

Middleware-based Kalman filter design for a driver's aid system

Wenwei Hou¹, Cecil Bruce-Boye², Dmitry A. Kazakov³ and Youling Zhou⁴

Abstract— In this paper we present an implementation of the Kalman filter based on a middleware. This middleware LabMap is applied as a data acquisition and distribution layer in industrial automation area, serving as a software bus. We use its virtual handles to implement the algorithm of the Kalman filter directly at the level of the software bus. The advantage of this approach is that neither of the system components needs to be changes as the implementation is enclosed in the middleware. We show that the Kalman filter is useful to remove the speed noise in a driver's aid system.

Note to Practitioners— This paper was motivated by the problem of the speed noise in the driver's aid system ErgoDrive Professional for the visual instruction of test drivers on chassis dynamometers. The speed data from the hardware has noise. This leads to trembling of the speed cursor in the driver's aid, which makes it is difficult for the driver to drive the test profile. In this work we applied the Kalman filter to remove the speed noise. Our experiments showed that we always were able to find suitable parameters for the Kalman filter for sampling rates up to 100ms. For larger sampling time intervals performance degrades. In future research we will try to solve this problem with a predictive filter.

Index Terms— Middleware, Software bus, LabMap, Kalman filter

I. INTRODUCTION

In a distributed computing system, middleware is defined as a software layer between software components used to reduce coupling.

With the growth of network-based applications, middleware technologies become more and more important. They cover a wide range of software systems, including distributed objects and components, message-oriented communication, and mobile application support.

The role of middleware is to make application development easier, by providing common programming framework, by abstracting heterogeneous and distributed nature of the underlying hardware and software components, and by hiding lower-level programming details.

LabMap [1, 2, 3, 4] represents an example of such middleware used in the process automation and data acquisition area. It provides an abstraction of the application level from the hardware specifics by decoupling the hardware interface modules from the application level. It also allows a smooth integration of numerous components with their variety of software and hardware protocol, supporting a component

based software design. Another important advantage of this particular middleware is that it has computable data channels, so-called virtual handle, so that a definite data processing can implement at the level of middleware without modification of other software components.

ErgoDrive Professional [5, 6] is a driver's aid system used to in chassis dynamometer automation solutions and test environments. It is being deployed in many application areas such as exhaust emission certification, brake force measurements, vehicle cooling testing, fuel consumption testing, gear box optimizations.

The Kalman filter [7] is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects.

In this paper we will use a new tool of the middleware LabMap virtual handle, to implement the Kalman filter and show it is useful to remove the noise of the speed in the driver's aid system.

II. THE MIDDLEWARE ARCHITECTURE

As a distributed middleware LabMap is operating as a software bus. It provides a high level interface for industrial communication, control, measurement, and logging. It allows asynchronous viewing and modification of the process variables from a potentially unlimited number of applications running on the same or different LAN/WAN network hosts. The middleware has an open architecture supporting smooth integration of new hardware and software components running on different bus architectures. It has integrated support of measurement units and time stamping of the system variables.

This software bus offers an abstraction of the application level from the hardware specific and decoupling the hardware interface modules from the application level. In other words there are two levels of abstraction supported by the bus system:

A. *The application interface*

B. *The hardware driver interface*

The application abstraction interface exposes LabMap as a set of variables (registers). Each register has a type, value, timestamp and I/O direction. The four basic I/O requests are supported for each register:

1) Get

The value of a register can be read in a safe way that warrants consistency of the read value bits and the timestamp. The application is relieved from the burden of locking the value in presence of other tasks accessing it concurrently.

2) Set

The value of a register can be set in a safe way. No I/O initiated by this request.

3) Request (for output registers only)

A new value of a register can be requested from the underlying hardware. The application needs not to know which actions are necessary to request the new value. It is the responsibility of the driver. The request is asynchronous and the application is not blocked until I/O completion. However it may enter a non-busy waiting for the I/O completion.

4) Send (for input registers only)

The actual value of a register can be sent to the underlying hardware. Like in the case of request the application is unaware of the actions the hardware undertakes upon send. The application may enter a non-busy waiting for the I/O completion.

