PROTOTYPING A FAULT-TOLERANT CAN BUS BASED DISTRIBUTED SERVOSYSTEM

Michael Skipper Andersen, Jørgen Friis, Niels Nørregård Hansen, Johnny Jensen,
Rene Just Nielsen and Michael Pedersen, Aalborg University
Department of Control Engineering, group 731

***Abstract*— In conventional car power steerings the steering arms on the hinged wheels are mechanically connected by a track rod. When the steering wheel is turned the rotation from the steering column, assisted by a hydraulic servo-cylinder moves the track rod sideways, turning both wheels. This construction has the advantage of being simple and cheap. It is, however, quite space consuming and vibrations from the wheels are transmitted through the steering column to the steering wheel. This raises the idea of a distributed servosystem, utilizing electric actuators to position the wheels and for force feedback to the steering wheel. The objective of this paper is to present and review the prototype of a duplicated Controller Area Network (CAN) bus based fault-tolerant distributed power steering, especially considering the pitfalls of such a system.**

A prototype of a distributed servosystem has been tested with different scenarios and fault cases. The node connections and fault tolerance work satisfactorily and acts as expected, but the actuators have proven to be underdimensioned.

***Keywords*— distributed systems, power steering, CAN bus, fault tolerance, redundancy, robustness.**

I. Introduction

CAR steering gear is often equipped with a hydraulic servosystem to assist the driver turning the steering wheel by adding extra torque to the steering column. This aid is particularly useful when a vehicle moves at low speeds and much torque must be used to move the wheels.

The key components in a power steering are a torsion bar that measures the torque applied on the steering wheel by the driver, a pump that generates the hydraulic power and a rotary valve that directs the pressurized hydraulic fluid to the right side of a piston on the track rod. Together, these components make up a reliable and relatively simple system. If one part fails to work, the steering gear can stil function without the servomechanism.

The two main drawbacks of such a system are that vibrations from the wheels are transfered to the steering wheel and that the track rod and the steering column occupy space across the engine compartment.

This suggests the concept of a distributed steering gear with independent actuators to position the vehicle's wheels and a steering wheel to dictate the positions of these. To make the distributed steering gear act like a mechanical one, the steering wheel should have a force-feedback behaviour to help the driver sense forces applied to the wheels, e.g. if the vehicle bumps into a kerb.

In such a system, special care should be taken to ensure robustness so that the correlation between the steering wheel and the wheel position is maintained at all times. For that reason, it is evident that actuator positions must always be computed correctly and that communication between the network nodes attached to the three actuators should always be possible even in the case of a partial breakdown of the network.

II. Methods And Material

The main considerations made in connection with the developed prototype of a distributed power steering were the maintenance of the connection between the three network nodes and the reliability of correctly computed values in each node.

A model of a power steering has been constructed. It consists of a modified off-the-shelf PC force feedback steering wheel and two electric servomotors, each representing a wheel actuator.

Each network node is based on two Peripheral Interface Controllers (PICs) (PIC18F458) with integrated pulse-width modulator (PWM), analog-to-digital converter (ADC), serial peripheral interface (SPI), RS232 serial interface, and CAN controller.

The CAN bus with CAN protocol version 2.0A (using 11 bit identifiers to address the network nodes) was chosen as the communication channel between the network nodes. It features a good quality of service (QoS) in terms of packet loss, high noise immunity, relative throughput (that is, a high priority packet will always win the bus arbitration over packets with lower priorities), and automatic retransmission of erroneous packets. Moreover, if one CAN controller repeatedly transmits or receives defective packets, it puts itself in

a bus-off state and does not send anything on the CAN bus until 128 occurrences of 11 consecutive recessive bits have been registered on the bus.

to ensure that a fault can always be announced on the bus and so that the nodes can take proper action in accordance to this alarm, alarm frames are given the highest priority on the CAN bus.

To preserve the network communication between the network nodes even in case of a cable breakdown, the CAN bus and its contents have been duplicated. The double cabling should be wired in different places to minimize the possibility of breaking both cables simultaneously.

