15th Triennial World Congress, Barcelona, Spain

A SOFTWARE FRAMEWORK FOR MOBILE ROBOT SENSOR FUSION AND TELEOPERATION

P.Pérez, J.L.Posadas, J.E.Simó, G.Benet, F.Blanes

Departamento de Informática de Sistemas y Computadores (D.I.S.C.A.)
Universidad Politécnica de Valencia, Spain
{ pperez,jposadas,jsimo,gbenet,pblanes}@disca.upv.es

Abstract: This paper describes an architecture for mobile robots that is suitable for teleoperation. The architecture is hybrid and has three levels: a reactive level, a deliberative level and an interface level. Reactive level is distributed and it is made up of sensor nodes and controllers. On the other hand, deliberative level establishes the knowledge and the reasoning capability to the robot. This deliberative level uses reactive level (sensor and actuator values) through the interface level. Interface level is made up of a communication system that allows local and remote access to the robot from any deliberative node. The architecture has been implemented as a case study in the YAIR robot, enabling its teleoperation from a remote node using Windows CE.

Keywords: Mobile robots, Teleoperation, Communications Systems, Distributed computer control systems, Sensor fusion, Fieldbuses.

1. INTRODUCTION

Teleoperation and remote changes in mobile robots behaviour is often necessary. For example, for changing the operation mode, status monitoring or manual control of the robot. Therefore, it is important to develop a system to allow easy runtime access to the robot. The goal of the work is to develop a multilevel architecture for mobile robots that is based on a communication system called SC (Posadas, 1997) that allow transparent local and remote access to the robot from any node. The SC is a communication system used in various applications all related with distributed control with soft real-time constraints. The combination of the SC and a reactive local control in the robot allows an easy communication between nodes accomplishing with real-time constraints.

The architecture has been implemented and tested in YAIR[1] (Gil, 1997; Blanes, 1998; Blanes, 2000), an autonomous robot with intelligent sensors that produces different measurements about the environment and its location within it. A case study for this robot teleoperation has been developed using an embedded PC running Windows CE in a real scenario.

The development of the communication system is based on the concept of temporal firewall (Kopetz, 1998). Therefore, the values of the sensors and actuators are labelled with the time of its acquisition. This time stamp is updated through the different levels and components of the architecture and communication system. When the requested values of

[1] YAIR stands for Yet Another Intelligent Robot, and is currently being developed under CICYT grant TAP98-0333-C03-02 from the Spanish Government

the sensors arrive to their destination, the corresponding software can validate them according to the time elapsed since they were obtained.

In the following sections, the developed work and the results obtained are described. First, the global architecture of YAIR is described. Then, the implemented communication system called SC is presented. Finally, the results of the tests carried out are presented.

2. ROBOT ARCHITECTURE

The distributed architecture is hybrid and it is composed by three levels: a reactive level that deals with real-time constraints, a deliberative level without real-time constraints (although good mean response time must be guaranteed as well as some processing distribution capabilities), and an interface level acting as a connection between the other two levels.

YAIR is an autonomous robot prototype (Figure 1) that has been built for the experimental study of reactive systems, sensor fusion and distributed computing.

Fig.1: YAIR Robot Prototype

The backbone of the YAIR's reactive level is the CAN bus (Bosch, 1991), a fieldbus initially developed for the automotive industry that is actually being used in numerous technological areas, specially in mobile robotics, due mainly to its reliability and versatility. Its medium access mechanism, its multimaster capability, and the ability to detect transmission errors make it suitable for distributed real-time systems. Reactive level is made up of intelligent sensor modules and computing nodes that use the bus to share the sensory information.

Reactive level of YAIR has mainly three sensory nodes: a motion controller and odometry reckoning node, an infrared node that supervises 16 IR detectors (it detects objects and estimates the distance from them and the robot within a range from 10 cm. to 1 m.), and an ultrasonic sonar node that provides range information about environment objects up to 4 metres in distance.

