Vehicle Applications of Controller Area Network

Karl Henrik Johansson,[1,⋆] Martin Törngren,[2,⋆⋆] Lars Nielsen[3]

[1] Department of Signals, Sensors and Systems, Royal Institute of Technology, Stockholm, Sweden, kallej@s3.kth.se
[2] Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden, martin@damek.kth.se
[3] Department of Electrical Engineering, Linköping University, Sweden, lars@isy.liu.se

1 Introduction

The Controller Area Network (CAN) is a serial bus communications protocol developed by Bosch in the early 1980s. It defines a standard for efficient and reliable communication between sensor, actuator, controller, and other nodes in real-time applications. CAN is the de facto standard in a large variety of networked embedded control systems. The early CAN development was mainly supported by the vehicle industry: CAN is found in a variety of passenger cars, trucks, boats, spacecraft, and other types of vehicles. The protocol is also widely used today in industrial automation and other areas of networked embedded control, with applications in diverse products such as production machinery, medical equipment, building automation, weaving machines, and wheelchairs.

In the automotive industry, embedded control has grown from stand-alone systems to highly integrated and networked control systems [11, 7]. By networking electro-mechanical subsystems, it becomes possible to modularize functionalities and hardware, which facilitates reuse and adds capabilities. Fig. 1 shows an example of an electronic control unit (ECU) mounted on a diesel engine of a Scania truck. The ECU handles the control of engine, turbo, fan, etc. but also the CAN communication. Combining networks and mechatronic modules makes it possible to reduce both the cabling and the number

⋆The work of K. H. Johansson was partially supported by the European Commission through the ARTIST2 Network of Excellence on Embedded Systems Design, by the Swedish Research Council, and by the Swedish Foundation for Strategic Research through an Individual Grant for the Advancement of Research Leaders.

⋆⋆The work of M. Törngren was partially supported by the European Commission through ARTIST2 and by the Swedish Foundation for Strategic Research through the project SAVE.

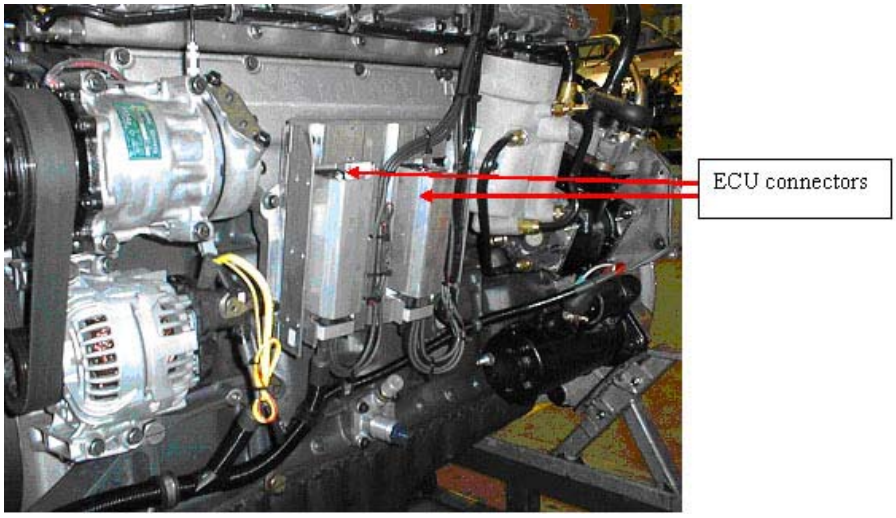


Fig. 1. An ECU mounted directly on a diesel engine of a Scania truck. The arrows indicate the ECU connectors, which are interfaces to the CAN. (Courtesy of Scania AB.)

of connectors, which facilitates production and increases reliability. Introducing networks in vehicles also makes it possible to more efficiently carry out diagnostics and to coordinate the operation of the separate subsystems.

The CAN protocol standardizes the physical and data link layers, which are the two lowest layers of the open systems interconnect (OSI) communication model (see Fig. 2). For most systems, higher-layer protocols are needed to enable efficient development and operation. Such protocols are needed for defining how the CAN protocol should be used in applications, for example, how to refer to the configuration of identifiers with respect to application messages, how to package application messages into frames, and how to deal with start-up and fault handling. Note that in many cases only a few of the OSI layers are required. Note also that real-time issues and redundancy management are not covered by the OSI model. The adoption of CAN in a variety of application fields has led to the development of several higher-layer protocols, including SAE J1939, CANopen, DeviceNet, and CAN Kingdom. Their characteristics reflect differences in requirements and traditions of application areas. An example is the adoption of certain communication models, such as either the client-server model or the distributed data-flow model [13].

The progress and success of CAN are due to a number of factors. The evolution of microelectronics paved the way for introducing distributed control in vehicles. In the early 1980s there was, however, a lack of low-cost and standardized protocols suitable for real-time control systems. Therefore, in 1983 Kiencke started the development of a new serial bus system at Bosch,

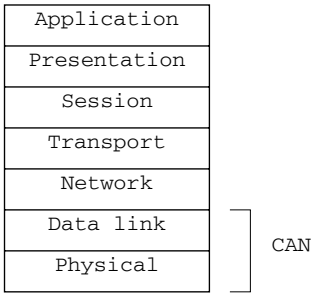


Fig. 2. The CAN protocol defines the lowest two layers of the OSI model. There exist several CAN-based higher-layer protocols that are standardized. The user choice depends on the application.

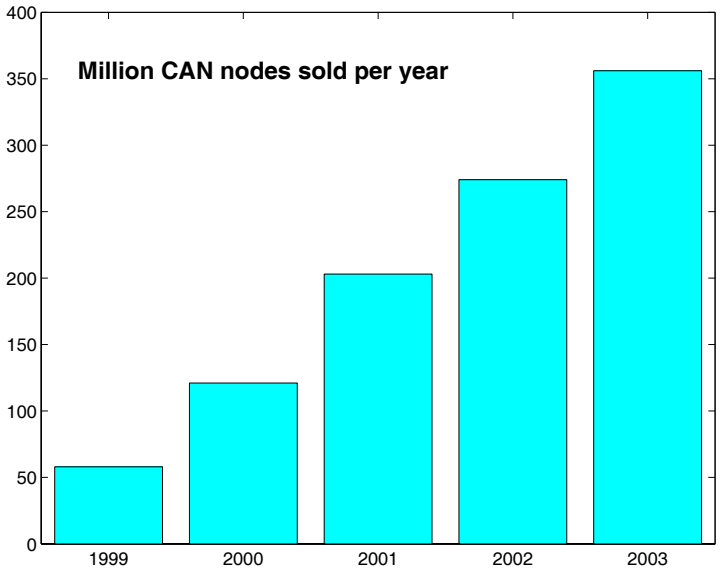


Fig. 3. The number of CAN nodes sold per year is currently about 400 million. (Data from the association CAN in Automation [3].)

which was presented as CAN in 1986 at the SAE congress in Detroit [8]. The CAN protocol was internationally standardized in 1993 as ISO 11898-1. The development of CAN was mainly motivated by the need for new functionality, but it also reduced the need for wiring. The use of CAN in the automotive industry has caused mass production of CAN controllers. Today, CAN controllers are integrated on many microcontrollers and available at a low cost. Fig. 3 shows the number of CAN nodes that were sold during 1999–2003.