Likewise, the two PICs on each network node form a redundant system, intercommunicating via an SPI and with either PIC connected to one of the two CAN lines. The intercommunication ensures that both use the same data and run synchronously through the software.

One PIC is configured as a master and the other as a hot-standby slave. The master controls a multiplexer, selecting which PIC should control the motor. If one PIC fails to work correctly the other one should take over the motor control and continue.

The system is illustrated in figure 1 where the “CAN packet sniffer” is a standard Personal Computer (PC) registering all CAN messages on the bus and saving these in a log file.

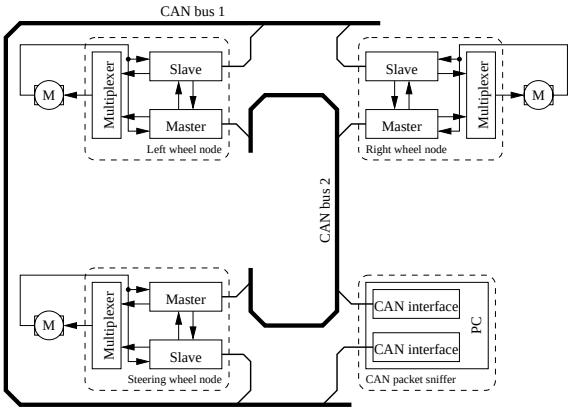

Fig. 1. Distributed architecture of power steering system with redundant PICs and CAN bus.

To delimit the problem, the fault-tolerant system should be able to handle Single Point of Failure (SPF) situations only. This assumes that only one error will occur at a time. Moreover, the following assumptions have been made.

- The SPI communication is error-free.
- Simple components such as CAN drivers, multiplexers and position feedback potentiometers are very unlikely to fail.
- Position feedback from the actuators is always available.
- Supply voltage is always present and adequate.
- An interrupt request can always be served if the program runs into a deadlock.

A. Steering System Functionality

To simplify the system, the Ackermann steering geometry normally used in vehicles [5] has been omitted in the prototype design, and both wheel nodes (WNs) are given the same angular reference by the steering wheel node (SWN).

The SWN broadcasts a reference value to the WNs 40 times per second, and when they have intercepted this reference and attempted to adjust the actuators to this position, they return their measured wheel angle to the SWN.

The functionality of the system can be classified into four cases.

1) If the measured angles of the WNs and the SWN are equal, nothing is done.
2) If the measured angles of the WNs are equal but different from the angle of the SWN, this angle is used as a reference for the WNs.
3) If the measured angles of WN1 and the SWN are equal but are different from the measured angle of WN2, the SWN and WN1 uses the angle of WN2 as a reference angle, vice versa if the angles of SWN and WN2 are equal.
4) If all three angles differ, the SWN and WN1 will attempt to adjust their actuators to the position of WN2 over 5 samples. If they do not succeed, the SWN and WN2 will attempt to track to the position of WN1.

In case one, both wheels have the desired position and the previous valid reference values are re-applied to the actuator.

Case two represents the situation where the driver turns the stering wheel to alter the direction of the vehicle.

In case three the differing wheel angle indicates that the wheel in question is forced into a certain position, e.g. if it bumps into a kerb. The driver then senses this as the steering wheel, along with the other wheel, will attempt to change their positions.

If case four should happen, this, in a mechanical steering gear, would mean that both the steering column and the track rod are broken. In the distributed system this is allowed for up to 2×125 milliseconds because the three actuators are relaxed most of the time and thus susceptible to exterior forces. The only time when

voltage is applied to them is in the control loop part of the software.

In all four cases the term “equal” is not exactly used in the software. Instead, a small tolerance has been introduced to allow some numerical resilience.

The software on all three network nodes is implemented as a cyclic main loop, calling a number of functions 40 times per second.