Sensory nodes have reckoning capability, meaning that they pre-process the information. This pre-processed data (sensations) are shared through the bus, making it accessible to the remaining nodes. (i.e.: the speed vector computed by the motion controller can be used by the ultrasonic sonar module to point the sonar head towards the displacement direction).

Communications between processes running in CAN nodes use shared variables. Processes use an uniform interface to access to the shared variables: Request Function (to read an specific variable) and Write Function (to write an specific variable). This interface uses CAN messages in a transparent way. When a process needs to know data from a sensor, it just only reads the corresponding variable or object.

Deliberative level of YAIR has a local control node that manages the external communications and executes the main control application program. Deliberative level establishes the knowledge and the reasoning capability to the robot. It can be implemented by external nodes for distributed processing, remote controlling, status monitoring, and so on. Communications between these two levels and external nodes are possible using an interface level that it is described in the following section.

3. COMMUNICATION SYSTEM

Due to the different set of processes involved in robot sensor fusion and control, an heterogeneous and flexible framework has been designed to access all information with independence of the communication channel used. This is performed using common interface for all the channels (adapter) defined in the CAdapt C++ class. This class only defines a set of virtual functions ensuring the coherence for all derived class from this. Actually in the robot there are three different physical channels to obtain data: the CAN bus, serial ports and Radio-Ethernet network. This last one is covered using an application interface layer called SC. With these assumptions three adapters have been developed: (CAdaptCAN, CAdaptCOM and CSCAdapt) with the same interface.

With this approach, independence from channels is obtained, and the problems related with them (frame formats, bit or character oriented communication etc) are hidden in the adapter objects which resolve these details internally, offering a common access to all the information in all levels, to fused information from different sources.

High-level access to distributed data has been provided by developing a system called Communication System (SC) (Posadas, 1997; Posadas, 2000) (see fig.2). This SC hides communication details behind an uniform bind-notification interface.

SC holds an internal representation of the data objects using a distributed blackboard (Penny, 1989). This data structure is continually updated with the changing values of the objects. SC needs also a program instance running in each node of the system. SC software establishes the required communications to ensure that all the copies of the distributed blackboard are consistent.

Processes must only execute local accesses to contact with all the system. That is, when a process needs to obtain the value of a sensor (for example, the velocity of the motors) it must only to contact with the associated object, which is defined in the local SC that is executing in the computer where belongs the process.
The system is Event Driven. So, it's possible to associate the code execution with specific events (a change on the value of one object).

Several C++ classes and interfaces to contact with the SC have been defined. In this form, processes connect with SC in similar way following these steps:

1. First, processes have to instance an object from the CSCAdapt class. This class offers a common interface that permits local accesses with SC using Dynamic Data Exchange (DDE).

```
class CAdaptSC : public CAdapt
{
        .....................................
 bool Init();
 bool Finish();
 bool RegisterSensor(CSensor* pSensor, char* pId);
 bool UnregisterSens(CSensor* pSensor, char* pId);
 int Write(char* pBuffer, int len, char* pId);
 int Read(char* pBuffer, int len, char* pId);
};
```

2. Second, processes have to implement an object from the parent class called CSensor, too. CSensor is a virtual class that defines a set of functions which have to be implemented to communicate with SC through CSCAdapt class.

```
class CSensor: public CObject
{
 public:
  virtual bool OnInit(CAdapt* pAdapt)=0;
  virtual bool OnFinish(CAdapt* pAdapt)=0;
  virtual bool OnMessage(const char* pBuffer, int
len, const char* pId) = 0;
};
```

3. Third, processes register its sensors objects calling RegisterSensor function from CSCAdapt. With this function, processes associate a sensor object (pSensor parameter) with an object (pID parameter) of the SC distributed blackboard ("pID" parameter is the name of the object in the blackboard).
4. After that, when the value of the blackboard object changes, the CSCAdapt object instance executes the OnMessage function of the sensor object. In this way, a process receives new values from SC automatically (data is obtained from pBuffer parameter of OnMessage function).
5. When a process wants to change the value of a blackboard object, it only has to execute the local Write function. This is the way that processes in the system can control the robot using the variables associated with the actuators.