The hardware abstraction interface of the software bus allows an easy integration of new hardware devices and communication protocols into the system. Each class of hardware devices or communication protocol supported by the software bus is represented by a hardware interface. The hardware interface is responsible for implementation of the basic I/O operations upon the registers assigned to it. The software bus uses a messages mechanism for interacting with the hardware interfaces. A hardware interface is developed as dynamic-link library which code can be maintained independently from the middleware core.

C. Computable registers

Computable registers are represented in the middleware LabMap as a virtual hardware. The corresponding interface is named LabVirt. In this paper we use this functionality to implement a Kalman algorithm solely in the middleware without modification of the driver's aid component. The computable registers interface gives an ability to calculate registers out of other register values. The way a register is evaluated is determined by its configuration, which basically a program specified in the language of LabVirt. It is a compiled language. The code is generated for the virtual machine integrated into the computable registers interface implementation.

For example if we wanted to add values of two registers and store the result into a third register, we could define the virtual register like:

$$(\$100 + \$200) \rightarrow 300$$

III. THE ALGORITHM OF THE DISCRETE KALMAN FILTER

The Kalman filter is a set of mathematical equations that provides efficient computational means to estimate the state of a process, in a way that minimizes the mean of the squared

error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown [8, 9, 10].

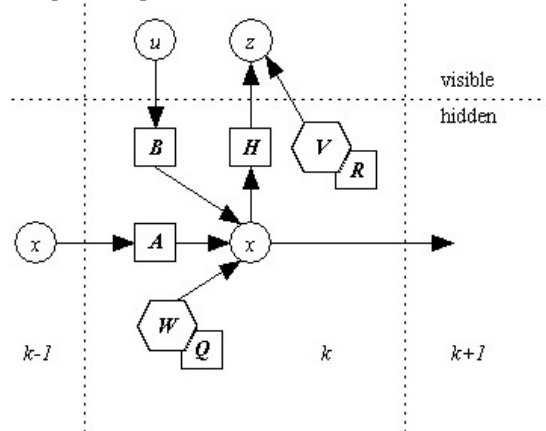


FIG.1 The model of the Kalman filter

A. The Process To Be Estimated

The Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \quad (1)$$

with a measurement that is

$$z_k = Hx_k + v_k \quad (2)$$

The random variables w_k and v_k represent the process and measurement noise (respectively). They are assumed to be independent (of each other), white, and with normal probability distributions

$$p(w) \sim N(0, Q), \quad (3)$$

$$p(v) \sim N(0, R) \quad (4)$$

In practice, the process noise covariance Q and measurement noise covariance R matrices might change with each time step or measurement, however here we assume they are constant.

The $n \times n$ matrix A in the difference equation (1) relates the state at the previous time step $k-1$ to the state at the current step k , in the absence of either a driving function or process noise. The matrix B relates the optional control input $u \in R^l$ to the state x . The $m \times n$ matrix H in the measurement equation (2) relates the state to the measurement z_k . In practice A , B and H might change with each time step or measurement, but here we assume it is constant.

B. The Discrete Kalman Filter Algorithm

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter

fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. Indeed the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems as shown below in Fig. 2.

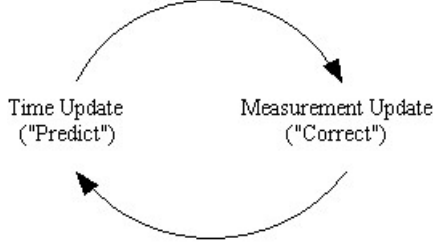


FIG.2 The ongoing discrete Kalman filter cycle

The specific equations for the time and measurement updates are presented below.

Discrete Kalman filter time update equations:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \quad (5)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (6)$$

Discrete Kalman filter measurement update equations:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (7)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (8)$$

$$P_k = (I - K_k H)P_k^- \quad (9)$$

IV. APPLICATION OF THE KALMAN FILTER FOR DRIVER'S AID SYSTEM

A. System Structure

The system structure is shown on Fig.3. On the computer LabMap runs with registers configured for a DAC and ADC modules used to set a defined voltage over the amplifier to the motor and get the voltage signal generated by the tachomachine. The modules are connected to the middleware using the ModBus protocol. The hardware interface responsible for its implementation in the middleware is LabModBus. The corresponding variables are the registers served in the middleware by this interface.