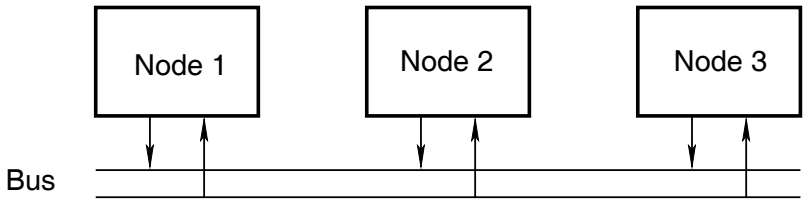


Fig. 4. Three nodes connected through a CAN bus

The purpose of this chapter is to give an introduction to CAN and some of its vehicle applications. The outline is as follows. Section 2 describes the CAN protocol, including its message formats and error handling. The section is concluded by a brief history of CAN. Examples of vehicle application architectures based on CAN are given in Section 3. A few specific control loops closed over CAN buses are discussed in Section 4. The paper is concluded with some perspectives in Section 5, where current research issues such as x-by-wire and standardized software architectures are considered. The examples are described in more detail in [14]. A detailed description of CAN is given in the textbook [6]. Another good resource for further information is the homepage of the organization CAN-in-Automation (CiA) [3]. The use of CAN as a basis for distributed control systems is discussed in [13].

2 Controller Area Network

The Controller Area Network (CAN) is a serial communications protocol suited for networking sensors, actuators, and other nodes in real-time systems. In this section, we first give a general description of CAN including its message formats, principle of bus arbitration, and error-handling mechanisms. Extensions of CAN, such as application-oriented higher-layer protocols and time-triggered CAN, are described, followed by a brief history of CAN.

2.1 Description

A CAN bus with three nodes is depicted in Fig. 4. The CAN specification [4] defines the protocols for the physical and the data link layers, which enable the communication between the network nodes. The application process of a node, e.g., a temperature sensor, decides when it should request the transmission of a message frame. The frame consists of a data field and overhead, such as identifier and control fields. Since the application processes in general are asynchronous, the bus has a mechanism for resolving conflicts. For CAN, it is based on a non-destructive arbitration process. The CAN protocol therefore belongs to the class of protocols denoted as carrier sense multiple access/collision avoidance (CSMA/CA), which means that the protocol listens

Fig. 5. CAN message frame

to the network in order to avoid collisions. CSMA/CD protocols like Ethernet have instead a mechanism to deal with collisions once they are detected. CAN also includes various methods for error detection and error handling. The communication rate of a network based on CAN depends on the physical distances between the nodes. If the distance is less than 40 m, the rate can be up to 1 Mbps.

Message formats

CAN distinguishes four message formats: data, remote, error, and overload frames. Here we limit the discussion to the data frame, shown in Fig. 5. A data frame begins with the start-of-frame (SOF) bit. It is followed by an eleven-bit identifier and the remote transmission request (RTR) bit. The identifier and the RTR bit form the arbitration field. The control field consists of six bits and indicates how many bytes of data follow in the data field. The data field can be zero to eight bytes. The data field is followed by the cyclic redundancy checksum (CRC) field, which enables the receiver to check if the received bit sequence was corrupted. The two-bit acknowledgment (ACK) field is used by the transmitter to receive an acknowledgment of a valid frame from any receiver. The end of a message frame is signaled through a seven-bit end-of-frame (EOF). There is also an extended data frame with a twenty-nine-bit identifier (instead of eleven bits).

Arbitration

Arbitration is the mechanism that handles bus access conflicts. Whenever the CAN bus is free, any unit can start to transmit a message. Possible conflicts, due to more than one unit starting to transmit simultaneously, are resolved by bit-wise arbitration using the identifier of each unit. During the arbitration phase, each transmitting unit transmits its identifier and compares it with the level monitored on the bus. If these levels are equal, the unit continues to transmit. If the unit detects a dominant level on the bus, while it was trying to transmit a recessive level, then it quits transmitting (and becomes a receiver). The arbitration phase is performed over the whole arbitration field. When it is over, there is only one transmitter left on the bus.

The arbitration is illustrated by the following example with three nodes (see Fig. 6). Let the recessive level correspond to “1” and the dominant level to “0”, and suppose the three nodes have identifiers I_i, $i = 1, 2, 3$, equal to

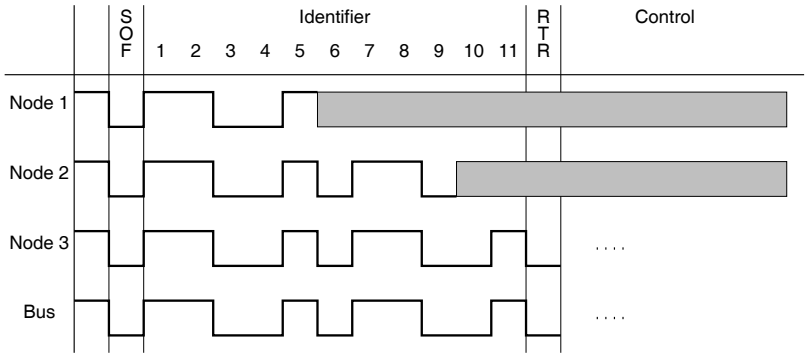


Fig. 6. Example illustrating CAN arbitration when three nodes start transmitting their SOF bits simultaneously. Nodes 1 and 2 stop transmitting as soon as they transmit one (recessive level), and Node 3 is transmitting zero (dominant level). At these instances, Nodes 1 and 2 enter the receiver mode, indicated in grey. When the identifier has been transmitted, the bus belongs to Node 3 which thus continues transmitting its control field, data field, etc.

$$I_1 = 11001101010, \quad I_2 = 11001011011, \quad I_3 = 11001011001.$$

If the nodes start transmitting simultaneously, the arbitration process illustrated in the figure takes place. First all three nodes send their SOF bit. Then, they start transmitting their identifiers and all of them continue as long as they are equal. The sixth bit of I_1 is at the recessive level, while the corresponding bits are at the dominant level of I_2 and I_3. Therefore, Node 1 stops transmitting immediately and continues only listening to the bus. This listening phase is indicated in Fig. 6 with the grey field. Since the tenth bit of I_2 is at the recessive level while it is dominant for I_3, Node 3 is the transmitter that has access to the bus after the arbitration phase and thus continues with the transmission of the control and data fields, etc.

There is no notion of message destination addresses in CAN. Instead each node picks up all traffic on the bus. Hence, every node needs to filter out the interesting messages on the bus. The arbitration mechanism of CAN is an effective way to resolve bus conflicts. Note that a minimum amount of address data is transmitted and that no extra bus control information has to be transmitted. A consequence of the arbitration mechanism, however, is that units with low priority may experience large latency if high-priority units are very active.

Error handling

Error detection and error handling are important for the performance of CAN. Because of complementary error detection mechanisms, the probability of having an undetected error is very small. Error detection is done in five different

ways in CAN: bit monitoring and bit stuffing, as well as frame check, ACK check, and CRC. Bit monitoring simply means that each transmitter monitors the bus level, and signals a bit error if the level does not agree with the transmitted signal. (Bit monitoring is not done during the arbitration phase.) After having transmitted five identical bits, a node will always transmit the opposite bit. This extra bit is neglected by the receiver. The procedure is called bit stuffing, and it can be used to detect errors. The frame check consists of checking that the fixed bits of the frame have the values they are supposed to have, e.g., EOF consists of seven recessive bits. During the ACK in the message frame, all receivers are supposed to send a dominant level. If the transmitter, which transmits a recessive level, does not detect the dominant level, then an error is signaled by the ACK check mechanism. Finally, the CRC is that every receiver calculates a checksum based on the message and compares it with the CRC field of the message.

Every receiver node obviously tries to detect errors within each message. If an error is detected, it leads to an immediate and automatic retransmission of the incorrect message. In comparison to other network protocols, this mechanism leads to a high data integrity and a short error recovery time. CAN thus provides elaborate procedures for error handling, including retransmission and reinitialization. The procedures have to be studied carefully for each application to ensure that the automated error handling is in line with the system requirements.