The distributed blackboard generated by the SC software is extensive to the data in the CAN network. The gateway software ISCCAN performs specific translations between CAN protocol and SC data (see fig.2). The ISCCAN gateway supports communication of the CAN raw data, as well as the mapped mode that consists of a bi-directional mirroring of CAN identifiers and objects in the distributed blackboard. The mapped mode allows processes running in every node into the IP network to have access to the CAN information through the SC software and the defined notification scheme.

Two kind of problems arise in implementing the data chain that links low-level processes running in CAN nodes and high-level processes running in computer nodes:

- Data format conversions and serialisation coherence.
- Semantic guided data filtering.

The ISCCAN gateway solves the data format conversion and serialisation using ASCII-Hex representation of CAN binary streams. SC distributes these streams for selective processing. Processes translate this information using a supplied object

toolbox. The SC mapped mode allows the use of defined filtering by applying the SC general bind-notification scheme.

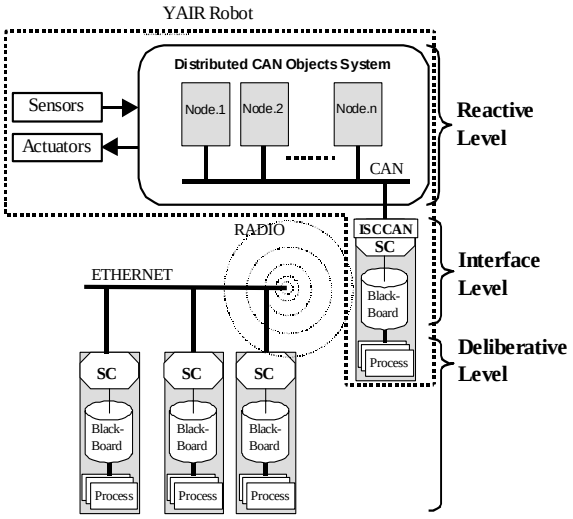

Fig.2: Communication System

The SC blackboard level is intended to provide communications for deliberative soft real-time processes. This kind of processes must manage communication overloads of 20ms introduced by the SC+ISCCAN system.

Typically, deliberative processes are related to sensory integration, data fusion and map building. In this case, when temporal and spatial sensory fusion is essential, time properties must be attached to sensor data and control actions. The time property attached to each SC blackboard object is in the form of a time firewall (a register that accumulates all the communication overloads). To achive this, we attach a temporal counter reset at the moment of information generation, which is increased between stages in the same distributed node. If this information travels through the communication channel, the information time is increased using the maximum latencies of transmission and preset with this time in the reception instant. From this information, each process using data have available the t_{use}-$t_{observation}$ difference and uses it in data integration tasks.

In a recent paper (Han et al., 2001), a remote control architecture for robots is described. It is based on the control of the internet time delay, and reduces the time difference between a real internet-based robot and a virtual one. On the other hand, this work distinguishes between communication time (where internet time delay is unavoidable) and processing time. That is, deliberative nodes communicate through Internet to send information and program code (with soft real-time restrictions) to reactive nodes, and then reactive nodes process this information and program code with hard real-time restrictions.

Thus, it is not necessary to consider internet time delay to control obstacle avoidance. In reactive nodes are the processes that control automatically obstacle avoidance, and in deliberative nodes are only the processes which send to the robot information about the path to follow (objective). Reactive nodes receive this information (reception time delay is not critical) and compose it with the local information about obstacle avoidance. The result is a forward movement to the objective without crashing into obstacles. The next section describes the prototype implemented to test the architecture designed.

4. PROTOTYPE

Each deliberative node of YAIR uses SC to gain access to distributed CAN objects system through the gateway ISCCAN. Processes that provide YAIR a deliberative behaviour can be executed in every computer where a local SC instance exists. These computers are called homogeneous nodes because of they belong to the SC configuration (see fig.3). There are nodes where it is impossible to execute a SC instance (for example, an embedded PC where the operating system does not have the SC requirements or and Unix system), in these cases the processes have to communicate with a remote SC through a specific gateway application. These nodes are called heterogeneous nodes (see fig.3). For example, there is a socket gateway that permits every process in every heterogeneous node to connect with SC establishing a standard socket connection. Processes connect with the gateway and this communicates with SC.

