Contents lists available at SciVerse ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

A new error handling algorithm for controller area network in networked control system

M.B. Nor Shah [a,b,*], A.R. Husain [b], S. Punekkat [c], R.S. Dobrin [c]

[a] Faculty of Engineering Technology, Universiti Teknikal Malaysia, Melaka, Malaysia
[b] Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Johor, Malaysia
[c] Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden

ARTICLE INFO

Article history:
Received 10 August 2012
Received in revised form 22 April 2013
Accepted 31 May 2013
Available online 19 July 2013

Keywords:
Controller area network (CAN)
Error handling
Network control system

ABSTRACT

An effective error handling mechanism plays an important role to ensure the reliability and robustness of the application of controller area network (CAN) in controlling dynamic systems. This paper addresses a new online error handling approach or named per-sample-error-counting (PSeC) technique that tends to replace native error handling protocol in controller area network (CAN). The mechanism is designed to manage transmission errors of both sensor and control data in networked control system (NCS) used in controlling dynamic system such that the stability of the feedback system is preserved. A new parameter denoted as maximum allowable number of error burst (MAEB) is introduced in which MAEB is selected based on available bandwidth of the CAN network. MAEB serves as the maximum number of attempt of re-transmission of erroneous data per sample which allows the maximum transmission period to be known and guaranteed for time-critical control system. The efficacy of the proposed method is verified by applying the algorithm on the fourth order inverted pendulum system simulated on Matlab/Truetime simulator and the performance is benchmarked with the existing CAN error management protocol. The simulation run under various systems conditions demonstrate that the proposed method results in superior system performance in handling data transmission error as well as meeting control system requirement.

1. Introduction

Networked control systems (NCSs) are real-time systems where sensor, controller and actuator data packets are transmitted through a shared communication network forming a closed loop system. As depicted in Fig. 1, each loop of NCS consists of sensor, controller and actuator nodes that are interconnected in a network, where executing multiple control loops for spatially distributed plants is viable. This flexible architecture of NCS is an obvious alternative to the point-to-point communication system where it has gained high popularity due to many advantages such as low installation cost, easy maintenance, re-configurability and more structured for fault diagnosis purposes, to list a few [1,2]. In addition, with the advancement of high-speed low-cost micro-computing technology, the possibility of operating at very high frequency which implies higher bandwidth availability has further attracted the application of this network communication in control system development and becoming more prevalent in many high-end applications such as spacecraft [3], unmanned aircraft [4], automotive [5] and factory automation [6]. Many excellent literatures reporting the trend, design and result of NCS can be found in [1,7–10].

In term of controlling dynamic systems that have strict temporal requirement, high-speed serial bus communication has been used as the 'backbone' or the enabler of NCS in the application. Fieldbus technology such as PROFIBUS [11], WorldFIP [12], ControlNet [13], DeviceNet [14], switched Ethernet [15,16] and CAN are among the most popular fieldbuses that are being adapted in application where each of this fieldbus has their own specific protocol to handle data management that includes arbitration process, data encapsulation and handling as well as error management and confinement. The mechanisms of each protocol that handle the bit and frames transmitted over the network are being monitored ensuring the data is correctly transmitted and received and the controlled system to perform tasks assigned. The implementation of each of the mechanisms does consume the bandwidth allocated in the network. Specifically

* Corresponding author at: Universiti Teknologi Malaysia, Control and Mechatronics Engineering Department, Faculty of Electrical Engineering, 81310 Skudai, Johor, Malaysia. Tel.: +60 196848520.
E-mail addresses: bad_z81@yahoo.com (M.B. Nor Shah), rashid@fke.utm.my (A.R. Husain), sasikumar.punnekkat@mdh.se (S. Punekkat), radu.dobrin@mdh.se (R.S. Dobrin).