2.2 Protocol extensions

CAN provides the basic functionality described above. In many situations, it is desirable to use standardized protocols that define the communication layers on top of the CAN. Such higher-layer protocols are described below together with CAN gateways and the time-triggered extension of CAN denoted TTCAN, which allows periodic access to the communication bus with a high degree of certainty.

Higher-layer protocols

The CAN protocol defines the lowest two layers of the OSI model in Fig. 2. In order to use CAN, protocols are needed to define the other layers. Fieldbus protocols usually do not define the session and presentation layers, since they are not needed in these applications. The users may either decide to define their own software for handling the higher layers, or they may use a standardized protocol. Existing higher-layer protocols are often tuned to a certain application domain. Examples of such protocols include SAE J1939, CANopen, and DeviceNet. It is only SAE J1939 that is specially developed for vehicle applications. Recently, attempts have been made to interface CAN and Ethernet, which is the dominant technology for local area networks and widely applied for connecting to the Internet.

SAE J1939 is a protocol that defines the higher-layer communication control. It was developed by the American Society of Automotive Engineers (SAE) and is thus targeted to the automotive industry. The advantage of having a standard is considerable, since it enables independent development of the individual networked components, which also allows vehicle manufacturers to use components from different suppliers. SAE J1939 specifies, e.g., how to read and write data, but also how to calibrate certain subsystems. The data rate of SAE J1939 is about 250 kbps, which gives up to about 1850 messages per second [6]. Applications of SAE J1939 include truck-and-trailer communication, vehicles in agriculture and forestry, as well as navigation systems in marine applications.

CANopen is a standardized application defined on top of CAN and widely used in Europe for the application of CAN in distributed industrial automation. It is a standard of the organization CAN in Automation (CiA) [3]. CANopen specifies communication profiles and device profiles, which enable an application-independent use of CAN. The communication profile defines the underlying communication mechanism. Device profiles exist for the most common devices in industrial automation, such as digital and analog I/O components, encoders, and controllers. The device can be configured through CANopen independent of its manufacturer. CANopen distinguishes real-time data exchange and less critical data exchange. It provides standardized communication objects for real-time data, configuration data, network management data, and certain special functions (e.g., time stamp and synchronization messages).

DeviceNet is another standardized application defined on top of CAN for distributed industrial automation. It is mainly used in the U.S.A. and Asia and was originally developed by Rockwell Automation. DeviceNet, ControlNet, and transmission control protocol/Internet protocol (TCP/IP) are open network technologies that share upper layers of the communication protocol, but are based on lower layers: DeviceNet is built on CAN, ControlNet on a token-passing bus protocol, and TCP/IP on Ethernet.

CANKingdom is a high-layer protocol used for motion control systems. It is also used in the maritime industry, as described in a boat example in Section 3. CANKingdom allows the changing of network behavior at any time, including while the system is running. For example, CANKingdom allows the system troubleshooter to turn off individual nodes. The CAN node identifiers and the triggering conditions for sending messages can be changed while the system is running. One instance when real-time network reconfiguration is used is during failure conditions. An example is a loss of a radio link ECU in a maritime application. The network monitor, also known as the King, in that case first shuts off the radio node to keep it from sending any more commands, and then tells the appropriate nodes to get data from the King. This operation eliminates the problem of a node receiving two simultaneous but conflicting commands. It also eliminates the problem of two nodes sending the same CAN id.

The high-level protocols described above have been developed with different applications and traditions in mind, which is reflected, for example, in their support for real-time control. Although SAE J1939 is used for implementing control algorithms, it does not provide explicit support for time-constrained messaging. In contrast, such functionalities are provided by CANKingdom and CANopen, which handle explicit support for inter-node synchronization. CANKingdom and CANopen allow static and dynamic configuration of the network, whereas SAE J1939 provides little flexibility.

CAN gateways

Gateways and bridges enable CAN-based networks to be linked together or linked to networks with other protocols. A gateway between a CAN and another communication network maps the protocols of the individual networks. There exist many different types of CAN gateways, e.g., CAN-RS232 and CAN-TCP/IP gateways. The latter can provide remote access to a CAN through the Internet, which allows worldwide monitoring and maintenance. The networks connected through a gateway or a bridge are disconnected in terms of their real-time behavior, so obviously the timing and performance of the complex inter-connected network can be hard to predict even if the individual networks are predictable.

Ethernet (or rather Ethernet/IP) is quite a different communication protocol compared to CAN, but is still of growing importance in industrial automation either in constellations with CAN or on its own. Traditionally, Ethernet is used in office automation and multimedia applications, while CAN dominates in vehicles and in certain industrial automation systems. The strength of Ethernet is the ability to quickly move large amounts of data over long distances and that the number of nodes in the network can be large. CAN, on the other hand, is optimized for transmitting small messages over relatively short distances. A drawback with a network based on the Ethernet protocol is that the nodes need to be quite powerful and complex (and therefore more expensive) in order to handle the communication control. Another drawback with Ethernet is that during network traffic congestion the delay jitter can be severe and unpredictable, although at low network load Ethernet gives almost no delay.

Time-triggered communication on CAN

Traditional CAN communication is event based: asynchronous events are triggered by node applications that initialize each transmission session. In many cases, this strategy is an efficient way to share the network resource. There are a variety of applications, however, that require a guaranteed access to the communication bus with a fixed periodic rate. This constraint is typical for sampled-data feedback control systems. In the automotive industry, x-by-wire

systems are examples of such control systems with deterministic communication behavior during regular operation.

By introducing the notion of global network time, the standard ISO 11898-4 defines the extension *Time-triggered communication on CAN* (TTCAN). It is built on top of the traditional event-triggered CAN protocol and enables existing CAN nodes to work in parallel with TTCAN nodes. The global clock requires hardware implementation; otherwise, TTCAN is a pure software extension of CAN. The synchronization in TTCAN takes place through a periodic reference message, which all TTCAN nodes recognize and use to synchronize their clocks. The nodes are configured to know when to send their message after the reference message. The period time of the transmission of a periodic node should be a multiple of the reference period. Traditional CAN nodes (or event-based TTCAN nodes) compete for the access of the free windows between the reference messages, along the line of the conventional CAN protocol. This mechanism is thus the reason why time-triggered and event-triggered scheduling is possible simultaneously in TTCAN.

The sender of the reference message is obviously a crucial node in TTCAN to guarantee clock synchronization. Therefore, an automatic procedure is provided for letting another node take over if the reference sender fails, and taking the reference back when the original clock master recovers. It is possible to use an external clock, for example, from the global positioning system (GPS).

2.3 A brief history

The evolution of microelectronics paved the way for introducing distributed control systems in vehicles. In the early 1980s there was, however, no low-cost and standardized protocol that was suitable for real-time control systems. Therefore, as we stated before, in 1983 Kiencke started the development of a new serial bus system at Bosch, which was presented as CAN in 1986 at the SAE congress in Detroit [8]. The development of CAN was mainly motivated by the need for new functionalities, but it also substantially reduced the need for wiring. The Bosch CAN Specification 2.0 was published in 1991 and then two years later the CAN protocol was internationally standardized as ISO 11898-1. The need for higher-layer protocols was recognized early. In 1991, CAN Kingdom was introduced by Kvaser. DeviceNet, another higher-layer protocol, was introduced by Allen-Bradley in 1994, and CANopen by CAN in Automation (CiA) in 1995. CiA is an international users and manufacturers group, which was founded in 1992. Mercedes-Benz has been using CAN in its passenger cars since 1992. Originally, CAN was used only for engine control, but today there are a variety of CAN nodes not only for powertrain and chassis control but also for body electronics and infotainment systems. Many other car manufacturers base their control architecture on CAN, including BMW, Fiat, Renault, SAAB, Volkswagen, and Volvo. The CAN architecture for a Volvo passenger car is described in the next section. The notion of time-triggered protocols for real-time systems was introduced by Kopetz and co-workers [10].

