situation, which means that, with the results from the data tree, they generate an internal model, e.g. the Local Area Map of the Navigator, and use this model for individual tasks. At the moment the sensor data processing is done in three stages: first, the sensor modules produce infos with lines as a main representing feature of the measured sensor data. Then this multi-sensor information is analysed and combined by the Composer into the Sensor Map which represents the environment around the actual robot position and finally the entire environment model is built up by the Cartographer and stored in the Global Geometric Map. Generating a topologic model will be part of future work.

A great advantage of the proposed architecture is the stability of the system against loss of data packages. Because of the continuous and cyclic data flow and the redundancy of information from one data sequence to the following one, a loss of a package implies no system crash. For example, if a radar shot is lost in the data stream, the modules can use the last one, because the robot only moves a small distance between two shots and as a result of the cyclic data generation in the perception system, a new radar shot will be available soon.

4. Software Concept of the Communication System
The main question which must be answered with regard to the computer architecture, is how the data are transported between the processing modules. A common memory is not the optimal solution, because it implies problems during the concurrent access between several independent modules. So the broadcasting mechanism is chosen to distribute the data packages, which are called "infos" here. To do this job, every processing module consists of a processing unit, a transporting unit and a communication interface between them (figure 6). Several different net structures could be used to transport the data from one module to another, but because of the broadcasting and the possibility of a simple expansion, a serial bus structure was favored.

To decouple the processing unit from the physical bus structure and to allow a concurrent processing and transportation of data, an intelligent transport unit with its own processing capacity is used. This gives flexibility and the ability to simply expand the system. To decouple the processing unit from the communication structures and to hide the transport system a common memory between them is used as communication interface. Finally the data is transported from the memory of one processing unit to the memory of the other processing unit without any help from either. This feature of the processing modules is mostly responsible for the efficiency of the proposed computer architecture.

Command Link
The logical interface between the processing modules depends on the interaction between them. In the command direction a point-to-point link is supported by the transport system. Here, one module could send a command to a specified module or inquire about a special piece of information which only belongs to these two. This means that, if the pilot sends a new track to the track control, or asks whether it could follow a given track or not, a command link is performed.

The message concept is used to realize command transportation. If a module wants to start an action at another module, it sends a command message. To do this, it tells the transport unit the module number and delivers the data that represent the command. When the command arrives at the final module, its transport unit brings the message to the common memory, sets a flag and initiates an interrupt, if arranged. In this function a mechanism is integrated, which detects a blocked module. If, at the time of the reception of a command, the transport unit recognizes that
a previous command was not accepted by the processing unit, it informs the sending module with a special receipt in the transport protocol.

**Data Link**

On the other hand, if i.e. the track control has determined the robot position as a result of the internal navigation sensors, this information is used by more than one module to adjust and to mark the processed data. By using the broadcasting mechanism for all data and results produced by the perception system and by the data tree, all modules can access these information simply. Because the transport and the processing unit share a common memory, the data can be transported directly to it, without any help from the processing unit. So the access to an info data package requires no extra time for the processing unit. The only thing it has to do is to access a variable, which holds the information and which is located at a known place in the common memory between processing and transport unit.

So in the processing unit, from the applications point of view, no explicit action must be done to access such an info data package. Here an abstract data type is used to hide the physical structure of the data packages. This gives more flexibility in data packaging and the ability to keep data format changes local. In MODULA-II, which was used to implement the application software, every information package has its own module, which supports the facilities described above. At system start these modules inform the transport unit which information packages must be caught from the bus and to which memory location they must be transported.

**Bus Arbitration and Synchronization**

In a real-time system, the transport system must be able to guarantee, that the data packages and commands are transported in time, this means that the timing of transfers must be precomputable. To avoid time expensive bus collision problems, the bus arbitration is done in a token bus manner. To guarantee the broadcasting of the infos, every bus master uses a copy of a predetermined time table, which fixes the sequence and time of data transportation. This means that the time table informs the current bus master, to interrupt the command transfer and to do the broadcasting of an info.

If needed, the transport unit can send an interrupt to the processing unit, which informs it that an information package has been received. This interrupt could trigger a process of the MODULA-II environment, which then starts the processing of the incoming data. With this mechanism the processing modules and especially the data tree can be synchronized simply. Also, a process of the application can be triggered via an interrupt by the broadcasting of the results. If a module receives a data package, it processes it and sends the results to other modules, where the same process starts again. Only the modules of the first stage of data processing, the sensor modules, synchronize themselves by a predetermined or hardware dependent cycle time. Because the transport unit increments a counter every time a data package arrives and the attached processing unit decrements it, overloads can be recognized by the transport unit and by the process itself. And because this condition is visible in the module state, the supervisor can also recognize this overload.

The result of a processing module, which must be sent to other modules, is placed in the common memory by the application. Then the application must set a flag, which signals that the data are valid. If the time table in the transport unit determines that the package must be transported, the transport unit gets the data and resets the flag. If the flag has not been set, which means that no package is available, this fail condition is signaled in the module state which is watched by the supervisor.