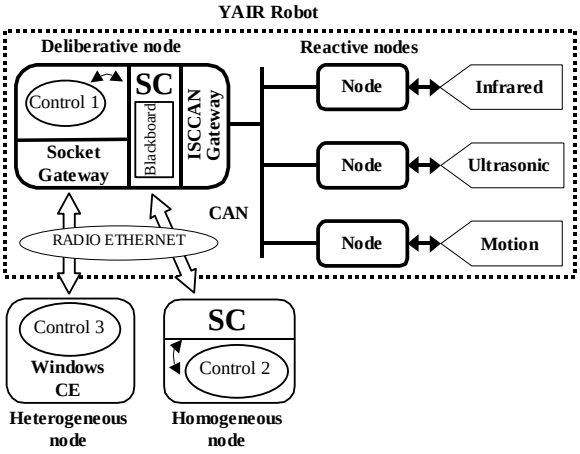

Fig.3: Implemented prototype communications architecture.

Each reactive node can execute basic behaviours using real-time distributed data on CAN. Control actions are obtained in each node using the behaviour composition model described in (Simo, 1997). This model is similar to Arkin's model (Arkin, 1990) with the addition of the distributed behaviours selection system based on motivation concept. In agreement

with this model, teleoperation can be considered as the execution of a behaviours set that contribute to control actions together with the executed behaviours in reactive nodes (i.e. obstacles avoidance).

On the other hand, teleoperation can consist of the remote execution of a motivation tasks set that contribute to basic behaviour composition.

In the last release there are three CSensor derived classes: CSensorIR, CSensorBrujula, CSensorHaz, CSensorTelemando and CSensMotores.
The CSensorIR reads data from the IR sensor in the YAIR robot using the CAN bus adapter. The instance of this derived class holds a thread that periodically (50 milliseconds) reads a communication object in the CAN adapter. This involves sending a CAN message to the IR sensor modules which returns another one with data from sensors. This message is filtered (extracting the header), routed to the objects interested in this information (those registered) using OnMessage function. Once the information arrives to the CSensIR instance, a pre-processing is performed and data stored internally.
The CSensMotores derived class works in a similar way, reading periodically (100 milliseconds) a communication object to obtain from motor encoders the information about odometry. This information is the base for location calculus and could be fused with the data from CSensorBrujula (compass) for position estimation.

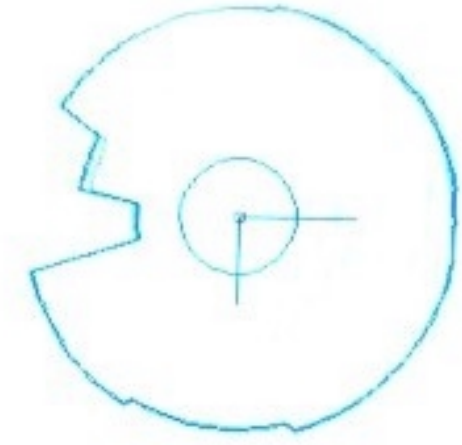

Fig.4:Graphical representation of the CSensorHaz after fusing the information obtained from CSensIR object.

The CSensorHaz is an object that holds data fused from CSensIR information. To obtain that information, the CSensIR object offers a set of interface functions that return the value of reflected light and ambient light from a sensor in the ring. These values are transformed to distance units in the CSensorHaz object and fused to a ring representation using the distance values, the angle covering each sensor, robot position and time elapsed. This ring representation could be used to avoid obstacles in the environment. The graphical representation of the CSensorHaz is shown in Figure 4.

Finally, the CSensorTelemando is a sensor that acts like a joystick using a graphical representation in a tactile panel. The data from the sensor (orientation, speed) is used to form a direction vector for the robot. This vector is sent to the robot using the CSCAdpat described in previous sections.

Using these software components, a software module based on Windows CE for remote control of YAIR has been built to validate the communication system and the communications among SC and heterogeneous nodes through gateways. This module allows to send the speed for each wheel to the motion controller.