Time-triggered extensions of CAN were discussed in the late 1990s and early 2000s. This led to the standardization of TTCAN as ISO 11897-4 in 2004. Currently, there are intensive activities on utilizing TTCAN in a variety of vehicle applications.

3 Architectures

In this section, four vehicular examples of distributed control architectures based on CAN are presented. The architectures are implemented in a passenger car, a truck, a boat, and a spacecraft.

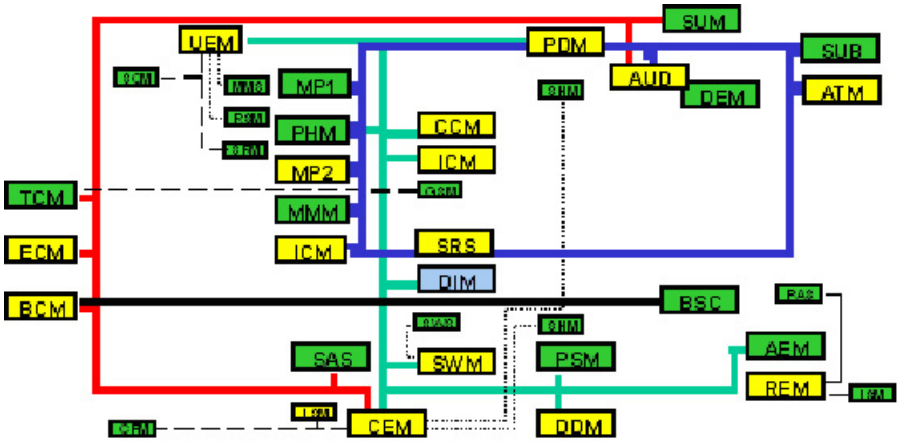
3.1 Volvo passenger car

In the automotive industry, there has been a remarkable evolution over the last few years in which embedded control systems have grown from stand-alone control systems to highly integrated and networked control systems. Originally motivated by reduced cabling and the specific addition of functionalities with sensor sharing and diagnostics, there are currently several new x-by-wire systems under development that involve distributed coordination of many subsystems.

Fig. 7 shows the distributed control architecture of the Volvo XC90. The blocks represent ECUs and the thick lines represent networks. The actual location of an ECU in the car is approximately indicated by its location in the block diagram. There are three classes of ECUs: powertrain and chassis, infotainment, and body electronics. Many of the ECU acronyms are defined in the figure. Several networks are used to connect the ECUs and the subsystems. There are two CAN buses. The leftmost network in the diagram is a CAN for power train and chassis subsystems. It connects for example engine and brake control (TCM, ECM, BCM, etc.) and has a communication rate of 500 kbps. The other CAN connects body electronics such as door and climate control (DDM, PDM, CCM, etc.) and has a communication rate of 125 kbps. The central electronic module (CEM) is an ECU that acts as a gateway between the two CAN buses. A media oriented system transport (MOST) network defines networking for infotainment and telematics subsystems. It consequently connects ECUs for multimedia, phone, and antenna. Finally, local interconnect networks (LINs) are used to connect slave nodes into a subsystem and are denoted by dashed lines in the block diagram. The maximum configuration for the vehicle contains about 40 ECUs [7].

3.2 Scania truck

There are several similarities between the control architecture in passenger cars and trucks. There are also many important differences, some of which

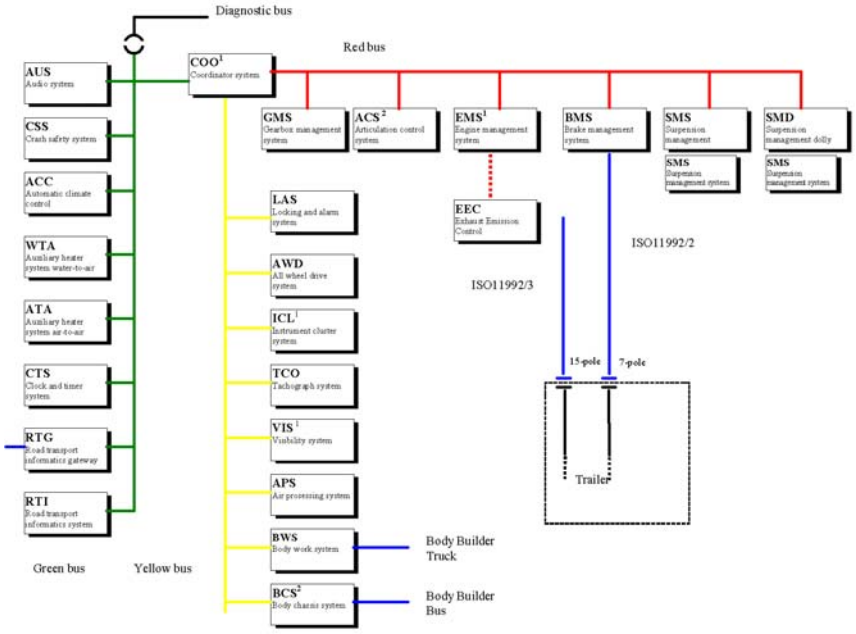


	Powertrain and chassis		Body electronics
TCM	Transmission control module	CEM	Central electronic module
ECM	Engine control module	SWM	Steering wheel module
BCM	Brake control module	DDM	Driver door module
BSC	Body sensor cluster	REM	Rear electronic module
SAS	Steering angle sensor	SWM	Steering wheel module
SUM	Suspension module	DDM	Driver door module
AUD	Audio module	PDM	Passenger door module
		REM	Rear electronic module
	Infotainment/Telematics	CCM	Climate control module
MP1,2	Media players 1 and 2	ICM	Infotainment control
PHM	Phone module	UEM	Upper electronic module
MMM	Multimedia module	DIM	Driver information module
SUB	Subwoofer	AEM	Auxiliary electronic
ATM	Antenna tuner module		

Fig. 7. Distributed control architecture for the Volvo XC90. Two CAN buses and some other networks connect up to about 40 ECUs. (Courtesy of Volvo Car Corporation.)

are due to the fact that trucks are configured in a large number of physical variants and have longer expected life times. These characteristics impose requirements on flexibility with respect to connecting, adding, and removing equipments and trailers.

The control architecture for a Scania truck is shown in Fig. 8. It consists of three CAN buses, denoted green, yellow, and red by Scania due to their relative importance. The leftmost (vertical) CAN contains less critical ECUs such as the audio system and the climate control. The middle (vertical) CAN handles the communication for important subsystems that are not directly involved in the engine and brake management. For example, connected to this



	Green bus		**Yellow bus**
AUS	Audio system	LAS	Locking and alarm system
CSS	Crash safety system	AWD	All wheel drive system
ACC	Automatic climate control	ICL	Instrument cluster system
WTA	Auxiliary heater water-to-air	TCO	Tachograph system
ATA	Auxiliary heater air-to-air	VIS	Visibility system
CTS	Clock and timer system	APS	Air processing system
RTG	Road transport info gateway	BWS	Body work system
RTI	Road transport info system	BCS	Body chassis system

	Red bus		
GMS	Gearbox management system	COO	Coordinator system
ACS	Articulation control system		
EMS	Engine management system		
EEC	Exhaust emission control		
BMS	Brake management system		
SMS	Suspension management system		
SMD	Suspension management dolly		

Fig. 8. Distributed control architecture for a Scania truck. Three CAN buses (denoted green, yellow, and red due to their relative criticality) connect up to more than twenty ECUs. The coordinator system ECU (COO) is a gateway between the three CAN buses. (Courtesy of Scania AB.)

bus is the instrument cluster system. Finally, the rightmost (horizontal) bus is the most critical CAN. It connects all ECUs for the driveline subsystems. The coordinator system ECU (COO) is a gateway between the three CAN buses. Connected to the leftmost CAN is a diagnostic bus, which is used to collect information on the status of the ECUs. The diagnostic bus can thus be used for error detection and debugging. Variants of the truck are equipped with different numbers of ECUs (the figure illustrates a configuration close to maximum). As for passenger cars, there are also subnetworks, but these are not shown in the figure.