**Processing Speed Adaption**

If a module cannot execute the given task in time, meaning if the time between data entry and result output is greater than the time table allows, the supervisor slows down system speed and adjusts the time table for the package transport. For example, if the cartographer could process 50 lines of the sensor map per cycle maximum, and an unstructured and perhaps unexpected environment produces more, the transport system recognizes the overload, due to the results not being available as described in the time table. As a consequence, the supervisor slows down the driving speed of the robot and changes the time table. With this facility to adapt the action speed to the current processing capacity and finally to the complexity of the environment, the system can handle unexpected situations without a system collapse.

**Test Facilities**

Besides the command link and the broadcasting a byte sequential debugging link is implemented. It has no fixed partner and is used to build a channel between the operating system or application software of a module and the outside world. With this channel the application has the possibility of communicating with a user, even if the system is running. From the applications point of view, it looks and operates like a normal terminal interface. The transport system could connect this debugging link to several output stations of the system, like the communication interface or a speech output. Besides the link to the application layer, a link to the operating system is implemented, which allows downloading and debugging of application programs in a running system. With this facility the processing modules of the system can easily be tested under runtime conditions.

5. **Realization of the Computer Architecture**

Besides the partition of the control structure, which is discussed in the previous chapters and which is done in a functional manner, the computer architecture itself could be divided into two layers. The low-level computer system consists of independent hardware interfaces and supports a logical interface to the hardware. The low-level includes the motor units, the system control unit, the communication interfaces and simple sensors, like bumpers, infrared light barriers and ultrasonic modules.
Because the high level system, which represents the control and sensor data processing structure of the robot, is a focus of the current research, its design must be flexible enough to allow future changes in its structure. On the other hand, the low level with its separate hardware modules must only be flexible in such a manner that new hardware modules can simply be appended. So every hardware module gets its own microcontroller to support the desired logical interface and local intelligence. The current computer architecture is shown in figure 7.

Processing Module Implementation
As described in the previous chapter, the connection between the processing modules is done by an intelligent transport system and the connection between the transport system and the main processor (figure 8) is realized by a dual port ram. This allows the transport unit to place the received data and commands in the common memory. Also the data could be preprocessed and the message handling could be done in a reliable manner.

The transport processor is able to generate an interrupt for the main processor by placing a vector in the interrupt vector register. For test purposes a RS232 interface and a LCD-display is installed, which is logically connected to the debugging link of the processing unit. It is therefore invisible to the application, whether the link is connected via the RBUS or via the RS232 interface to a terminal. Also for test purposes the main processor has 5 state lamps and 3 switches, which could be used to select different features during the test phase of the application software.

Because the CPU in a low-level module is shared between the transport and processing unit, this type of module can not be online continuously at the bus. They also have a simplified version of the transport system, which only uses one command and data buffer. These form the connection of the application layer to the high level system. Aside from these buffers there are two other buffers, which function as the previously discussed debugging link between the outside world and the application and test environment.

Communication System
After the decision to use a distributed computer architecture, the main focus belongs to the communication system, which is responsible for the data and command transport. One constraint of the low level layer is the use of simple modules, which use only a simple microcontroller. Therefore LANs, like Ethernet, could not be implemented, because the interface hardware is more expensive than the control hardware itself. In most cases such systems also need a DMA transfer to transport the data from the interface to the memory quickly. Harmon [9] showed an implementation of an ethernet based system, and a similar approach will be used by Kampmann [10].
Finally, the decision was made to use a 187.5 kbaud serial data link, as is supported by the INTEL microcontroller family 8031 or 8096. One interesting system, the INTEL 8044, which supports the BITBUS interface [11] was not used, because no CMOS version exists. Moreover it is the only controller with this interface, so there is no opportunity to select another processor. Again, serial interface controllers like the INTEL 82510 are available today, which can be connected to every microprocessor and which have the same interface type as the 8031 controller family. The name of the bus is RBUS, which means Remote BUS and describes the functional characteristics of the bus: remote control hardware.

At the moment, the communication between the high level modules is also done by the RBUS; because of the chosen control and sensor data processing structure (figure 5), the required bandwidth is so small that the RBUS is fast enough to support the requirements of the communication system. If expansion time problems arise in the future, there is the possibility of adding a second transport system called CBUS. It will have the same structure as the RBUS, but use a faster bus. The applications point of view will be the same and the only difference will be in using another memory location for the data and commands.

6. Conclusion

In this paper a distributed computer architecture for an AMR is described, which is derived from a requirement analysis driven by the logical structure of the data processing in real-time systems with sensory feedback. It consists of independent processing modules and uses two different communication mechanisms: the direct point-to-point connection for the command link and broadcasting to distribute the data in the data processing tree. Because every module has an intelligent transport system, the communication and data processing is done concurrently. The transport system ensures the transparency of the underlying network structure.

The actual computer architecture consists of seven data processing modules performing the high level tasks and about 20 modules for low level tasks. In the chosen control architecture, all the communication is performed by the low data rate (20 kByte/sec) of the implemented RBUS. Another advantage of the architecture is the good testability of application and system software in a single data processing module of the running system by the implemented debugging link.

References


417