Windows CE software is executed in an embedded PC (fig.4). The application establishes a radio Ethernet connection with a “socket gateway” that is running in the deliberative node of YAIR (fig.3).

The “Socket gateway” sends to local SC the messages that it receives from Windows CE module. On the other hand, SC sends the received messages to ISCCAN gateway and, finally, ISCCAN sends them to reactive nodes through CAN bus. In this way, motion controller node can receive messages from the remote embedded PC and it can compose the corresponding control actions.

Fig.5: Robot teleoperation using an embedded PC

The Windows CE application has an easy graphical interface (fig.5) that allows to select the desired target direction of YAIR. Then, it is sent to motion controller node through a wireless IP network and the described SC+ISCCAN facilities. Composition of teleoperation orders (desired target direction) with reactive behaviour orders (obstacle avoidance) produces the suitable commands to be sent to the motion controller. The result is a friendly and robust

teleoperation because of operator has not to worry about robot collisions.

5. CONCLUSIONS

A communication system suitable for remote control and access to real-time systems with distributed sensory architecture is described. The main advantage of the system is that different type of computers (from a palmtop to main workstations in the network), could be linked to access the robot throught the distributed blackboard.

This system has been implemented in the YAIR robot, an autonomous robot with intelligent sensors that produces different measurements about the environment and its position within it.

The SC+ISCCAN combination solves the high-level data diffusion in the distributed blackboard system. Remote accesses to YAIR are possible from homogeneous SC nodes and from heterogeneous nodes using gateways.

A Windows CE module is described for controlling the movements of YAIR. This application allows to send speed values to motion controller node through a wireless IP network and the described "Socket gateway" and SC+ISCCAN facilities.

6. REFERENCES

Arkin, R.C., *"Integrating behavioural, perceptual and world knowledge in reactive navigation"*. Robot and Autonomous Systems vol. 6, pp. 105-122, 1990.

Blanes, F., Benet, G., Pérez P., Simó, J.E. (2000) *"Map Building in an autonomous robot using infrared sensors"*, in proc. of IFAC Symposium on Intelligent Components and Instruments for Control Applications SICICA'00. Buenos Aires.

Blanes, F., G. Benet, M. Martinez, J.Simó. (1998). *"Grid Map Building from Reduced Sonar Data"*. in proc. of IFAC International Symposium on Intelligent Autonomous Vehicles. IAV'98. Madrid.

Bosch (1991) *"CAN Specification 2.0A"*. © Robert Bosch GmbH.

Gil, J.A (1997). *"A CAN Architecture for an Intelligent Mobile Robot"*, in Proc. of SICICA-97. pp.65-70.

Han Kuk-Hyun, Sinn Kim, Young-Jae Kim, Jong-Hwan Kim (2001). Internet Control Architecture for Internet-Based Personal Robot. Autonomous Robots 10, 135-147.

Kopetz, H., Nossal, R. (1998) *"Temporal Firewalls in Large Distributed Real-Time Systems"*. in proc. of 6th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems.

Penny, H. (1989) *"Blackboard Architectures and Applications"*. Edited by V. Jagannathan, Rajendra Dodhiawala, Lawrence S. Baum.

Posadas, J.L., J.Simó, F.Blanes (1997). *"Un modelo para el desarrollo de aplicaciones distribuidas. El Servidor de Comunicaciones"*. in Proc. of Jornadas españolas de Automática (JA'97). Gerona 1997.

Posadas, J.L., Pérez, P., Simó, J.E., Benet, G., Blanes, F. .(2000) *"Communications Structure For Sensor Fusion in Distributed Real Time Systems"*. in proc. of 6th IFAC Workshop on Algorithms and Architectures for Real-Time Control AARTC'00.

Simó, J., A. Crespo, J.F. Blanes (1997). *"Behaviour Selection in the YAIR Architecture"*. in proc. of Proceedings of IFAC Conference on Algorithms and Architectures for Real Time Control AARTC'97. Vilamoura, Portugal.