RTOS-based Software Architecture for Multisensor Fusion System

Zhang Kejun* and **Su Jianbo**

* Department of Automation & Research Center of Intelligent Robotics
Shanghai Jiaotong University
Shanghai, China.
e-mail: kejunzh@sjtu.edu.cn

Abstract

In this paper, software architecture for multisensor fusion system (MFS) in dynamic environment is described. The architecture has the characteristics of robustness, dynamically reconfigurable framework, "plug-in-play" for sensor, and predictability of sensor action. According to requirements of MFS, the real-time operating system (RTOS) is applied in this software architecture. The result is an RTOS-based software architecture, which consists of five kinds of tasks: Sensor Sampling Task (SST), Sensor Data Processing Task (SDPT), Fusion Task (FT), Management Task (MT) and Man/Machine Interface Task (MMIT). A calligraphic robot has been designed and developed in our lab based on the software architecture proposed. The application of the architecture makes the sensor management of the calligraphic robot system easy.

1 Introduction

Multisensor fusion system (MFS) is a complex information processing system. Several sensors that have different functions or spatial locations are integrated to get better information of environment. MFS has been applied in many fields such as military, satellite, aircraft navigation, robots and industrial assembly [19]. Currently, the research of MFS in dynamic environment has become an interested area [25]. In dynamic environment, a significant feature of MFS is that the sensor sampling, data processing and fusion must be finished in limited time. That is to say that real-time characteristic is an important factor for the availability of whole system. Normally, more sensors and complex hardware are applied to meet the requirement of real-time. Accordingly, with the workload of hardware management increasing dramatically, a good software structure is necessary to satisfy the real-time requirement of the whole system.

In fact, the software architecture has been of great interest in the field of MFS integration. The concept of "logical sensor" (LS) has been widely studied in MFS, which is a specification for the abstract definition of a sensor that can be used to provide a uniform framework for multisensor system integration [1, 2, 3]. In [4], an object-oriented programming method is proposed to develop a uniform framework for implementing multisensor systems, in which each sensor is represented as an object and objects communicate by passing messages. Literature [5] described the software architecture of science/autonomy system (SAS) in which each sensor was represented by a sensor object. Jun Wang [6] proposed an open, module and hierarchical software architecture for MFS, where component object model (COM) was used to build the whole software architecture. These methods facilitate system constitution, software packing and software re-usage. Unfortunately, they only focus on the efficiency of software design rather than the time constraint of system operating. MFS working under dynamic environment are required to be a real-time system. Thus time constraint is an additional important aspect, which determines performance of the system [7].

Accordingly, it is necessary to introduce the real-time operating system (RTOS) to develop the MFS under the dynamic environment. An RTOS can be divided into several parts, including the real-time kernel, the file system, and the programming environment. Among them, the real-time kernel is the key to endow a MFS with real-time feature. It provides local task management, scheduling, timing primitives, memory management, local communication, interrupt handling, error handling, and an interface to hardware devices. The architecture of a practical RTOS is shown in Fig. 1. As shown in it, several application tasks are running concurrently on the RTOS.

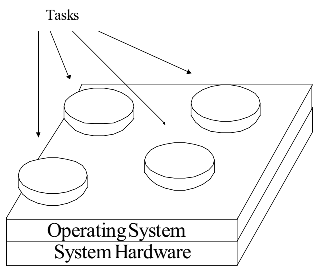

Figure 1: The architecture of RTOS in application

There are many kinds of RTOS. Every kind of RTOSs has its own characteristics. The most famous one is Vxworks, which has a successful application in the Mars Pathfinder. This RTOS implements a time slice of several microseconds [8]. QNX is another frequently used RTOS. In [9], the analysis and application of QNX were given. RT_Linux is another famous RTOS. Many applications of robot used RT_Linux [10]. All these OS are hard real-time operating system. Besides, Windows NT is also a kind of RTOS, which is a soft real-time operating system, and has widely been accepted as in developing applications [11]. This paper is to explore RTOS-based MFS software architecture, considering the requirements of MFS and the characteristics of RTOS, e.g. Windows NT. The proposed architecture is implemented in developing a calligraphic robot, which shows its feasibility and validity.

