SpeedFace – A realtime window to a robot control

Prof. Dr.-Ing. Egon Sommer, Munich University of Applied Sciences
Dr.-Ing. Thomas Rienmüller, Reis Robotics GmbH
Dipl.-Ing. Franz Som, Reis Robotics GmbH

New applications require more complex and sophisticated sensors to be connected to modern robot controls. Some tasks even need sensor based online generation or manipulation of robot motion in realtime. Increasingly personal computers are used for processing sensor signals and calculating control algorithms. This paper shows the design and implementation of a universal CAN based connection between a personal computer and a robot control, which was given the name SpeedFace. The link is very flexible and operates in synchrony with the interpolation cycle of the robot. It was developed as a basis and toolkit for experts to program new sensor supported robot applications.

History

The roots of the project date back to the year 1998. At the beginning the prime objective was to use standards wherever possible. Some promising announcements of a software company concerning their OPC solution based on CAN led to a first development on this basis. Unfortunately, it proved at a later stage, that the necessary performance could never be reached with this approach. So the project group started to make a new concept for the communication. It should guarantee the lowest latency in operation and still be flexible and adaptable to a wide range of applications.

Requirements

The task of remote controlling movements of a robot needs at least the following amount of data to be exchanged (number of bytes in brackets):
status information (4), position information X,Y,Z and orientation A,B,C as floating point figures in double resolution (6*8 = 48), plus axis positions in increments for the main axis and for possible additional axis ((6+6)*4= 48). This adds up to about 100 Bytes to be transmitted. With additional information from sensors or drives of the robot it ends up at about 150 to 200 bytes, which means about 25 CAN telegrams of full length in both directions. All this information has to be exchanged within the interpolation cycle time of the robot, which was 12 ms. The amount of information is dependent on how many additional items of information have to be transmitted as feedback from the robot.
To address a broad range of possible users the communication partner of the robot control should be a personal computer running under a Windows™ NT/2000 operating system. The software is produced as a DLL (dynamic link library) and should have a C, C++ programming interface. (Figure 1)

General design

The SpeedFace link is intended to be used by researchers and developers who want to program their own path generator or influence the robot in a very special way that is not supported by the built-in software. In this paper this group is referred to as 'user'.
To guarantee a stable, jitterless time base it was clear that the robot control has to be the bus master. The CAN interface on the PC must have the lowest possible delay. This is the reason why an intelligent interface card with its own processing unit was not regarded as suitable. Processing all the CAN messages by the main processor leads to the lowest latency, but also consumes some computing power. With modern processors this is no big issue anymore. The main CPU carries out all the interpretation of the received CAN

messages. The reaction time then is mainly related to the interrupt response time on the personal computer.

To get a most general solution all system variables of the robot control should be accessible by means of the communication link.

The examination of the CAN Open specifications revealed that it would be possible to make a realization fully in accordance with those definitions by using PDOs, at the expense of rather limited number of variables to be transferred. On the other hand transmissions off all variables on request via SDOs would cause too much overhead. So the decision was made to only use the basic network management functions (startup, node-guarding, etc) of the CAN Open standard. The variable transfer itself is a little different from the standardized rules to achieve both performance and flexibility. The approach takes into account that in robot applications there is almost never any change of the established connection between sensor/computer and robot while operating. So there was an obvious need to design a two-phase concept.

First configure the variables in an initialization phase and then transmit the raw data with little overhead during operation.

A rough calculation shows the remaining possibilities when transmitting the maximum sized 8 byte telegrams in a 12-ms cycle and accepting 80% busload:

Speed	Max number of telegrams
500 k	36
1 M	73

The calculated number of telegrams to be transmitted will result in a busload of about 60-70% and make it clear that there cannot be other devices on the same CAN-Bus besides the robot control and the PC. Also the link has to be operated at the maximum possible speed of 1 MBd. The two devices can use all of the available PDO address range.

In order to ensure efficiency when transmitting data that may not be aligned to the 8-byte spacing of the telegrams, all the data is considered to be one large byte array.

Realtime under Windows™ NT /2000

One other main design objective besides performance was the minimization of latency times. Knowing that the chosen operating system is not ideal, a lot of effort was made to get the best out of the given situation. The cycle time is given by the robot control. The functions a user wants to be executed should be activated immediately after reception of the last CAN message in a transferred block.

A key design aspect was to get most of the user application running at the highest possible priority level. So other programs running on the personal computer cannot disturb it. The SpeedFace DLL offers two different possibilities of how user programs can be executed. One is the execution as a high priority thread under the control of the windows dispatcher. The other is to run the code in kernel mode of the operation system. The latter is much faster, but has also got several limitations and needs advanced knowledge of the user. To enable developers to compile their own application and as a proof of concept two example application were developed. These are supplied including source code.