There are acceleration and brake pedals used for the user input. The values generated by the pedals according to the force applied are acquired by an application and stored into the software bus using the hardware interface called LabUser. This interface allows a direct control over the value of the corresponding variables without having any hardware behind them.

The variables controlling the driver's aid are served by the LabFLG hardware interface. The Ergo Drive driver's aid software is treated as a hardware in this setup.

The variables implementing the Kalman filter are served by the LabVirt interface.

The variables used for setting the filter parameters are LabUser registers.

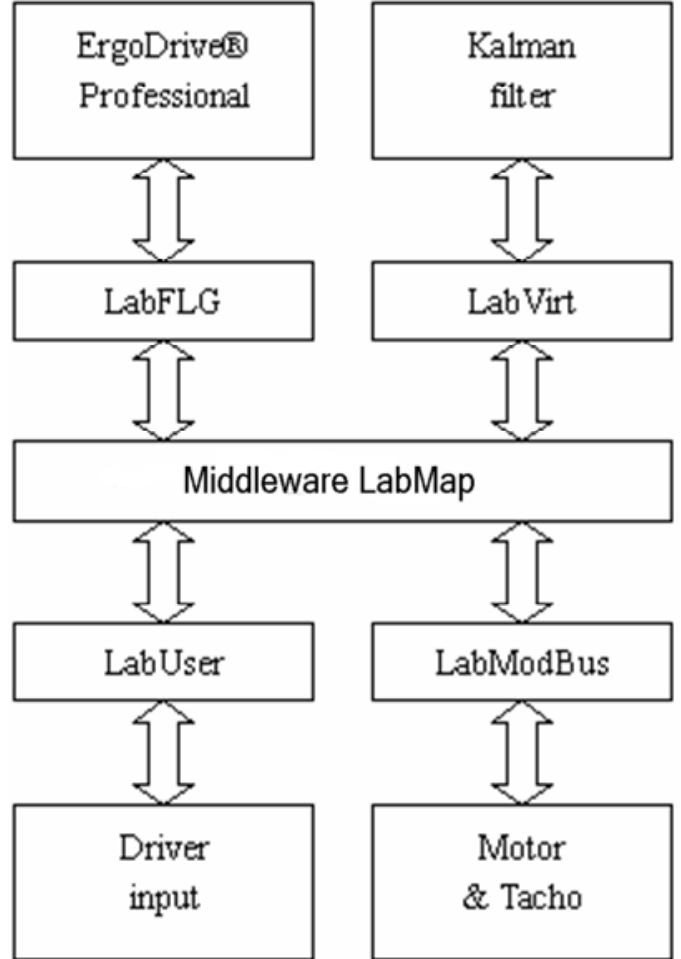


FIG.3 System structure

Due to the distributed nature of LabMap we were able to test and use different setups of our system in which some of the system components run on physically different computers.

B. The Process Model

We try to remove the noise from the speed of the motor that we get from indirectly the tachomachine. Let us assume that we give a constant voltage to the motor, so the speed of the motor is also a constant value, so $A=1$. There is no control input so $u=0$. So

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} = x_{k-1} + w_{k-1} \quad (10)$$

The tachomachine can generate voltage according to the speed of the motor. The relationship between voltage and speed is 0.52V per 1000rpm that is 0.0312V*s. So we can calculate the frequency from the voltage. On the other hand we can get the speed of motor by multiplying frequency with the girth of wheel. The input of the filter is that speed so $H=1$. So

$$z_k = Hx_k + v_k = x_k + v_k \quad (11)$$

Our time update equations are

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} = \hat{x}_{k-1} \quad (12)$$

$$P_k^- = AP_{k-1}A^T + Q = P_{k-1} + Q \quad (13)$$

And our measurement update equations are

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} = P_k^- (P_k^- + R)^{-1} \quad (14)$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-) = \hat{x}_k^- + K_k (z_k - \hat{x}_k^-) \quad (15)$$