0166-3615/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.compind.2013.05.008

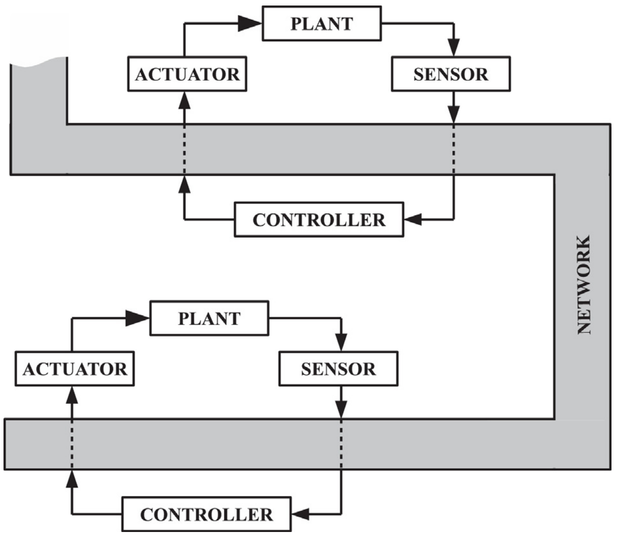

Fig. 1. Configuration of NCS.

to error management in CAN, erroneous data can be rooted from many factors such as Electromagnetic Interference (EMI) [17–19], dry solder on PCB, unsynchronized clock and hesienbug [18,20].

In many application, erroneous data can lead to undesirable result to the system being controlled and thus the error handling feature is designed to provide error checking mechanism in CAN protocol to perform not only overcoming the error, but also in a timely manner. The basic idea of this feature is to detect errors and retransmit automatically the affected messages. However, this error handling feature may not be suitable to applications with critical timing requirement, since repetitive data transmission can increase data transmission delay that degrade the performance of control system in the loop.

There are several works that has been done regarding data error handling and faulty node confinement of CAN to enhance dependability of network communication. In [21], a program called 'Monitor' has been design to diagnose faults in CAN nodes. The program has capability to check the memories including random access memory (RAM) and read only memory (ROM) of each CAN node, if the transmitted data is not identical to the data in node memories, the program then discard the message and re-transmit the correct data to the network. The program also has capabilities to reset the bus-off nodes. In [22], a simple bus guardian solution for the FlexCAN architecture is introduced to manage data error and faulty nodes. This solution has demonstrated it is more effective compare to native CAN protocol in handling babbling idiot faulty nodes. In [23], a scheduling technique has been designed to deal with data error in CAN where this scheduling technique override the native error handling of CAN. Gaujal and Navet (2005) in [20] performed a Markovian analysis of the CAN network under the EMI burst and permanent hardware failure. Based on the analysis, it is found that the system reaches the bus-off mode rather too quickly when it is under EMI-burst condition and two error confinement methods by quantifying the progression of nodes toward the bus-off and error passive modes are proposed. The experimental results validate the proposed approach, however, the total execution time of the algorithm is not known.

The operation of native error management of CAN protocol is to retransmit erroneous data until it is successfully transmitted. However, uncontrolled number of retransmitted data could cause bandwidth overload and thus lead the performance deterioration and system instability of NCS. Hence in this article, we propose a new error handling technique in CAN where the closed loop control system resides in the network. The technique named per-sample-error-counting (PSeC) is designed based on online monitoring and counting of erroneous sensor and control data at every sampling instance. A parameter, denoted as maximum allowable number of error burst (MAEB) is introduced to indicate the maximum number of attempt of re-transmission of erroneous data per sample which allows the maximum transmission period to be known and guaranteed for time-critical control system. The newly proposed PSeC method is shown to be effective to meet the time requirement of linear closed loop control system and tends to replace native error handling feature in CAN.

In the attempt to recover erroneous data in network, there exist several techniques such as forward error correction (FEC), partial order connection (POC) and automatic retransmission request (ARQ) that would be suitable for certain class of applications. For error handling protocol in CAN, ARQ technique has been adapted to recover erroneous data [24]. In the process of correcting faulty data in network, it involves two general steps which are: (1) error detection and (2) recovery mechanism. For FEC and POC, it covers both steps as reported in [25,26]. However for PSeC strategy proposed in this work, it is focused on error detection mechanism alone such that the decision resulted from this algorithm will be used in the native CAN recovery mechanism technique based on available bandwidth.