Section 2 discusses the requirements of the general MFS in detail, especially under dynamic environment. Section 3 proposes a RTOS based software architecture for MFS, which is independent of the specific applications. Section 4 shows a case study where a robot is used to write Chinese characters on the software architecture proposed, followed by the conclusions and future research.

2 Multi-sensor Fusion System

2.1 General Pattern

The general pattern of MFS proposed in [12] has been widely accepted. It is illustrated in Fig. 2. The following discussions are also based on this general pattern of the MFS.

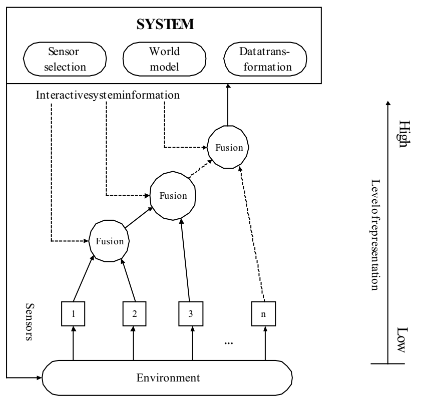

Figure 2: General pattern of MFS

The sensor units have the responsibility of acquiring the raw data from environment and preprocessing them, and then extracting the information to be fused from the sensor data. A network of fusion units is located above the sensor units. Information from different sources: sensor units, fusion units and interactive system are combined in each fusion unit. The integration functions play an important role in MFS. Three functions: sensor selection, world model and data transformation, can be used to manage and control the process. As shown in Fig. 2, the whole MFS is composed of several functional parts and all of the parts can be run concurrently.

2.2 General software framework

For MFS, general software framework is described in [6] (shown in Fig. 3). It is a three-tier framework that includes man/machine interface, logical part of MFS and database.

Man/machine interface is a key area of functionality for MFS, through which the user can interact with the system. It must provide flexibility of displays and open architecture of hardware to add new interacting ways.

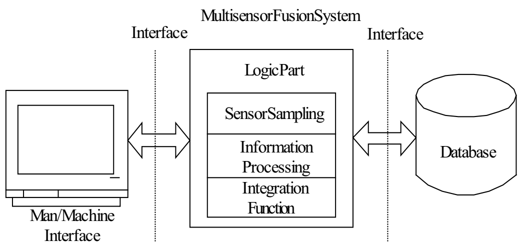

Figure 3: Three-tiered architecture for MFS

Database is used to maintain many various forms of data, including the sensory data, knowledge associated with sensor models, descriptions associated with the sensing environment. If an expert system is used, the database software must also manage a knowledge base including rules, frames, or other knowledge representation formats.

Logic part denotes the operation logics contained in the MFS. It is divided into three parts: the sensor sampling, the information processing and the integration functions.

The logic part is distinctive for MFS among these three parts. Therefore, the following sections will mostly focus on the design of logic part. The man/machine interface will also have a little mention. Detailed discussions for man/machine interface and database can be found in [13, 14].

2.3 Analysis of requirement

Robust software architecture is necessary in MFS especially in some cases that sensors may fail. When a sensor fails, the system should also be able to work, although in a suboptimal way. Under the dynamic or dangerous conditions, sensor's failures are very common. Thus redundant sensors are often included. Luo presented an example of a practical application where the system used two same sensor groups to improve the system robustness in fire monitoring task [15]. This implies that robust and efficient software architecture is on demand to organize the sensor system so that the system can work in a proper way.