SAE J1939 is the dominant higher-layer protocol for trucks. It facilitates plug-and-play functionality, but makes system changes and optimization difficult, partly because the priorities for scheduling the network traffic cannot be reconfigured. Manufacturers are using loopholes in SAE J1939 to work around these problems, but their existence indicates deficiencies in the protocol.

3.3 US Navy boat

There are several maritime applications of CAN. Here we give an example on unmanned seaborne targets provided by the United States Navy. The Navy has developed a distributed electronics architecture denoted SeaCAN, which is installed in all new seaborne targets and has been retrofitted into a number of older targets. A SeaCAN architecture for a 7 m remotely controlled rigid-hull inflatable boat is shown in Fig. 9. The system implements, for example, an autopilot based on a feedback control loop closed over the network. It involves the nodes Rudder Feedback, GPS Receiver, Pitch/Roll/Heading, Command/Control, and the two engine throttle nodes. The SeaCAN system uses a number of CPU boards with Infineon C167 microcontrollers connected together via a CAN bus running at 125 kbps. The lower communication rate is chosen to allow longer runs of copper and fiber suitable for larger boats and ships.

The SeaCAN system utilizes an operating system built around the CAN bus and based on the higher-layer protocol CANKingdom. The operating system contains a scheduler for tasks, which is synchronized with a higher-layer implementation of a global clock. It is thus possible to have coordinated behavior between two nodes, without any extra network communication. This functionality is used for generating periodic sampling, used in, e.g., the rudder servo control loop. In that case, the rudder sensing node samples the rudder angle at a rate of 10 Hz. The reception of the data messages from the rudder sensing node at the rudder actuator controller triggers the control loop routine. Because these messages have high priority on the bus and they are clocked out at a known rate, the control loop variability and data delay are very low, which thus enables a well-performing control loop.

Characteristics of SeaCAN include low usage of bandwidth in the order of 5%, global clock with a resolution of about 100 μsec, and special provisions with respect to safety and fail-safe shut-down. Network scheduling based on

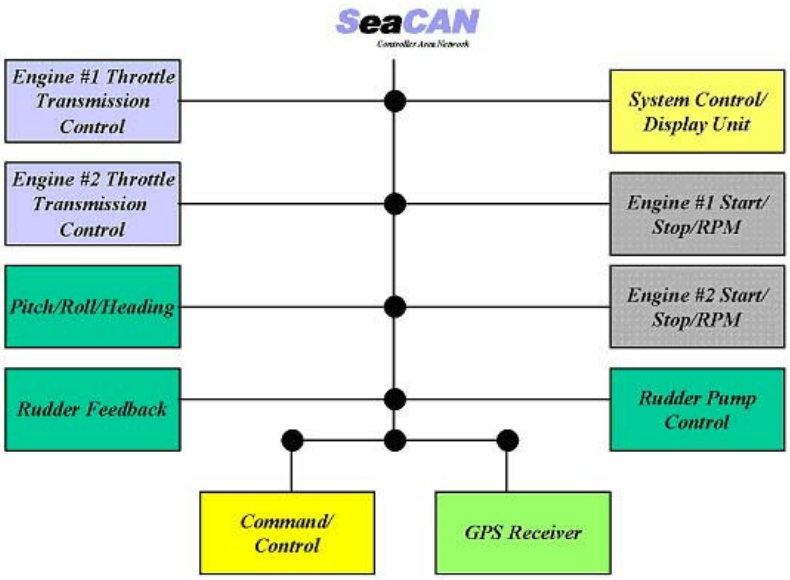


Fig. 9. Distributed control architecture for a boat. The block diagram shows a SeaCAN system for a 7 m remotely controlled rigid-hull inflatable boat. (Courtesy of the US Navy.)

the global clock is used to enforce a mixture of time-triggered and event-triggered communication.

3.4 SMART-1 spacecraft

The CAN protocol is also used in spacecraft and aircraft. SMART-1 is the first European lunar mission, where the acronym stands for "small missions for advanced research in technology." The spacecraft was successfully launched on September 27, 2003 by the European Space Agency on an Ariane V launcher. The Swedish Space Corporation was the prime contractor for SMART-1 and has developed several of the on-board subsystems including the on-board computer, avionics, and the attitude and orbit control system [2]. The main purpose of SMART-1 is to demonstrate the use of solar-electric propulsion in a low-thrust transfer from earth orbit into lunar orbit. The spacecraft carries several scientific instruments, and scientific observations are to be performed on the way to and in its lunar orbit. Currently (October 2004), SMART-1 is preparing for the maneuvers that will bring it into orbit [5].

Part of the distributed computer architecture of SMART-1 is presented in Fig. 10. The block diagram illustrates the decomposition of the system into two parts: one subsystem dedicated to the SMART-1 control system and

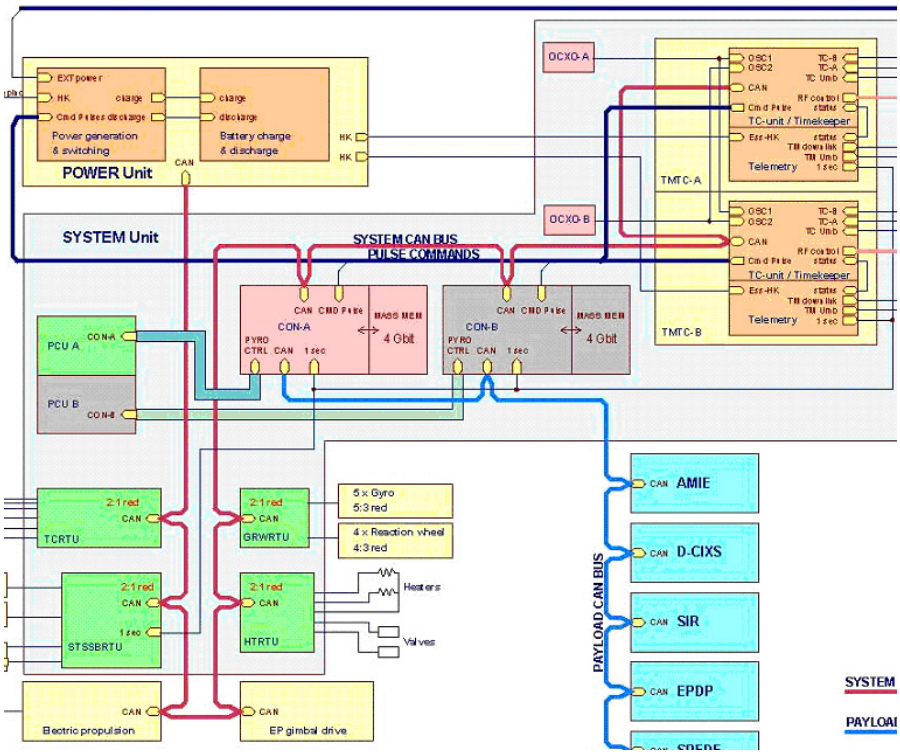


Fig. 10. Part of the distributed control architecture for the SMART-1 spacecraft. The system has two CAN buses: one for the control of the spacecraft and one for the payload. The spacecraft controllers are redundant and denoted CON-A and CON-B in the middle of the block diagram. (Courtesy of the Swedish Space Corporation.)

another for scientific experiments. Each of them is using a separate CAN, so there is one system CAN and one payload CAN. The spacecraft control is performed by the redundant controllers CON-A and CON-B, in the middle of the figure. Most control loops are closed over the system CAN with CAN nodes providing sensing and actuation capabilities. CAN nodes on the system bus include not only the spacecraft controller, but also nodes for telemetry and telecommand (earth communication), thermal control, star tracker and sun sensors, gyro and reaction wheels, hydrazine thruster, power control and distributions, and electronic propulsion and orientation.