$$P_k = (I - K_k H)P_k^- = P_k^- - K_k P_k^- \quad (16)$$

Let us substitute (13) into (14) so we have

$$K_k = (P_{k-1} + Q)(P_{k-1} + Q + R)^{-1} \quad (17)$$

Let us substitute (12) and (17) into (15) so we have

$$\hat{x}_k = \hat{x}_{k-1} + (P_{k-1} + Q)(P_{k-1} + Q + R)^{-1} (z_k - \hat{x}_{k-1}) \quad (18)$$

Let us substitute (13) and (17) into (16) so we have

$$P_k = P_{k-1} + Q - (P_{k-1} + Q)^2 (P_{k-1} + Q + R)^{-1} \quad (19)$$

So the key to implement this Kalman filter by the middleware is to define two virtual-Registers with the filter algorithm (18) and (19). The software bus LabMap configuration is presented in the Table 1.

TABLE I
LABMAP REGISTER CONFIGURATION

Register No.	Configuration	Description
10000	>User< [::]!O:R:0:0:-1:-1:: Y_Koordinate	The value got from the joy stick.
10002	>Virt< [V:V:V]!O:R:0:0:-1:-1:: joy value to motor voltage 10000- >[(value*0.00030517578 125-9.99969482421875 max 0)*1[V]->710]	Calculate the motor voltage from the value of joy stick ²
710	>ModBus< [V:V*0.000305185:V]!I: R:0:0:-1:-1:: Motor voltage [Coupler1 type=i addr=0 Len=16]	ModBus input register to set a defined voltage (+/- 10V) to the Wago analogue output module on output channel 0 -> motor voltage.
720	ModBus< [V:0.000305185*V+0.043 3363:V] !O:R:0:10:-1:-1:: tacho voltage [Coupler1 type=i addr=0 Len=16]	ModBus output register to get the noised voltage from the Wago analogue input module on input channel 0 <- tacho voltage.
721	>Virt< [km/h:km/h:km/h]!O:R:0: 0:-1:-1:: convert filter 720- >[\$720/0.0312[V*s]*0.29 9[m]*3600[s/h]/1000[m/k m]=>722]	Convert the voltage to the speed
722	>User< [km/h:km/h:km/h]!O:R:0: 0:-1:-1:: original speed	z_k The input of the filter
730	>User< [km/h:km/h:km/h]!I:R:0:0 :-1:-1:: speed_kalman	\hat{x}_k User register for the posteriori estimate speed of the motor. Before testing the filter the register must be set with a defined value.g
740	>User<	P_k User register for the

	[km ² /h ² : km ² /h ² : km ² /h ²] !I:R:0:0:-1:-1:: Covariance	posteriori estimate error covariance. Before testing the filter the register must be set with a defined value.
750	>User< [km ² /h ² : km ² /h ² : km ² /h ²] !I:R:0:0:-1:-1:: Q	Q User register for process noise covariance. It can be modified at runtime.
760	>User< [km ² /h ² : km ² /h ² : km ² /h ²] !I:R:0:0:-1:-1:: R	R User register for measurement noise covariance. It can be modified at runtime.
770	>Virt< [km ² /h ² : km ² /h ² : km ² /h ²] !O:R:0:0:-1:-1:: covariance_filter 722->[((\$740+\$750)*(1- 1/(1+\$760/(\$740+\$750))) >740]	Virtual output register with filter algorithmic of the posteriori estimate error covariance.
780	>Virt< [km/h:km/h:km/h] !O:R:0:0:-1:-1:: speed_filter 740->[\$730+(\$722- \$730)/(1+\$760/(delay(val ue)+\$750))]->730]	Virtual output register with filter algorithmic of the posteriori estimate speed of the motor

V. RESULTS

To use the filter firstly we must set suitable value to Q and R. Their values have great influence on the performance of the filter. So we always have to find a relatively better parameter, which can keep the output smooth and do not have too much time delay.