Dynamic configuration is an essential feature of a MFS in dynamic environment. There are two situations that require reconfiguring the sensor system. In some applications, it is not necessary to involve all sensors in the system at the same time to fulfill tasks. Thus the sensor system needs to be reconfigured in its structure. Moreover, a specific sensor in the sensor system may work in different patterns during the procedure of task fulfillment. Thus the parameters of the sensor or its weight contribution to the whole fusion system may be adjusted. Any of the two cases require proper software architecture to support the dynamic reconfiguration of the sensor system that makes the MFS works more efficiently [16].

An MFS should be able to include new sensors to meet requirements from more complex environment and applications. Thus the software structure of MFS must also support the "plug-and-play" of sensors.

In dynamic environment, the primary concern in MFS is that it is predictable. The predictability in a MFS means that each sensor has its own activating time slot. This ability needs the software to support. We mention that the deadline handling mechanism in the RTOS allows an exception handler to be automatically called when the deadline fails.

3 Multi-sensor Fusion System

3.1 Real-Time Operating System (RTOS)

RTOS is a multitask operating system. The multiple tasks are run by switching tasks (threads) scheduled by RTOS. The task is a simple program. This program runs as if the CPU is occupied by itself. Normally, every task is a never-stop circulation. At any time, the task is at one of the five states: idle state, ready state, executing state, blocked or suspended state and interrupted state. Fig. 4 shows the state relationships together with the jumping conditions. RTOS schedules all the tasks to ensure the right task gets the resource and runs. There are three basic task-scheduling methods, including round-robin, non-preemptive and preemptive. In round-robin scheduling algorithm, every task is assigned the same time slice. Thus, even if a more emergent task is ready, it should still wait for its own time slice to run. In non-preemptive scheduling algorithm, the interrupt can be respected quickly. If the current task is interrupted by a higher priority task, non-preemptive scheduling preserves the current task and only switch to the higher priority one after the current task is over. But in preemptive scheduling algorithm, the higher priority task will be run immediately by stopping the current task [17]. In RTOS, preemptive scheduling method is required [18], which is an inherent characteristic to meet the requirements of MFS. Table 1 shows the MFS requirements and the according performing on RTOS.

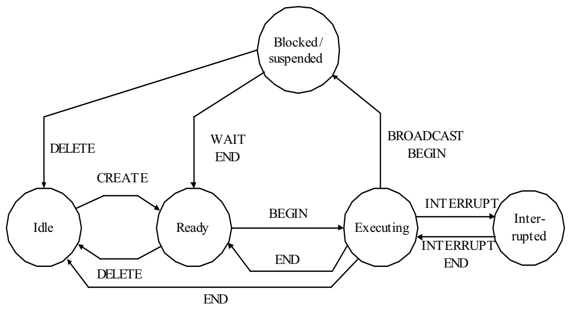

Figure 4: Task states of RTOS

Table 1 The MFS requirements and the solutions by RTOS

Requirements of MFS	Solutions by RTOS
Robust software architecture when some sensors fail.	The sensor performing can be packed into tasks that are running concurrently. When one sensor fails, the other tasks can run without being affected.
Dynamically reconfigurable of the sensor performing.	The tasks of sensor performing have several states on RTOS. The change of sensor performing states is finished by the change of tasks'.
The software support of "plug-and-play" of sensors.	Adding or deleting the corresponding tasks of sensor can meet the requirement of this.
The software support of predictability of sensor's activating time slot.	The deadline mechanism of RTOS meets this requirement very well.

3.2 RTOS-based software architecture for MFS

According to the requirements of MFS and features of RTOS, a software architecture based on RTOS is illustrated in Fig.5. Five kinds of tasks are classified in the RTOS-based MFS software. The Sensor Sample Task (SST) is responsible for sensor data sampling. Sensor Data Processing Task (SDPT) deals with the data processing of sensor measurements. A Fusion Task (FT) achieves data from SDPT, other FTs and database to get a fusion result. Management Task (MT) deals with some management events of MFS. Man/Machine Interface Task (MMIT) supplies the interaction of human and computer of MFS.