Several provisions have been taken to ensure system robustness. All nodes are redundant; some in an active, others in a passive fashion. Each CAN bus has one nominal and one redundant communication path. A strategy and a hierarchy for error detection, redundancy management, and recovery have been defined. The spacecraft controller can take the decision to switch over to the redundant bus (but not back again). Most other nodes will check

for the life sign message from the spacecraft controller. If the life sign is not available, the nodes will attempt to switch to the other bus. The power unit has highest authority in the autonomy hierarchy and will switch to the redundant spacecraft controller if the primary controller is considered to have failed. The power unit can also control the activation and deactivation of the other nodes. As a last means for recovery, ground can intervene manually.

Radiation tolerance and the detection of radiation-induced errors are crucial for SMART-1. It was not possible to use ordinary CAN controllers because they are not radiation tolerant. Instead, the license to use the VHDL-code for CAN was purchased from Bosch and was used to implement a CAN controller in a radiation-tolerant field programmable gate array (FPGA). Some other features were added to the CAN protocol simultaneously, such as specific error detection and error handling mechanisms. In addition, clock synchronization was added, so that the resolution of the global clock became better than 1 msec. Effects due to radiation can still cause corrupted frames. Therefore, one of the identifier bits was chosen to be used as an extra parity bit for the identifier field.

4 Control Applications

Two vehicular control systems with loops closed over CAN buses are discussed in this section. The first example is a vehicle dynamics control system for passenger cars that is manufactured by Bosch. The second example is an attitude and orbit control system for the SMART-1 spacecraft discussed in the previous section.

4.1 Vehicle dynamics control system

Vehicle dynamics control[4] systems are designed to assist the driver in over-steering, under-steering and roll-over situations [15, 9]. The principle of a vehicle dynamics control (VDC) system is illustrated in Fig. 11. The left figure shows a situation where over-steering takes place, illustrating the case where the friction limits are reached for the rear wheels causing the tire forces to saturate (saturation on the front wheels will instead cause an under-steer situation). Unless the driver is very skilled, the car will start to skid, meaning that the vehicle yaw rate and vehicle side slip angle will deviate from what the driver intended. This is the situation shown for the left vehicle. For the vehicle on the right, the on-board VDC will detect the emerging skidding situation and will compute a compensating torque, which for the situation illustrated is translated into applying a braking force to the outer front wheel. This braking force will provide a compensating torque and the braking will also reduce the lateral force for this wheel.

[4] Also known as electronic stability program, dynamic stability control, or active yaw control.

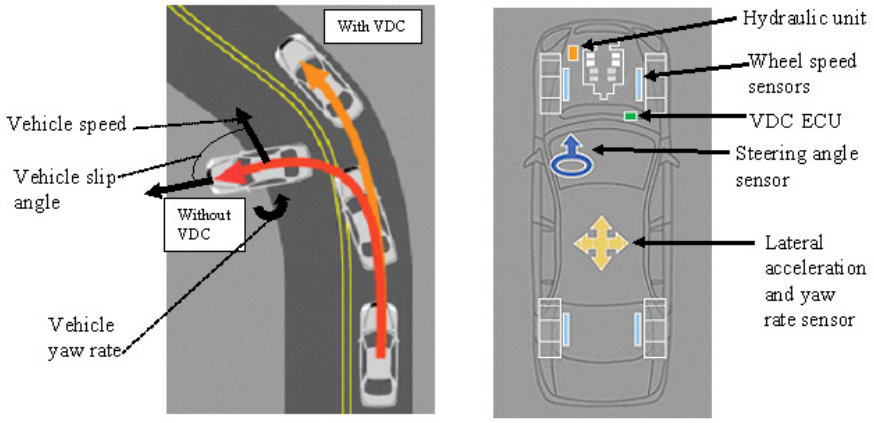


Fig. 11. Illustration of behavior during over-steering for vehicle with and without VDC system (left figure). Central components of VDC (right figure). (Based on figures provided by the Electronic Stability Control Coalition.)

The VDC system compares the driver's estimated intended course, by measuring the steering wheel angle and other relevant sensor data, with the actual motion of the vehicle. When these deviate too much, the VDC will intervene by automatically applying the brakes of the individual wheels and also by controlling the engine torque, in order to make the vehicle follow the path intended by the driver as closely as possible. The central components of VDC are illustrated on the right in Fig. 11. In essence, the VDC will assist the driver by making the car easier to steer and by improving its stability margin. See [9] for details.

A block diagram of a conceptual VDC is shown in Fig. 12. The cascade control structure consists of three controllers: (1) the yaw/slip controller, which controls the overall vehicle dynamics in terms of the vehicle yaw rate and the vehicle side slip angle; (2) the brake controller, which controls the individual wheel braking forces; and (3) the engine controller, which controls the engine torque. The inputs to the yaw/slip controller include the driver's commands: accelerator pedal position, steering wheel angle, and brake pressure. Based on these inputs and other sensor data, nominal values for the yaw rate and the vehicle side slip are computed. They are compared with the measured yaw rate and the estimated side slip. A gain-scheduled feedback control law is applied to derive set-points for the engine and brake controllers; for example, during over-steering, braking actions are normally performed on the front outer wheel and for under-steering normally on the rear inner wheel. The gains of the controllers depend on the driving conditions (e.g., vehicle speed, under-steering, over-steering). The brake and the engine controllers are typically proportional-integral-derivative (PID) controllers and also use local sensor information such as wheel speed. The VDC system has to take

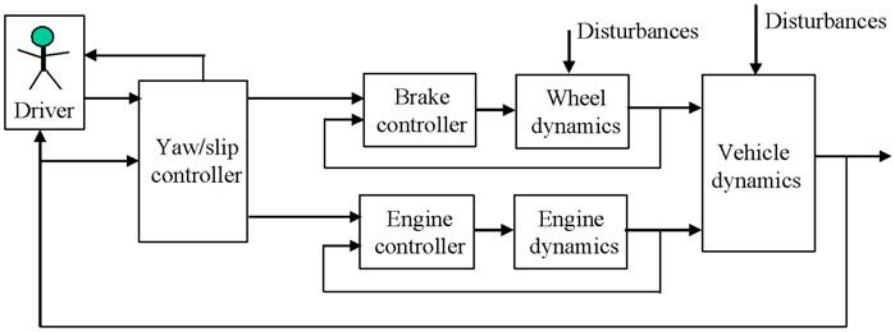


Fig. 12. Cascade control structure of VDC system

the driver behavior into account as well as disturbances acting on the vehicle, including cross-wind, asymmetric friction coefficients, and even a flat tire.

The VDC system utilizes the CAN bus, as it is depending on several ECUs, although the main functionality resides in a specific ECU. The implementation strongly depends on the choice of braking mechanics (e.g., hydraulics, pneumatics, electro-hydraulics, or even electro-mechanics), the availability of a transmission ECU, and the interface to the engine ECU. A separate distributed control system is often used for the brakes, extending from a brake control node; for example, trucks often have one ECU per wheel pair and an additional controller for the trailer. Since some of the control loops of a VDC system are closed over a vehicle CAN, special care has to be taken with respect to end-to-end delays and faults in the distributed system.