B. Actuator Control

DC servomotors have been chosen as actuators for the wheels and steering wheel. To control the motor speeds, the PWM signals from the PICs have been fed through an H-bridge amplifier and applied to the motors. Each motor is fitted with a potentiometer for position feedback, and the output voltage has been fed into the ADC inputs of the respective master and slave PIC.

Since the master and the slave PIC on each node perform the same motor control signal computations, but only one at a time can actually control the motor attached to the node, a multiplexer is used to direct the control signal from one PIC to the motor. At the beginning of each main loop iteration, the master takes the motor control and, if during the course of iteration the master fails to work correctly, the slave takes over the motor control.

Since no supervisory arbiting circuit has been implemented to detect master/slave faults, the master and slave PIC themselves must decide which should have the motor control rights.

One solution to this problem has been to make a simple logic circuit allowing only one PIC at a time to *take* the control rights. This solution, however, poses the question of which PIC should have the right to overrule the other in case one of them makes a faulty attempt to take the control rights.

The chosen solution is to let the master control the multiplexer and, in case that the slave discovers a fault or missing response on the master, the slave sends an interrupt request to the master which hands over the motor control to the slave.

This solution is based on the assumption that it is always possible for the master to run an interrupt service routine (ISR), even when the main program has run into a deadlock. This is taken as a fact since the detection and handling of interrupts is handled on the hardware level of the PIC.

C. Fault Handling

A number of possible runtime faults have been taken into account in the prototype.

- Missing synchronization and communication between master and slave.
- The master and slave do not use the same data.

The PICs perform handshakes at certain program breakpoints to ensure that they both enter the same program segments simultaneously. If one PIC fails to respond in the handshaking procedure, the other one will invoke an interrupt request on the failing PIC, causing it to reset and go to the beginning of the main loop where it waits for the other one to start over in a new iteration. Moreover, when the master is interrupted as the result of an error, it hands over the motor control rights to the slave.

Both PICs on each network node run through approximately the same algorithms and to ensure that they use the same data (i.e. received CAN messages and measured actuator positions) throughout the program, they exchange these data at certain breakpoints. If the data differ, the master decides which are correct from the following criteria. If one PIC has no data, data from the other PIC will be used if they are valid. Otherwise, an alarm will be broadcasted on the CAN bus and the previous valid data are used. This is derived from the philosophy that it is better to use old data than no data. The data interchanged between the two PICs consist of two data bytes and one byte for flags. The flags indicate, among other things, which type of data is contained in the data byte and whether an error has occured.

D. Software

On each PIC a bootloader has been written and installed to make it easier to download software to the nodes during the prototyping process. The bootloader connects the PICs to the serial port of a PC utilizing their integrated RS232 interfaces.

When the PICs have been reset they run through an initialization procedure setting up necessary registers and handshaking with each other. The handshake between master and slave is important to mutually ensure that they both run. The handshake is carried out with the use of two dedicated input/output pins, Ready In (RDI) and Ready Out (RDO), on each PIC. When one of the PICs is ready to commence it asserts RDO and polls RDI forever or until the other PIC asserts its RDO.

Each WN measures the position of its attached actuator and use this as a reference value for its motor controller. Then they send these values on the CAN bus to the SWN and carry on to the main loop.

The SWN waits until it has received the actuator positions of the two WNs and then it starts its main loop. If it does not receive the position from one of the WNs, the main loop will not start.

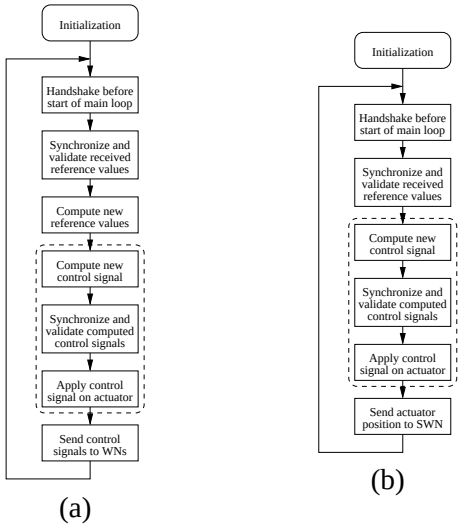

Fig. 2. Flow chart illustrating the main loop of the PIC software. (a) shows the software in the Steering Wheel Node (SWN) and (b) the software in the Wheel Nodes (WNs).