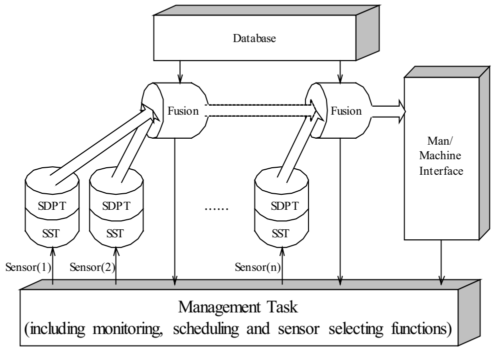

Figure 5: RTOS-based software architecture for MFS

Since the requirement analysis above is done on the general pattern, this architecture is independent of the specific application. In RTOS-based MFS software, the priorities of all tasks decide the sequence by which the tasks run. This means we can easily change current system tasks by adjusting its priorities. We can also stop some functions of the MFS by simply suspending or killing correspondent tasks. For example, if an MFS does not need camera, the task of camera's SST and SDPT must be suspended. Moreover, if an MFS needs new sensors, what we should do is just to add corresponding tasks.

Fig. 6 illustrates the concurrently executing mode of the MFS tasks on RTOS. RTOS locates over the MFS hardware. Every kind of tasks of MFS run concurrently based on RTOS and the communication of them are performed through RTOS.

The software frame of tasks, except for MT, is as follows:

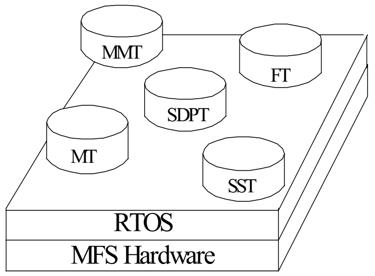

Figure 6: Tasks of MFS based on RTOS software architecture

```
Task (LPVOID lparam)
{
        Initiating
        Sending task start signal to MT
        Task implementing
        Error detecting
        Sending task end signal to MT
}
```

The start and end signal of the task are used to justify if the task fails the deadline.

SST is located at the bottom of whole system, which is in charge of the direct interaction with hardware. It includes sensor sampling, sensor error detecting and data transferring functions. Combining these functions into SST, a device-independence can be achieved by MFS. It is worth mentioning that the sensor error detecting function included in SST is able to detect its own errors and transfer the errors to MT.

SDPT gets raw data from SST and extracts the valid information for fusion. It includes data processing, error detecting, and information transferring functions. Here, the error detecting function is to justify the overflow of raw data. The extracted valid information is transferred to FT by information transferring function. In some conditions, SST and SDPT might be simple and could be combined into one task.

The purpose of FT is to analyze the information from SDPT and get a description of environment. It includes fusion and information transferring functions. The fusion function further comprises of three sub-functions as data alignment, data association and information fusion. The final fusion result could be observed from MMIT. Since the structure of the fusion task is independent from other tasks, a MFS can employ different fusion architecture and algorithms to meet the requirement of applications in a flexible way.

MT is a very important task for MFS based on RTOS. There are three functions: the monitor function, the schedule function and the sensor selecting function. The monitor function includes the deadline monitoring function (DMF) and the error monitor function (EMF). As a real-time system, the correctness of the system depends not only on the logical result of computation but also on the time moment in which the results are produced. The time constrains are expressed by deadlines of the tasks. Normally, before building MFS, the rationality of the deadlines should have been analyzed. However, the dynamic features of the environment or the uncertainty of the hardware can not guarantee that all tasks can meet their deadlines in practice. DMF is used to monitor if the tasks fail their deadlines. It chooses one of following solutions to deal with deadline failures:

Stop the task immediately and switch to another task when the deadline failures are caused by some un-resumable errors, such as a hardware failure.

If the deadline failure is caused by some resumable errors, the task’s priority is adjusted to be the highest. All of the system resources are assigned to the task so that the task could be executed as fast as possible. This could happen when a FT fails the deadline.

In the worst case, the system must be shut down.