In real time system (m,k)-firm model is usually adapted to evaluate the system performance as in the embedded application shown in [27,28]. However from control point of view, this model is less favorable since in the application of NCS in control system that involve the control of fast dynamical system based on CAN network, i.e. robotic arm, engine control, the performance and stability of NCS is not only influenced by data transmission rate, but also the data transmission delay related to system dynamic stability that contribute a significant performance degradation in NCS. PSeC strategy is formulated from the nature of data transmission delay in CAN, subsequently creating the property to control the deterministic and boundedness of NCS by limiting the time bound of messages retransmission when the number of error bursts exceed MAED, which is recommended from aspect of system stability.

The rest of this paper is organized as follows. Section 2 covers the fundamental of CAN protocol on data transmission and error management within the scope that is sufficient for the development of the PSeD method. Section 3 discusses the development of task model and error model that encapsulates the time division of message in CAN frame. Section 4 gives a brief on NCS model with time delay. Section 5 covers the details on this newly proposed new error handling technique and bandwidth allocation for control data and non-control data. Section 6 shows the derivation of delay analysis in CAN and also stability condition of NCS under error burst. Simulation and analysis of the PSeD on inverted pendulum system is shown in Section 7 with some discussions and the conclusion is drawn in Section 8.

2. Overview of CAN protocol

CAN is an advanced serial bus system with high speed, high reliability and low cost which make it suitable for many distributed real time control applications. It was initially developed for automotive use in late 1980s by Robert Bosch, but now CAN is widely utilized in most real time automation system due to robustness to electrical interferences, ability to self diagnose and data errors repair, high performances and suitable for harsh environment. CAN uses carrier sense multiple access protocol with collision detection (CSMA/CD) and arbitration on message priority as its communication protocol to ensures that a message is successfully transmitted to particular node. The most important feature of CAN from the real-time perspective is its predictable behavior by providing means for prioritized control of the transmission medium by using an arbitration mechanism which guarantees that the highest priority message that enters arbitration will be transmitted.

CAN is an advanced serial communication bus designed for short messages transmission and currently it can operate at the speed up to 1 Mbps. Each data transmission frames carry 0–8 bytes of data, encapsulated with message identifier and other control protocol bits. There are two versions of protocol that are widely used: 2.0A (standard version) which supports 11-bit message identifier and 2.0B (extended version) which supports both 11-bit and 29-bit identifier. In this paper, we only consider standard version 11-bit identifier of CAN data (the generalization of this proposed method to CAN 2.0B is straight forward and purposely omitted). The identifier is important in CAN data transmission as to determine the priority of the messages. The lower the value of message identifier means the higher priority of the message since in CAN network, logic bit 0 is set as dominant bit and a logic bit 1 as a recessive bit. A dominant bit state will always win arbitration over a recessive bit state due to wired-AND configuration and this serves as the mechanism to allow lower identifier values to have higher priority message. As an illustration of this arbitration sequence, Fig. 2 shows two nodes trying to send messages where the value of message identifier of node A is lower than message identifier of node B. At 4th bit, dominant bit of node A collides with recessive bit of node B, where node A wins the arbitration of the bus. Node A continues transmitting the message and node B has to wait for the next idle period of network try to re-transmit the message.