The flow chart in figure 2 illustrates the most important program segments of the main loop after the initialization.

First, the master and slave handshake to ensure that they start their main loop timers (the timers ensuring that the main loop is run through 40 times per second) simultaneously to run synchronously through the loop.

After this handshake the PICs synchronize and validate the contents of the CAN messages received from the other network nodes. The reception of a CAN message is registered and handled by an ISR on each PIC. The validation process is performed on the master. Only a certain change in reference value since the previous main loop iteration is allowed and the reference value must be within a certain numerical range. If both the master and slave data are valid but differ, the average value is used on both PICs.

The SWN then computes new reference values for the wheels and the steering wheel from an algorithm conforming to the abovementioned functionality of the distributed power steering.

The actuators at all three nodes are controlled with a Proportional-Integral-Derivative (PID) controller consisting of the blocks shown inside the dashed box on figure 2.

First, the required control signal is computed. Then the new control signal is synchronized and validated before it is applied to the actuator. The PID controlled actuators should be as fast as possible and not have any overshoot as this would cause the actuators to oscillate.

Finally, the WNs send their measured actuator positions to the SWN, and the SWN sends the new reference values to the WNs, and the main loop is run over again.

If a PIC fails to respond in one of the handshake procedures during the main loop, it will be reset by the other PIC which will finish the main loop alone. After the reset PIC has finished its initialization, it will wait for the other in the first of the handshake procedures.

III. Results

The software for the distributed power steering was tested to see if it fulfills the demands of functionality and robustness mentioned above.

During the prototype development the hardware and program segments have been tested individually and gradually incorporated and tested in the main program. Subsequently the system as a whole has been tested.

The test strategy was to invoke a number of faults one by one (in accordance with the SPF approach) and thus cause the system to fail. These faults were.

1) Partial and total breakdown of the CAN lines.
2) Software deadlocks in one of the PICs.
3) Wrong data measured by the ADCs or received on the CAN line.

The partial breakdown of a CAN line was simulated by unplugging one of the CAN bus connectors on a node and confirming that the steering gear still worked. This is illustrated in figure 3.

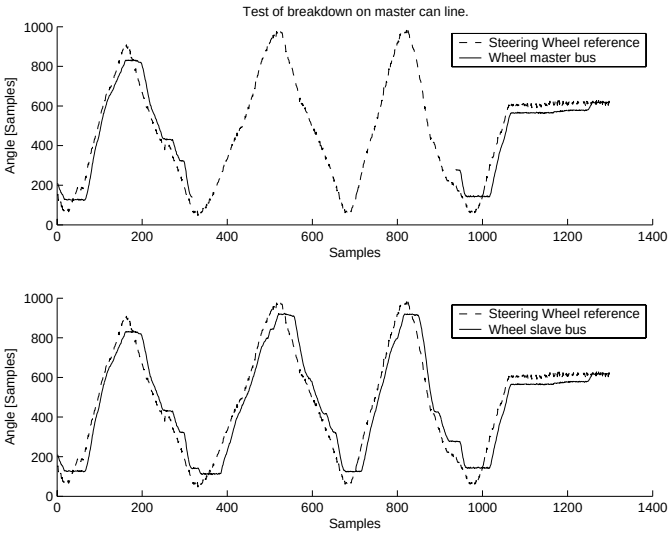

Fig. 3. Partly breakdown of the CAN lines. Each of the two figures show the reference values from the steering wheel node together with the measured positions from a wheel node on one CAN line. On the top graph one CAN line is disrupted while on the bottom graph the other CAN line remains intact.