In MT, EMF deals with all kinds of errors of MFS, such as hardware errors of SST, errors of SDPT and so on. Just like the DMF described above, the EMF deals with these errors in different ways. The basic way is to stop the task that has errors. Since this function is independent from other functions, it is easy to add new solution mechanisms according to applications.

The scheduling function in MT decides the priority of each task. In different states of work, the focus of MFS is different. For example, when the FT is executing, the priority of it should be highest to sure this task can be finished ASAP. For the variability of MFS, different applications have different priority rules. The basic scheduling algorithms are RM and EDF [21].

The sensor selecting function chooses which sensors should be used by MFS now. In many conditions, not all of the sensors in a MFS should be involved. Over-using of all sensors not only may not improve the fusion results, but also increase workload of the system. For the robustness of a MFS, many redundant sensors are installed in the system [22]. The sensor planning is necessary in these systems. By applying RTOS to design the MFS software, we can easily stop or activate a sensor by simply suspending (killing) or resuming the corresponding tasks instead of changing the raw code of programs if a non-RTOS is invoked.

The man/machine interface task (MMIT) shown in Fig. 6 is used to transfer messages between man and machine. This is necessary if operator should interfere in the action of the MFS or observe the fusion results.

In addition, there are communication and synchronism problems in RTOS-based MFS software. This problem is an inherent problem in all RTOSs. Data reservations are important issues that should be addressed. Possible solutions may be from techniques, such as critical protection, semaphore, mailbox, and message queue [23].

4 A real application

The software architecture for MFS has been utilized and verified in a calligraphic robot system, shown in Fig.7.

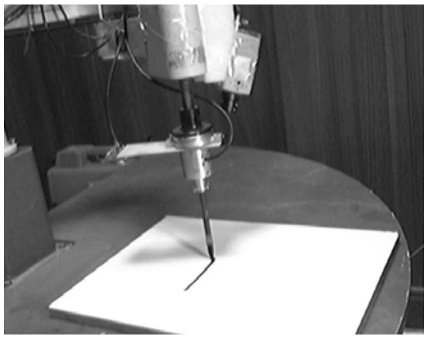

Figure 7: Hardware framework of the calligraphic robot

4.1 Software architecture for the calligraphic robot

The system is constructed by a robot and an MFS. The robot is an Adept One robot arm that is an assembly manipulator with four degree-of-freedoms. MFS includes three sensors: a camera, sonar and a tactile sensor. The writing brush is connected with the tactile sensor. The tactile sensor and sonar are fixed at the end of the robot. The camera is located over the robot workspace (See Fig. 8).

The objective of the system is to control the robot to write Chinese characters with the help of multi-sensor feedback information. The Chinese characters are achieved from a man/machine interface. The camera gets the image of Chinese characters and extracts the width of the current stroke. The sonar measures the distance from brush's end to paper. The width of stroke is wider with the distance shorter. Thus, the system can extract the width of the stroke from the data of the distance. The tactile sensor measures the pressure of brush during writing. The width of the stroke is proportional to the pressure. Thus, the width of current point of the stroke can be extracted from the pressure. Three widths coming from three sensors are fused to get a more precise one. By comparing with the width of the stroke in database, the system computes the next coordinate of x, y, z and transfers it to robot. Then the robot is driven to finish the movement prescribed.

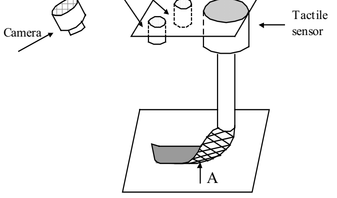

Figure 8: framework of system configuration

The MFS in the calligraphic robot are realized by seven tasks according to the three kinds of sensors adopted and run in the RTOS. Table 2 describes each task and its functions.

There are 4 tasks that are instances of SST and SDPT:

The camera sampling task is used to acquire the images of Chinese characters. The camera is a Color-Camera. By using a Graphic Capture Card, the images are transferred into PC.

In image processing task, the width of stroke is extracted through the picture graying, binarizing and edge extraction. Then the width is transferred to the fusion task.