Besides the data and arbitration field, CAN frame also comprises of Cyclic Redundancy Check (CRC) and the acknowledgment fields which constitute 47 bits of normal CAN frame. However, during the incidents of instantaneous six similar bits, i.e. '111111' or '000000', CAN system will introduce a stuff bit in order to maintain the network synchronization. The frame format is specified such that only 34 of the 47 control bits are subjected to bit stuffing. Thus, the maximum number of stuff bits in a message frame with n bytes of data is $\lfloor (8n+34-1)/4 \rfloor$. Hence, by considering size of data, protocol control information and stuffing bits, the size of a transmitted CAN message frame, denoted as f can be calculated to become

$$f = 8n + 47 + \lfloor \frac{8n+34-1}{4} \rfloor \tag{1}$$

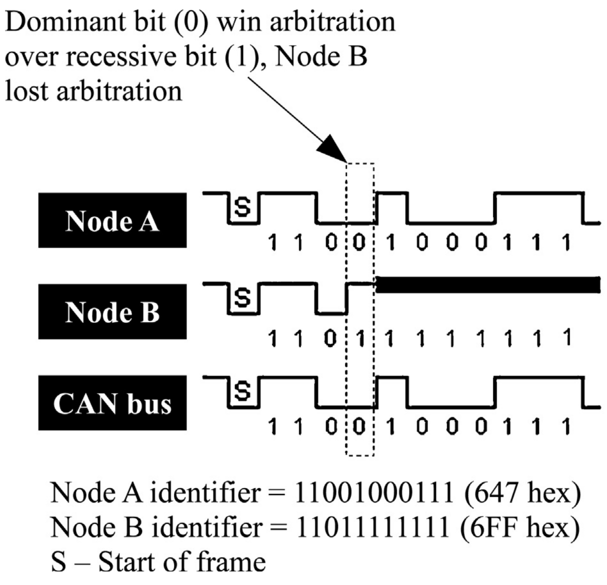
Dominant bit (0) win arbitration over recessive bit (1), Node B lost arbitration
Node A
S
1 1 0 0 1 0 0 0 1 1 1
Node B
S
1 1 0 1 1 1 1 1 1 1 1
CAN bus
S
1 1 0 0 1 0 0 0 1 1 1
Node A identifier = 11001000111 (647 hex)
Node B identifier = 11011111111 (6FF hex)
S – Start of frame

Fig. 2. Node A wins arbitration over Node B.

As explained in previous section, high-speed CAN message is prone to errors during transmission due to EMI and possible hardware or software faults. In order to overcome the situation, CAN protocol also provide error handling mechanism to retain the transmission of the erroneous messages. This mechanism is able to handle all the five types of errors that are stuffing error, bit error, checksum error, frame error and acknowledgment error. This built-in CAN error detection is proven to be very efficient since the probability of undetected transmission error is extremely small and thus it can be assumed that all errors can be detected in our analysis [29]. Once an error is detected, detecting node will transmit an error flag containing six bits of the same polarity. This is purposely to make the error globalized to all nodes. Each node then discards their messages in order to give access for the sender node to retransmit the erroneous message. However, the retransmission of the message could be subjected to arbitration with other messages during retransmission process. If any higher priority messages get queued during the transmission and error is signaled for the current message, then those messages with higher priority will be transmitted before the erroneous message is re-transmitted. This native error mechanism handling with retransmission feature implies additional undesirable transmission delay in the NCS that would possibly lead to degradation of system performance or, in worst case, system instability.

To further compartmentalize the handling of CAN error management, two types of error frame which are active frame and passive frame are defined where the active error frame is composed of six consecutive dominant bits while passive error frame composed of six consecutive recessive bits. This bit sequence actively violates the bit-stuffing rule. All other stations recognize the resulting bit-stuffing error and in turn generate error frame themselves, called superposed error flags. The error delimited field (eight recessive bits) completes the error frame. Upon completion of the error frame, bus activity return to normal and the interrupted node attempts to resend the aborted message. Type of transmitted error frame is specified by fault confinement protocol based on receive error counter (REC) and transmit error counter (TEC) [29]. Error signaling and recovery time is typically between 17 and 31 bit times [30].