Unfortunately, the interrupt level of a CAN interface card in a computer can only partly be adjusted and is BIOS dependent. This implies that reaction time depends on the time that other interrupts with higher priority consume. Also very important are time periods during which interrupts are disabled. So according to our experience the biggest effects arise when using improper or inefficiently programmed device drivers on the computer. Figure 2 shows the difference times in receiving subsequent packages. The results were obtained on a 1 GHz computer with running file checks and copying files over the network as background computing load. All of the measurements are within +/-100μs, but some exceptions (e.g. start of a TV application), which are not shown in the diagram, could be generated where the latency reached up to 10ms. So in general execution is fast enough, but cannot be guaranteed because other software may disturb it.

Application Examples

In order to show the versatility of this new link an example program was developed which allows the display and manipulation of all system variables. (see figure 3) The second example is a demonstration program where the robot can be moved to any given position. In this application all the realtime performance is needed, because the generation of the path is done online on the personal computer and not in the robot control. In both cases it is absolutely crucial that the user is absolutely aware of what he is doing. If an external device like the PC takes over the control of the movements then there is a total loss off the build in security functions. The same is the case when manipulating system variables like e.g. traveling limits. So this is the main trade off that has to be taken, when allowing full flexibility.

Based on the toolbox two first real applications are under development. The first is an online optimization of drive control parameters. A university institute is optimizing and testing new adaptive parameter setting algorithms. The results are now being introduced into the series versions of the robot.

A second application is the use of a robot in man-machine-cooperation task. There sensor information is combined with knowledge that can be derived from CAD models. The main benefit is, that the robot can be totally remote controlled and the gathered sensor information can be used for adapting the behavior/methods in each cycle of the task. By this means new features can be put into practice that are usually not part of a state-of-the-art robot control.

Conclusion

A powerful new interface for robots has been developed and has proved that it is meeting the needs of new industrial applications.

The steady trend to ever faster robot controls may well make it necessary to reconsider other bus systems for future high-speed applications of robots. The chosen approach will still be valid for new solutions.

Prof. Dr.-Ing. Egon Sommer
Munich University of Applied Sciences
Address: Lothstrasse 34
D-80335 Muenchen
Phone +49 89 1265 1380
Fax +49 89 1265 1299
E-mail sommer@ee.fhm.edu
Website www.ee.fhm.edu

Dr.-Ing. Thomas Rienmüller
Company: Reis Robotics
Address: Industriegebiet an der B426
D-63785 Obernburg
Phone +49 6022 503 576
Fax +49 6022 503 110
E-mail t.rienmueller@reisrobotics.de
Website www.reisrobotics.de

Dipl.-Ing.Franz Som
Company: Reis Robotics
Address: Industriegebiet an der B426
D-63785 Obernburg
Phone +49 6022 503 560
Fax +49 6022 503 110
E-mail f.som@reisrobotics.de
Website www.reisrobotics.de