The sonar sampling and processing task are used to get the distance of robot end to paper. The SST and SDPT of sonar are combined into one task for the job of computing width is light. The distances obtained by sonar are transferred into PC by serial port.

The tactile sensor's SST and SDPT deal with the tactile data achieved during the writing brush activating and get width from the data. They are also combined into one task for the same reason above. The force data are transferred into computer by a multi-function card.

The widths coming from different sensors are fused in the fusion task. In our application, a weighted average method is used as fusion algorithm.

4.2 Experimental results

In current stage, we just conduct the calligraphic robot to write a stroke of Chinese character based on an MFS with RTOS-based software architecture. Here we use Windows NT. Windows NT is a soft real-time OS where the switching time slice is longer than hard real-time OS' such as Vxworks or QNX, but it offers all the characters of hard RTOS. Moreover, it has many advantages over traditional RTOSs. Firstly, it offers a wide range of Commercial-Off-the-Shelf components unmatched by any traditional RTOS. Secondly, as a sophisticated PC OS, it includes many development tools from third-party vendors and reduces the risk of project developing. While Windows NT is not quite the de facto standard in the real-time market, it is making strong progress. And the commitment of Microsoft to continue to push and support Windows NT is unquestioned. By experience, the robot-activating interval is less than 100ms. The time slice of Windows NT is about 8~10 ms, so it is fit for our application.

By applying the RTOS-based software structure in the calligraphic robot, the sample rate and CPU usage are both higher. Furthermore, the MFS structure can be reconfigured easily.

This task is also implemented in the Windows NT without taking advantage of the idea of RTOS. Table 3, Fig. 9 and 10 show the comparison results.

Table 3 Comparison results of sensors sample rate

The experiment without the idea of RTOS (times/unit).			The experiment of RTOS-based Software Structure of MFS (times/unit).		
Camera	Sonar	Tactile	Camera	Sonar	Tactile
16	16	16	16	28	90

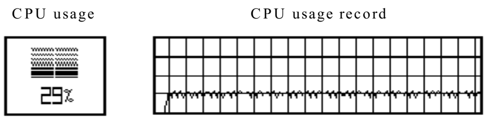

Figure 9: CPU usage rate in serial software structure

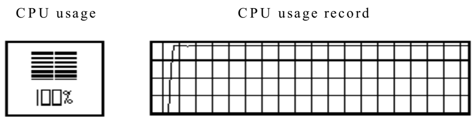

Figure 10: CPU usage rate in RTOS-based software

In Table 3, the unit is the time of writing a line of 150mm by the robot system. We can find that in one-by-one sampling mode, the times of sensor measuring is decided by the slowest sensor. In our application, it is camera. The number is 16 times in every unit. The measurement tasks of sensors in RTOS-based software structure are executing concurrently, so the number of sensors sampling is different. We can get a more precise description of the dynamic environment from the more sampling times of sensors. The satisfaction of the MFS real-time requirement is reached.

In Fig. 9 and 10, the left ones show the CPU usage rate and the right ones are the CPU usage records by using the two different software architectures in MFS. We can find that the efficiency of CPU usage is higher in RTOS-based software structure.

From experiments, we find out the data of tactile sensor is not stable at the beginning action of robot. Thus, we suspend the SST and SDPT of the tactile sensor at the beginning, and then resume the tasks when the action is stable by the MMIT. This is just one condition where the performing is passive for the sensor invalidity to reconfigure the MFS structure. There are many other conditions where we must actively reconfigure the structure of MFS in order to get better performance of the MFS.