4.2 Attitude and orbit control system

This section describe parts of the SMART-1 attitude and orbit control system and how it is implemented in the on-board distributed computer system [2]. The control architecture and the CAN buses of SMART-1 were described in Section 3. The control objectives of the attitude and orbit control system are to

- follow desired trajectories according to the goals of the mission,
- point the solar panels toward the sun, and
- minimize energy consumption.

The control objectives should be fulfilled despite the harsh environment and torque disturbances acting on the spacecraft, such as aero drag (initially when close to earth), gravitational gradient, magnetic torque, and solar pressure (mechanical pressure from photons). There are several phases that the control system should be able to handle, including the phase just after separation from the launcher, the thrusting phases on the orbit to the moon, and the moon observation phase.

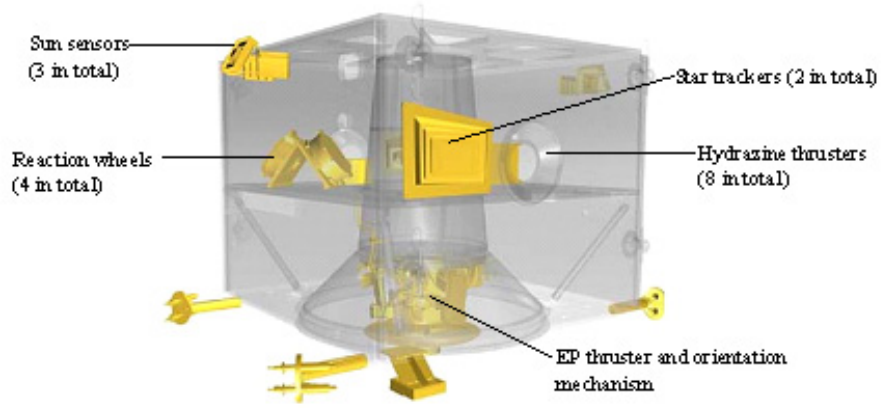


Fig. 13. Structure of SMART-1 spacecraft with sensors and actuators for the attitude and orbit control system. (Courtesy of the Swedish Space Corporation.)

The sensors and actuators used for controlling the spacecraft's attitude are illustrated in Fig. 13. Sensors are a star tracker and solid-state angular rate sensors. The star tracker provides estimates of the sun vector. It has one nominal and one redundant processing unit and two hot redundant camera heads, which can be operated from either of the two processing units. Five angular rate sensors are included to allow for detection and isolation of a failure in a sensor unit. The rate sensors can provide estimates of spacecraft attitude during shorter outages of attitude measurements from the star tracker. Actuators for the attitude control are reaction wheels and hydrazine thrusters. There are four reaction wheels aligned in a pyramid configuration based on considerations of environmental disturbances and momentum management. The angular momentum storage capability is 4 Nms per wheel with a reaction torque above 20 mNm. The hydrazine system consists of four nominal and four redundant 1 N thrusters.

The attitude and orbit control system consists of a set of control functions for rate damping, sun pointing, solar array rotation, momentum reduction, three-axis attitude control, and electric propulsion (EP) thruster orientation. The system has a number of operation modes, which consist of a subset of these control functions. The operation modes include the following:

- *Detumble mode:* In this mode, rotation is stabilized using one P-controller per axis with the aid of the hydrazine thrusters and the rate sensors.
- *Safe mode:* Here the EP thruster is pointed toward the sun and set to rotate one revolution per hour around the sun vector. The attitude is controlled using a bang-bang strategy for large sun angles and a PID controller for smaller angles. Both controllers use the reaction wheels as actuators and the sun tracker as sensor. The spacecraft rotation is controlled using a

PI controller. When the angular velocity of the reaction wheels exceeds a certain limit, their momentum is reduced by use of the hydrazine thrusters.

- *Science mode:* In this mode, ground provides the attitude set-points for the spacecraft and the star tracker provides the actual attitude. The reaction wheels and the hydrazine thrusters are used.
- *Electric propulsion control mode:* This mode is similar to the science mode apart from the additional control of the EP orientation mechanism. This mechanism can be used to tilt the thrust vector in order to off-load the reaction wheel momentum about the two spacecraft axes that form the nominal EP thrust plane. This reduces the amount of hydrazine needed. The EP mechanism is controlled in an outer and slower control loop (PI) based on the speed of the reaction wheels and the rotation of the spacecraft body.

Let us describe the Science mode in some detail. Fig. 14 shows a block diagram of the attitude control system in Science mode. As was described for the Safe mode above, different controllers are activated in the Science mode depending on the size of the control error. This switching between controllers is indicated by the block "Mode logic" in the figure. Anti-windup compensation is used in the control law to prevent integrator windup when the reaction wheel commands saturate. Also, as in the Safe mode, the hydrazine thrusters are used to introduce an external momentum when the angular momentum of the reaction wheels grows too high. The attitude control works independently of the thruster commanding. The entire control system is sampled at 1 Hz. The time constant for closed-loop control is about 30 sec for the Science mode (and 300 sec for the Safe mode). The estimation block in Fig. 14 provides filtering of signals such as the sun vector and computes the spacecraft body rates. It includes a Kalman filter with inputs from the star tracker and the rate sensors. The main purpose is to provide estimates of the attitude for short periods when the star tracker is not able to deliver the attitude, for example, due to blinding of the sensor camera heads. The control algorithms of the attitude and orbit control system reside in the spacecraft controllers of the control architecture depicted in Fig. 10. With a period of 1 sec, the spacecraft controller issues polling commands over the CAN to the corresponding sensors, including the gyros and the sun sensors. When all sensor data are received, the control commands are computed and then sent to the actuators, including the reaction wheels and the hydrazine thrusters. The maximum normal utilization of the CAN is about 30%, but under heavy disturbance, due to retransmission of corrupted messages, it rises to about 40%. The total communication time for communication over the CAN network for attitude and orbit control sensing and actuating data is approximately 12 msec. It is thus small compared to the sampling period.

The on-board software can be patched by uploading new software from ground during operation in space. So far this has been carried out once for the star tracker node. The need arose during a very intensive solar storm, which

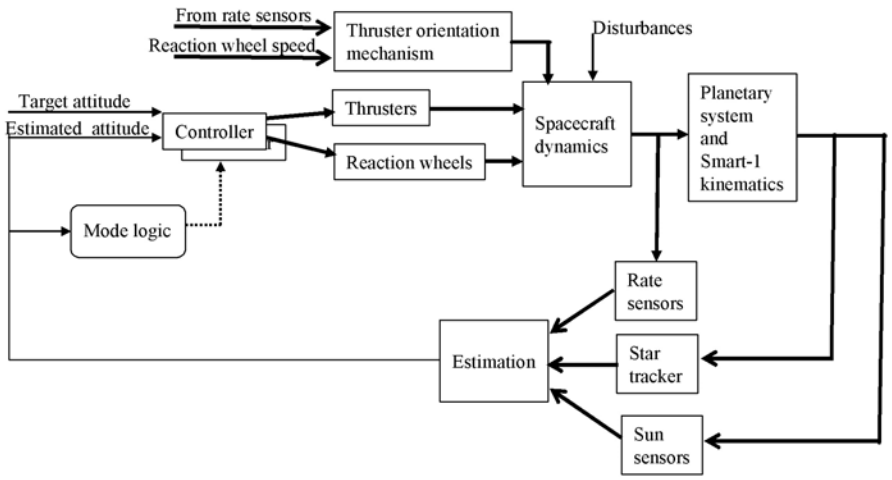


Fig. 14. Attitude control system under operation in Science mode. Disturbances include gravity, particles, and aero drag affecting the spacecraft.

necessitated modification of software filters to handle larger disturbance and noise levels than had been anticipated.