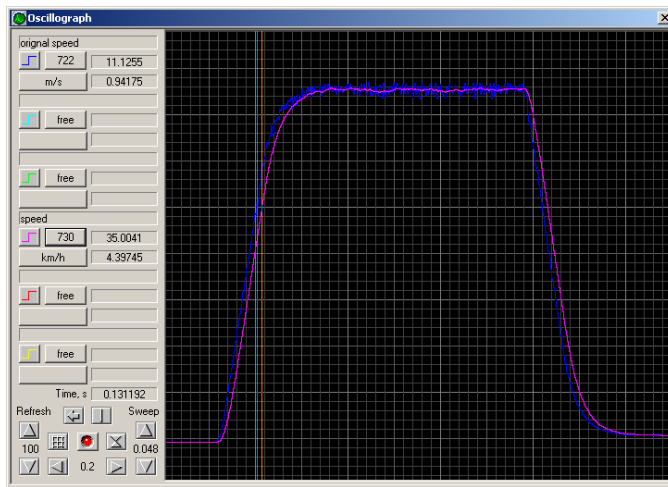
To use the filter we also must set initial value to the speed register and the covariance register. From the Kalman filter theory, we know that the alternative choice is not critical. We could choose almost any $P_0 \neq 0$ and \hat{x}_0 , the filter would eventually converge.

We try different values of Q and R to test the influence of them on the output signal. We can conclude that in the case of keeping R as a constant when Q becomes smaller, the output of the filter will become smoother but at the same time the time delay will become bigger.

On the other hand, in the case of keeping Q as a constant when R becomes bigger, the output of the filter will become smoother but at the same time the time delay will become bigger.

Finally we find that if sampling interval is 10ms, then $Q=0.000001 \text{ km}^2/\text{h}^2$, $R=0.001 \text{ km}^2/\text{h}^2$ would be a relatively better parameter.

Fig. 4 represents the input of the filter (blue line) and the output of filter (red line). From the figure we can see that the output signal has a delay around 150ms.

FIG.4 $Q=0.000001 \text{ km}^2/\text{h}^2$, $R=0.001 \text{ km}^2/\text{h}^2$

The figure is obtained from the oscilloscope tool integrated in the LabMap.

After some more experiment, we find when we change the sample time of the input signal we always can find a relatively better parameter of the filter.

TABLE II
DIFFERENT PARAMETER FOR DIFFERENT SAMPLING INTERVAL

Sample time (ms)	Q (km ² /h ²)	R (km ² /h ²)	Time delay (ms)
10	0.000001	0.001	150
20	0.00001	0.001	100
30	0.00001	0.001	100
50	0.00001	0.0001	50
70	0.00001	0.0001	70
100	0.00001	0.0001	100

This kind of Kalman filter has been used in a driver's aid system. It is useful to remove the noise of the speed signal from the chassis dynamometer so that the driver can see a smooth time-speed line on the screen. Fig. 5 represents the driver's aid display when driven without the filter.

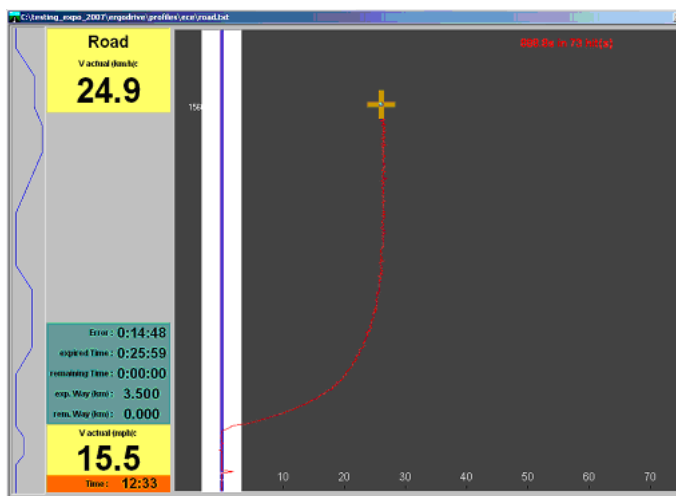


FIG.5 Driver's aid behavior without filtering

Fig. 6. represents the driver's aid performance with the Kalman filter we have designed.