4.3 Sensor management

The RTOS-based software architecture for MFS is used in the calligraphic robot system. It is convenient to deal with sensor management problem. The sensor management is to dynamically change sensor combination and configuration as to achieve the given sensing goals better [20]. Because of aiding in sensor data fusion, it has been used in many dynamic MFS applications [26]. The sensor management is necessary in the calligraphic robot as well. In our case, three different kinds of sensors are integrated. In many conditions, not all the data of each three sensors are valid. For example, in the connecting and overlaying parts of strokes, the image sensor cannot exact a right stroke width. As shown in Fig. 11, it is a middle stage of writing "王". Now, it is writing the third stroke: "Shu". The stroke starts at the middle of the first stroke and goes through the second one. At the connecting part (I) with the first stroke and in the overlaying part (II) with the second stroke, the image sensor cannot exact the valid width of "Shu" stroke. We should stop the image sensor in both parts and restart it when the brush passed them.

To illustrate the effect of the sensor management, we do a comparative experiment. In Fig. 12, it is the one without sensor management. In Fig. 13, it is the one using sensor management. We can find the part in dotted circle in Fig. 12 is too large and not smooth. The data of image sensor in the part do bad effects to the fusion result. The part (I) and part (II) in Fig. 14 is better.

The conditions of connecting and overlaying of strokes are very common. They appear in various locations of Chinese characters. Thus, the system must meet the requirement of easily performing the sensor management. The RTOS-based software architecture satisfies this requirement very well. In this software architecture, the sensor sampling and data processing of different sensors are synchronous executed in different tasks. Such as, stopping or restarting of image sensor can be performed by suspending or resuming the task..

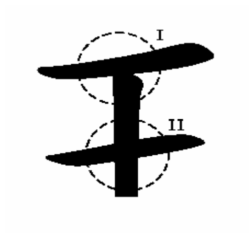

Figure 11: The middle stage of writing "王"

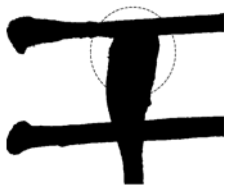

Figure 12: Without sensor management

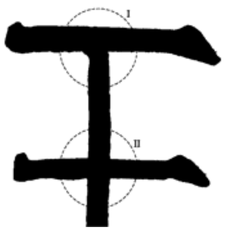

Figure 13: Using sensor management

5 Conclusions and future work

In this paper, we presented the RTOS-based software architecture for MFS. In dynamic conditions, MFS has following requirements: a robust software structure to deal with the sensor failure, dynamically reconfigure the structure, and the predictability for sensor action. We used RTOS-base software architecture for MFS to meet the requirements above. This software architecture includes not only the features of robustness, dynamically reconfiguring, "plug-in-play" for sensor and predictability but also the features of predecessor's software architecture such as: reusability, upgradeability and modularity.

We implemented this software architecture in a calligraphic robot that has three kinds of sensors: camera, sonar and tactile sensor. The software architecture is built in a soft RTOS: Windows NT. While Windows NT is not purposely built for real-time systems, the design of Windows NT includes enough RTOS elements (such as a priority-driven, preemptive scheduler) to make it a viable choice for building real-time applications. In the future, the number and type of sensors in our MFS will increase to perform more complex tasks. More features of RTOS-based MFS software structure will be explored, such as the scheduling algorithm, deadline analysis.

References

[1] T.C.Henderson, E.Shilcrat. *Logical Sensor Systems. Journal of Robotic Systems*, Vol 1(No.2), pp.~169--193, 1984.

[2] Tom Henderson, Chuck Hansen, Bir Bhanu, *The Specification of Distributed Sensing and Control*, Journal of Robotic Systems, Vol 2(No.4), pp.~387--396, 1985.

[3] Mohamed Dekhil and Thomas C. Henderson, *Instrumented Sensor System Architecture*, The International Journal of Robotics Research, Vol 17(No.4), pp.~402--417, 1998.

[4] T.C.Henderson, E.Weitz. Multisensor Integration in a Multiprocessor Environment. Proc.ASME Int. Comput. In Engr. Conf. and Exhibition, R. Raghavan and T.J.Cokonis, Eds.,New York, NY, pp.~311--316.