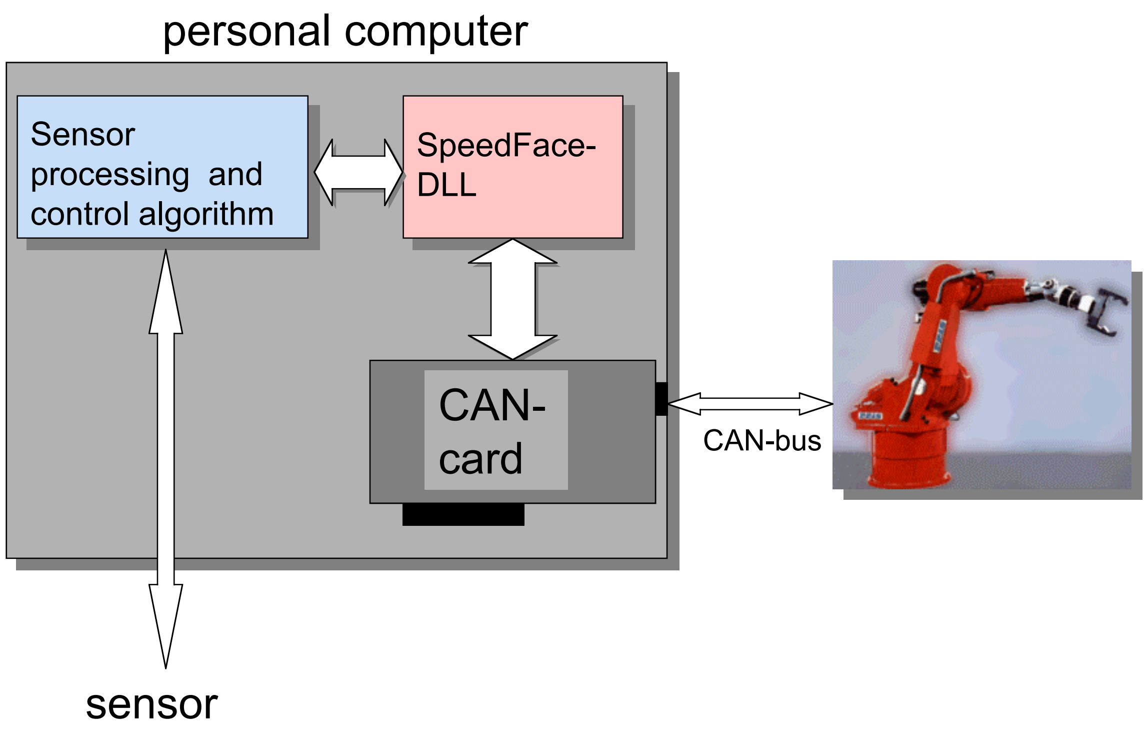


Figure 1: configuration of the link

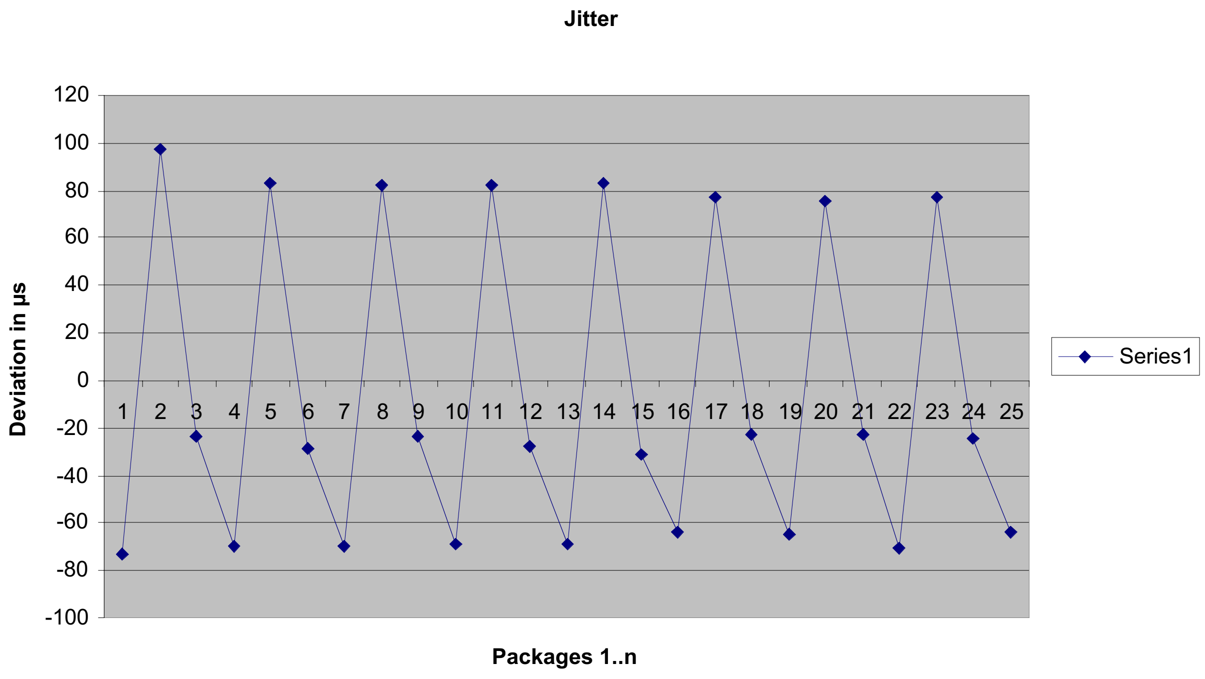
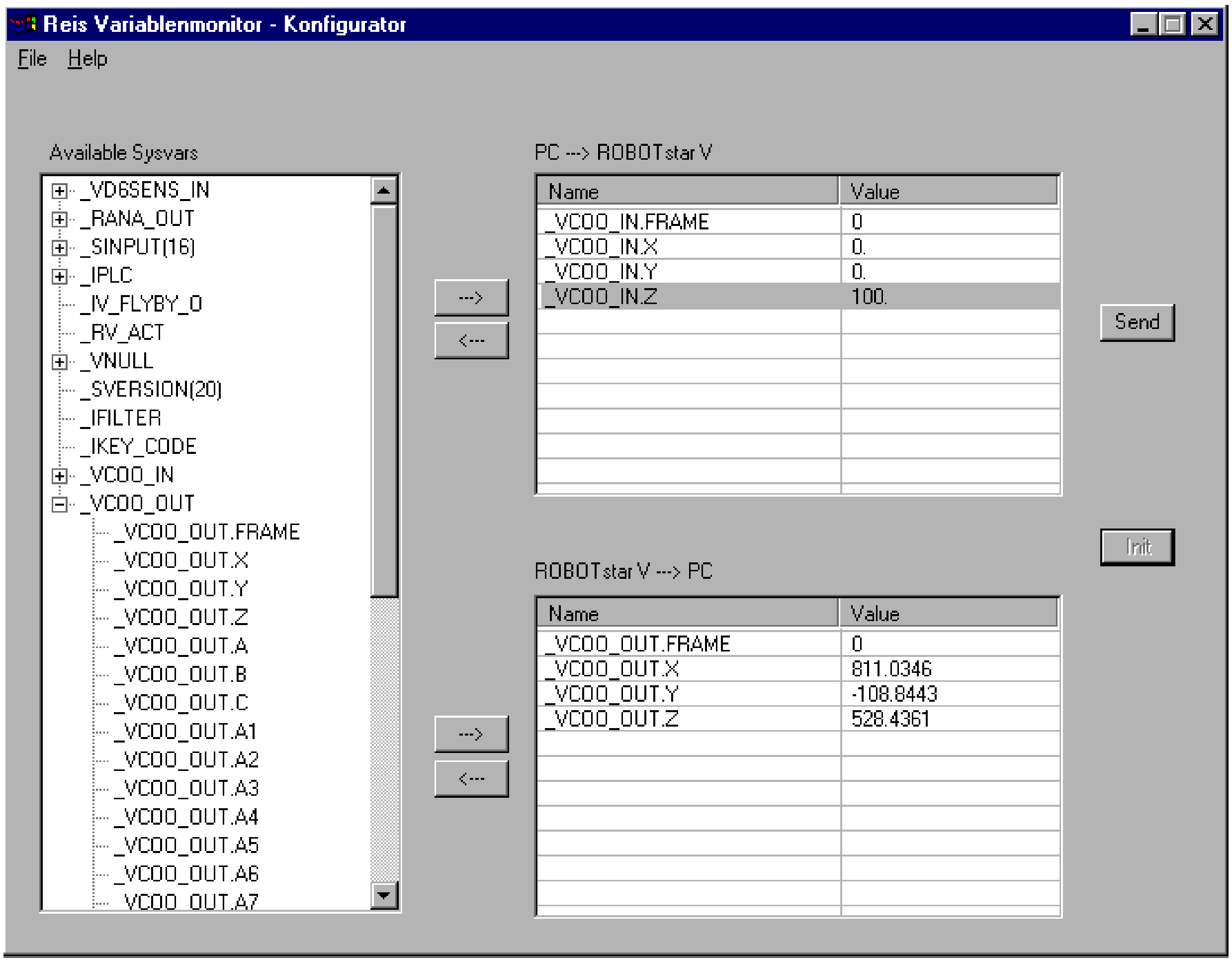


Figure 2: time measures receiving packages



_VD6SENS_IN
_RANA_OUT
_SINPUT(16)
_IPLC
_IV_FLYBY_0
_RV_ACT
_VNULL
_SVERSION(20)
_IFILTER
_IKEY_CODE
_VC00_IN
_VC00_OUT
_VC00_OUT.FRAME
_VC00_OUT.X
_VC00_OUT.Y
_VC00_OUT.Z
_VC00_OUT.A
_VC00_OUT.B
_VC00_OUT.C
_VC00_OUT.A1
_VC00_OUT.A2
_VC00_OUT.A3
_VC00_OUT.A4
_VC00_OUT.A5
_VC00_OUT.A6
_VC00_OUT.A7

Name	Value
_VC00_IN.FRAME	0
_VC00_IN.X	0.
_VC00_IN.Y	0.
_VC00_IN.Z	100.

Name	Value
_VC00_OUT.FRAME	0
_VC00_OUT.X	811.0346
_VC00_OUT.Y	-108.8443
_VC00_OUT.Z	528.4361

Figure 3: variable monitor