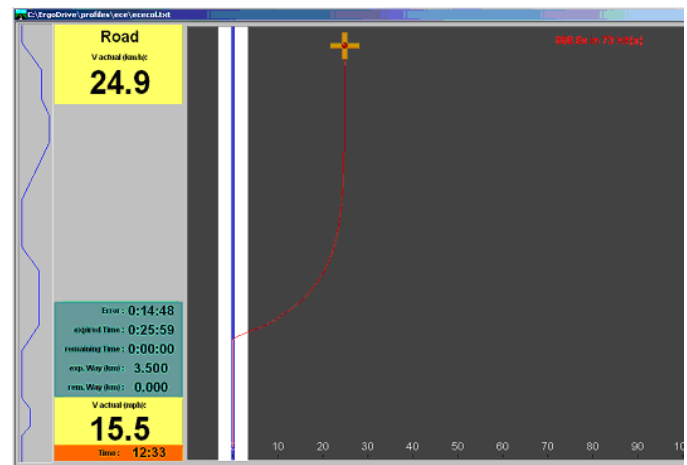


FIG.6 Driver's aid behavior with the Kalman filter

VI. CONCLUSION

We have observed that this software bus with its computable data channels is suitable for simple digital signal processing as required in engineering tasks. Some problems like on-line signal filtering can be designed, tested and deployed without modification of the legacy components. For an implementation of the algorithms such as the Kalman filter in the software bus, we need to define proper input and output registers and describe the algorithm in the language of the computable registers.

REFERENCES

- [1] C. Bruce-Boye and D. Kazakov, "Distributed data acquisition and control via software bus," Proceedings CSMITA'04, pp. 153–156, Sep 2004.
- [2] C. Bruce-Boye, D. Kazakov, Quality of Uni- and Multicast Services in a Middleware. LabMap Study Case," Conference CIS²E 06 (International Joint Conference on Computer, Information and System, Science and Engineering") 2006, IEEE, 4-14 December 2006
- [3] C. Bruce-Boye, D. Kazakov; Rüdiger zum Beck, "An approach to distributed remote control based on middleware technology, MATLAB/Simulink-LabMap/LabNet framework", Conference CIS²E (International Joint Conference on Computer, Information and System, Science and Engineering") 2005, IEEE, 10-20 December 2005.
- [4] C. Bruce-Boye, D. Kazakov, "Distributed data acquisition and control via software bus", International Industrial Ethernet Development High Level Forum 2004 (IEHF 2004) in Peking, Automation Panorama No. 5
- [5] R. Müller, D. Kazakov, A. Fechner, A. Wilde, „A smooth ride“, Testing technology International“, pp.82-84, Nov. 2000
- [6] "GPS Recorder", Testing Technology International, p.95, November 2004
- [7] Kalman, R. E., "A new approach to linear filtering and prediction problems," Transaction of the ASME—Journal of Basic Engineering, March 1960, pp. 35-45
- [8] Brown, R. G. and P. Y. C. Hwang, "Introduction to Random Signals and Applied Kalman Filtering", Second Edition, John Wiley & Sons, Inc, 1992
- [9] Grewal, Mohinder S., and Angus P. Andrews. "Kalman Filtering Theory and Practice". Upper Saddle River, NJ USA, Prentice Hall, 1993
- [10] Sorenson, H. W. "Least-Squares estimation: from Gauss to Kalman," IEEE Spectrum, vol. 7, pp. 63-68, July 1970.

²**Cecil Bruce-Boye**, University of applied science, 3 Stephensonstrasse
Luebeck, 23562 Germany

³**Dmitry A. Kazakov**, cbb software GmbH, 1 Charlottenstrasse Luebeck,
23560 Germany

⁴**Youling Zhou**, East China University of Science and Technology, 130
Meilong Road, Shanghai, 200237, P.R.China

ACKNOWLEDGES

We should give special thanks to Luebeck University of Applied Sciences and East China University of Science and Technology. They launch the exchange student program supported by DAAD, so that we can work together and write this paper.