[5] J.A. Lopez-Orozco, An Open Sensing Architecture to Autonomous Mobile Robots, in: Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference, Gaithersburg, MD, USA, September 1998, pp.~610--615

[6] Jun Wang, Jianbo Su, Yugeng Xi, *COM-based Software Architecture for Multisensor Fusion System*, Information Fusion, pp.~261—270, Feb, 2001.

[7] David B. Stewart, Donald E. Schmitz, Pradeep K. Khosla, The Chimera II Real-Time Operating System for Advanced Sensor-based Control Applications. Systems, Man and Cybernetics, IEEE Transactions on, Nov/Dec 1992, Vol 22, pp.~1282--1295.

[8] http://www.windriver.com/success/jpl_pathfinder.html

[9] Matt Verber, Real-time Operating System, http://people.msoe.edu/~sebern/courses/cs384/papers98/verber.pdf

[10] http://www.linuxdevices.com/news/NS8416393595.html

[11] Fully Robotic Hospital Pharmacy uses Windows® NT and Windows® CE, http://www.entivity.com/case_studies/128.pdf

[12] Ren C. Luo, Michael G.Kay. *Multisensor Integration and Fusion in Intelligent Systems*. IEEE Transaction on Systems, Man and Cybernetics, Vol 19(No.5), pp.~901--931, 1989.

[13] D.L Hall, Mathematical Techniques in Multisensor Data Fusion, Artech. House Inc, Norwood, MA, 1992.

[14] R.T. Antony, Database Support to Data Fusion Automation, Proceeding of the IEEE, January 1997, Vol 85 (No. 1), pp.~39--53.

[15] Ren C. Luo, Kuo L. Su, Kuo H. Tsai. Fire Detection and Isolation for Intelligent Building System Using Adaptive Sensory Fusion Method. Proc. IEEE International Conference on Robotics&Automation. Washington, DC. May 2002. pp.~1777--1781.

[16] Sukhan Lee, Xiaoming Zhao, A New Sensor Planning Paradigm and Its Application to Robot Self-localization, IEEE/RSJ/GI Int. Conf. on Intelligent Robots and Systems, 1995. Vol 2, pp.~462--467

[17] C. M. Krishna, Kang G. Shin, Real-Time Systems, Published by The McGraw-Hill Companies, Inc. 1997.

[18] http://www.faqs.org/faqs/real-time-computing/faq/.

[19] P. K. Varshney, *Multisensor Data Fusion*, Electron. Commun. Eng. pp.~245--253, June. 1997.

[20] Sukhan Lee and Xiaoming Zhao, "A New Sensor Planning Paradigm and its Application to Robot Self-localization", Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on , vol.2, pp.462 -467, 5-9 Aug. 1995.

[21] Kevin Jeffay, *Analysis of a Synchronization and Scheduling Discipline for Real-Time Tasks with Preemption Constraints*, IEEE, 1989, pp.~295--305.

[22] J. Gu and M. Meng, A. Cook, M. G. Faulkner, Micro Sensor Based Eye Movement Detection and Neural Network Based Sensor Fusion and Fault Detection and Recovery, the 3rd World Congress on Intelligent Control and Automation, June 2000. Vol 1, pp.~518- -522.

[23] Jean J. Labrosse, uc/OS-II: the Real Time Kernel, Published by R&D Books, an imprint of Miller Freeman, Inc.2000.

[24] Nakadai, K.; Hidai, K.; Okuno, H.G. and Kitano, H., Real-Time Speaker Localization and Speech Separation by Audio-Visual Integration, Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, Vol 1, pp.~1043 --1049.

[25] Luo, R.C.; Kay, M.G.; Lee, W.G., Future Trends in Multisensor Integration and Fusion, Industrial Electronics, 1994. Symposium Proceedings, ISIE '94, 1994 IEEE International Symposium on, pp,~7--1225-27.

[26] Adrian, R.A. “Sensor Management”, Digital Avionics Systems Conference, pp.~32--37, 1993. 12th DASC., AIAA/IEEE , 25-28 Oct. 1993.