During operation, the system platform software is responsible for detection of failing nodes and redundancy management. The control application is notified of detected errors such as temporary unavailability of data from one or more nodes. If the I/O nodes do not reply to poll messages for an extended period of time, the redundancy management will initiate recovery actions, including switching to the redundant slave nodes and attempting to use the redundant network.

5 Perspectives

The development of vehicles is going through a dramatic evolution, in their transition from pure mechanical systems to mechatronic machines with highly integrated hardware and software subsystems. DaimlerChrysler estimates that 90% of the innovations in the automotive area lie in electronics and software. A challenge in the development of vehicular embedded control systems is safety and real-time requirements. The control systems are increasingly being implemented in distributed computer systems and require a multitude of competences to be developed and integrated to meet quality requirements in a cost-efficient way. A major research problem is to develop techniques and tools to bridge the gap between functional requirements and the final design. In this section, we describe two particular trends in the vehicular networked embedded systems, namely, brake-by-wire and other x-by-wire systems, and

standardized platforms and open-ended architectures for distributed control units in vehicles.

5.1 X-by-wire systems

X-by-wire is a term for the addition of electronic systems to the vehicle to improve tasks that were previously accomplished purely with mechanical and hydraulic systems. Examples of x-by-wire systems are steer-by-wire and brake-by-wire, where steering and braking data, respectively, are communicated electronically from the driver to the actuators. For a brake-by-wire control system, it is common that the information transfer from the brake pedal to the braking actuator is handled electronically, but that the actuators are hydraulic or pneumatic. Sometimes the actuators are replaced with electrical motors, so that a full brake-by-wire system is created.

An early x-by-wire system is the fly-by-wire application in the Airbus aircraft A310, which has been in use since 1983. Airbus 320, which was certified in 1988, is the first aircraft that depends entirely on x-by-wire control. Examples of early x-by-wire systems in the automotive industry include automated manual transmission, which eliminates the mechanical connection to the transmission, so that the gears can be chosen manually by pushing buttons or automatically by a computer that runs a gear selection program. The technology, which was first developed for motor sports to relieve the driver from using the clutch, is available from many passenger car manufacturers including Alfa Romeo, BMW, Mercedes-Benz, Porsche and Volkswagen. Several solutions are also available for heavy vehicles, under names like I-Shift from Volvo Trucks and Opticruise from Scania. In the marine industry, a recent x-by-wire system is the throttle-by-wire system available from Mercury Marine, which is based on dual redundant CAN buses operating at 250 kbps.

Challenges for further applications of x-by-wire systems in the automotive industry are legislation, safety demands, cost efficiency, and user expectations. It should be noted that x-by-wire systems in cars are not the same as fly-by-wire systems. Differences include the sensitivity to frequency of failure in operations, and the cost sensitivity. Consequently, the technological concepts used for by-wire systems in airplanes are not necessarily cost efficient for the automotive industry.

5.2 Standardized software architectures

Standards in the automotive industry have been developed for communication, but the other parts of the distributed control systems of cars have mainly remained closed; for example, note that most automotive systems use proprietary real-time operating systems. The lack of existing standardization has generated research projects to standardize diagnostics and measurement systems, description languages, and software platforms. Two related initiatives focused on software platforms are OSEK/VDX [12] and AUTOSAR [1].

OSEK/VDX is a joint project of the automotive industry that aims at an industry standard for an open-ended architecture for distributed control units in vehicles. OSEK stands for *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug* (open systems and the corresponding interfaces for automotive electronics) and was founded by the German automotive industry. VDX, which is an acronym for vehicle distributed executive, was originally defined as part of a joint effort by the French companies PSA and Renault. In 1995 the OSEK/VDX group presented their first results of a specification. A goal of OSEK/VDX is to support the portability and reusability of application software. The open architecture defines three substandards of communication, network management, and operating system. It includes data exchange within and between ECUs, network configuration and monitoring, and real-time executive for ECU software. The basic idea of OSEK/VDX is to define standardized services, so that the costs are reduced to maintain and port application software. Obviously, by imposing portability, it should be possible to transfer application software from one ECU to another ECU. The application software module can have several interfaces. For example, there exist interfaces to the operating system for real-time control, but there also exist interfaces to other software modules. The interfaces should be rich enough to represent a complete functionality in a system. An OSEK/VDX implementation language has also been defined. It supports a portable description of all OSEK/VDX specific objects such as tasks and alarms. It remains to be seen whether the OSEK/VDX effort will be successful or not.

Automotive Open System Architecture (AUTOSAR) is a development partnership with the goal of standardizing basic system functions and functional interfaces in vehicles. The initiative is an indication of the difficulties faced today to fulfill the growing passenger and legal requirements, despite the increased system complexity. There needs to be a clear notion of how software components should be specified and integrated in automotive applications. The AUTOSAR standard is intended to serve as a platform upon which future vehicle applications could be implemented. The AUTOSAR project plan was released in 2003, so extensive test and verification remain to be done before AUTOSAR can be used in practice.

Acknowledgements

We would like to express our gratitude to the individuals and the companies that provided information on the examples described in this chapter. In particular, Per Bodin and Gunnar Andersson, Swedish Space Corporation, are acknowledged for providing information on the SMART-1 spacecraft. Dave Purdue, US Navy, is acknowledged for the description of SeaCAN. Jakob Axelsson, Volvo Car Corporation, is acknowledged for information on the Volvo XC90. Ola Larses, Scania AB, is acknowledged for information on the Scania truck. Lars-Berno Fredriksson is acknowledged for general advice on CAN.

References

1. http://www.autosar.org, 2004. Homepage of the development partnership Automotive Open System Architecture (AUTOSAR).
2. P. Bodin, S. Berge, M. Björk, A. Edfors, J Kugelberg, and P. Rathsman. The SMART-1 attitude and orbit control system: Flight results from the first mission phase. In *AIAA Guidance, Navigation, and Control Conference*, number AIAA-2004-5244, Providence, RI, 2004.
3. http://www.can-cia.de, 2004. Homepage of the organization CAN in Automation (CiA).
4. CAN specification version 2.0. Robert Bosch GmbH, Stuttgart, Germany, 1991.
5. http://www.esa.int/SPECIALS/SMART-1, 2004. Homepage of the SMART-1 spacegraft of the European Space Agency.
6. K. Etschberger. *Controller Area Network: Basics, Protocols, Chips and Applications*. IXXAT Automation GmbH, Weingarten, Germany, 2001.
7. J. Fröberg, K. Sandström, C. Norström, H. Hansson, J. Axelsson, and B. Villing. A omparative case study of distributed network architectures for different automotive applications. In *Handbook on Information Technology in Industrial Automation*. IEEE Press and CRC Press, 2004.
8. U. Kiencke, S. Dais, and M. Litschel. Automotive serial controller area network. In *SAE International Congress No. 860391*, Detroit, MI, 1986.
9. U. Kiencke and L. Nielsen. *Automotive Control Systems*. Springer-Verlag, Berlin, 2000.
10. H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Dordrecht, 1997.
11. G. Leen and D. Heffernan. Expanding automotive electronic systems. *Computer*, 35(1):88–93, Jan 2002.
12. http://www.osek-vdx.org, 2004. Homepage of a joint project of the automotive industry on a standard for an open-ended architecture for distributed control units in vehicles.
13. M. Törngren. A perspective to the design of distributed real-time control applications based on CAN. In *2nd International CiA CAN conference*, London, U.K., 1995.
14. M. Törngren, K. H. Johansson, G. Andersson, P. Bodin, and D. Purdue. A survey of contemporary embedded distributed control systems in vehicles. Technical Report ISSN 1400-1179, ISRN KTH/MMK-04/xx-SE, Dept. of Machine Design, KTH, 2004.
15. A. T. van Zanten, R. Erhardt, K. Landesfeind, and G. Pfaff. VDC systems development and perspective. In *SAE World Congress*, 1998.