3. Task model and error model

As shown in Fig. 1, NCS configuration consists of sensor node, controller node and actuator node with their dedicated pre-assigned tasks which are sensor task, controller task and actuator task. Sensor task is performed at sensor node and responsible to read sensor value from system and send it to controller node via network. Sensor task is clock driven and can be defined as $TS_s = (T_s, C_s, D_s, P_s, F_s, f_s)$ where T_s is the period, C_s is the worst case execution time, D_s is the relative deadline (assumed to be equal to the period T_s), P_s is message priority, F_s is number of frame and f_s is size of message frame of the transmitted sensor data. The worst case transmission time L_{sc} of the message in an error-free scenario is given by

$$L_{sc} = \frac{F_s f_s}{B} \tag{2}$$

Controller task is executed at controller node, responsible to retrieve sensor value from network, calculate desired control signal value based on dedicated control algorithm and send it to actuator node through network. Controller task is event driven that will be executed once controller node receive sensor data from sensor node via CAN network. Similarly, the controller can be defined as $TS_c = (C_c, D_c, P_c, F_c, f_c)$ with execution time C_c, relative deadline D_c, a message priority P_c, a number of frame F_c and size of message frame f_c of transmitted control signal data. In an error-free situation, the worst case transmission time L_{ca} of the message can be defined as

$$L_{ca} = \frac{F_c f_c}{B} \tag{3}$$

At the actuator node actuator task TS_a, is responsible to retrieve control signal value and send it to the physical input of system. It should be noted that there are signal conditioning and scaling processes involved that causes some delay in the actuation, however, the magnitude of the delay is significantly small as compared to transmission time and the delay effect can be ignored. TS_a is event driven which it will be executed upon receiving control signal data from controller node. The task, with only two parameters can be represented as $TS_a = (C_a, D_a)$ has execution time C_a and relative deadline D_a. In this work, it is assumed that sensor data and control data are transmitted in single frame, i.e. $F_c = F_s = 1$. Sensor data is set to have the highest priority while the priority of control data is set to be the next lower to sensor data priority.

Since interest of this work is on the overcoming error transmission in CAN, it is required to characterize the errors model before hand. Error model of this paper consists of the following parameters:

(i) n^i_{sc}: the number of error occurrences for sensor data which is transmitted from sensor node to controller node for every sampling instant i in period of T_s, i.e. $n^i_{sc} = 1$ for single error or $n^i_{sc} > 1$ for burst error.
(ii) n^i_{ca}: the number of consecutive error for control data which is transmitted from controller node to actuator node for every sampling instant i in period of T_s, i.e. $n^i_{ca} = 1$ for single error or $n^i_{ca} > 1$ for burst error.
(iii) N: the maximum allowable number of error bursts that occur in NCS data transmission in every sampling instant, as in $N = n^i_{sc} + n^i_{ca}$
(iv) E_{sc}: the error rate for sensor data in a given time t
(v) E_{ca}: the error rate for control signal data in a given time t.

The sensor data error rate, E_{sc} for parameter (iv) and control data error rate, E_{ca} parameter in (iv) and (v) can be calculated as follows:

$$E_{sc} = \frac{n_{e_{sc}}}{n_t} \tag{4}$$

$$E_{ca} = \frac{n_{e_{ca}}}{n_t} \tag{5}$$

where $n_{e_{sc}}$, $n_{e_{ca}}$ and n_t are the number of error occurrences for sensor data, control data and total number sampling instant in a given time, respectively.

4. NCS with delay model

A continuous time linear time invariant (LTI) system can be described as state space model

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + du \end{aligned} \tag{6}$$

where $x(t)$, $u(t)$ and $y(t)$ denote the state, control input and output vectors, respectively. A, B, C and D are matrices of appropriate sizes where A is state matrix, B is input matrix, C is output matrix and D is feedforward matrix. Nowadays, since the control system is prominently interfaced and executed by digital computer (i.e. microprocessor), the system (6) can be represented in discrete form. With having zero-order-hold element on its input and sampling time T_s, system (6) becomes

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k) \\ y(k) &= C_d x(k) + D_d u(k) \end{aligned} \tag{7}$$

where

$$A_d = e^{Ah}, \quad B_d = \int_0^{T_s} e^{At} dtB$$

$$C_d = C, \quad D_d = D$$

When system (6) is connected to NCS, the network will introduce delay in data transmission and (7) can be represented as

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k - \tau^i_{ca}) \\ y(k) &= C_d x(k) + D_d u(k - \tau_{ca}) \end{aligned} \tag{8}$$

where τ^i_{ca} represent controller to actuator delay at every sampling instant ith.