A total CAN line breakdown was invoked by disconnecting the SWN from the CAN bus, resulting in the WN actuators and the steering wheel remained in the same position even when exposed to external forces.

A software deadlock was simulated by writing a function containing an infinite loop on one of the PICs and, in the beginning of the main loop, to read the status of an input port to which a switch was connected. Closing this switch would then cause the function to be called forcing a software deadlock to occur.

The three abovementioned tests gave satisfactory results and showed that the functionality of the system's fault handling works.

The handling of wrong data has been tested by letting one PIC overwrite the data measured by the ADC or received on the CAN bus with data outside the valid range. This proved to have no influence on the behaviour of the system.

In the same manner, handling of wrong data on both master and slave PIC was tested. Here, the actuators on the node in question kept their former valid positions.

Prior to the these fault tests, the overall functionality of the system was tested. It showed that the wheels ceased to follow the steering wheel when this was turned quickly. Therefore, a test to monitor the tracking abilities of the system was carried out. From the log file of the CAN packet sniffer, the reference value from the SWN and the actuator positions of the WNs were plotted against each other. This plot is shown in figure 4 where it can be seen that the tracking error is approximately 40 samples. The figure also shows that one of the actuators

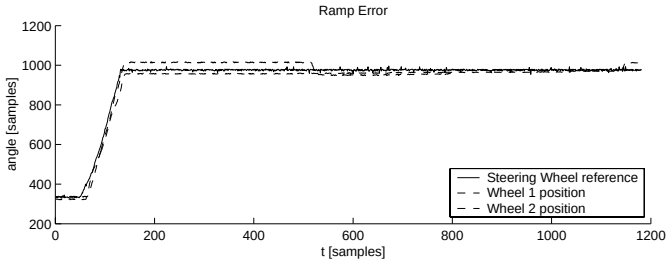

Fig. 4. Tracking error of the system. The two dashed lines show the measured actuator positions of the wheel nodes and the solid line shows the reference value sent from the steering wheel node.

has some tracking problems. This is caused by a higher bearing friction after a replacement of parts of the gear.

IV. Discussion

In general the distributed power steering system worked satisfactory, conforming to the defined functionality.

However, the speed of the system (i.e. the speed of the actuators and controllers) is rather low, limited by the actuators being geared DC motors. The problem is that it is possible to turn the steering wheel faster than the wheel actuators can turn. This can also cause the change in reference value to the wheel actuators to exceed its maximum allowed value and result in invalid data. A solution to the problem is to use more powerful DC motors with a lower gear ratio and thus being able to make a more aggressive controller.

In connection with testing the software ending in a deadlock, the placing of the simulated deadlock cannot be random. If the deadlock is placed in the master in connection with the SPI communication, the slave will end in an infinit waiting loop due to the functionallity of the SPI. This case is not considered since the SPI communication is assumed to be error free, which supports the way the test is performed.

Through the project, the following topics for further work have been discovered:

- Fault handling and signaling
- Controllers for the actuators

In the fault handling part of the constructed system most emphasis has been on the serious errors. This means that the simple errors, such as one faulty ADC in an node are not signaled under normal conditions. Furthermore, the faults are only signaled not handled.

The main area of interest in this project has been the distributed system.Therefore, only limited work has been done on motor control and motor modeling. To compensate for the tracking error, the system should be designed with the ability to follow a ramp or parabola signal.

References

[1] Microchip Technology Inc., *PIC18FXX8 Data Sheet*, U.S.A., Microchip Technology Inc., 2002.

[2] Microchip Technology Inc., *PIC18FXX2/FXX8 Programming Specifications*, U.S.A., Microchip Technology Inc., 2001.

[3] Robert Bosch GmbH, *CAN Specification Version 2.0*, Stuttgart, Germany: Bosch, 1991.

[4] *http://www.howstuffworks.com/steering.htm*, 2002.

[5] FDM, *Alt om bilen*, 2nd ed. Det Bedste fra Reader's Digest A/S, København 1974.