Assume that state feedback controller is designed for input of system (7), thus control data can be described as

$$u_d = -Kx_d(k - \tau^i_{sc}) \tag{9}$$

where τ^i_{sc} is sensor to controller delay in every sampling instant ith and K is controller gain. The value of K can be determined by using various established methods such as pole placement or linear quadratic regulator (LQR). Fig. 3 shows data transmission of sensor data and control data via network under delay influence. In NCS, the selection of sampling time T_s should be properly chosen since high sampling rate can increase network load, thus leads to network congestion and data loss, which in turn result in longer delay of the signals. On the other hand, lower sampling rate will make the system less tolerates to time delay. The 'rules of thumb' used by many reported works in selecting the sampling time is to choose $T_s > 10\tau$ where τ is the known time constant of the actual physical system to be controlled, however from the view of digital hardware execution, the speed and resources of the available computing resource should be able to support the required sampling time.

In LTI system theory, τ^i_{sc} and τ^i_{ca} can be lumped together, such that

$$\tau^i_k = \tau^i_{sc} + \tau^i_{ca} \tag{10}$$

where τ^i_k is known as total loop delay.

Other then selection of appropriate sampling time, it is also important to select the so-called maximum allowable loop delay (MALD) first in order to do the stability analysis of NCS. The relationship between sampling time and maximum allowable loop delay are as follows [31]

$$T_s + \Phi = \eta \tag{11}$$

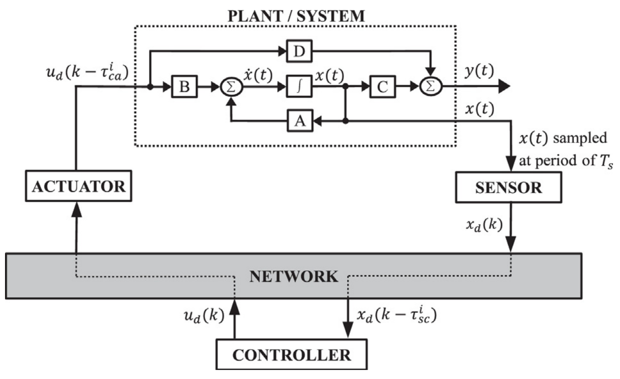

Fig. 3. Data transmission on NCS with state feedback controller.

where η is maximum allowable equivalent delay bound and Φ is value of MALD as in the maximum value of τ_k^i.

5. New error handling mechanism

In this section, we introduce per-sample-error-counter (PSeC) mechanism, which purposely designed to replace native error handling of CAN for NCS application. PSeC mechanism is operating based on parameters n_{sc}^i, n_{ca}^i and N, which are defined in Section 3. The PSeC algorithm is explained as follows:

(1) At sensor node, scheduler runs sensor task to obtain sensor reading and send it to network. If error occurs when transmitting sensor data to controller node, scheduler re-executes sensor task to obtain new sensor reading and send it to network. If $n_{sc}^i > N$, sensor data will not be sent to controller.
(2) Once controller node obtain sensor data from network, scheduler will run controller task to calculate appropriate control signal and send it to actuator node via network. If error occurs when transmitting control data to actuator node, scheduler will re-attempt to transmit previously sent control data. If $n_{ca}^i > N - n_{sc}^i$, control signal will not be sent to actuator. This is to prevent network overload at next instant of time, ith.

Fig. 4 shows the flowchart of the newly proposed PSeC algorithm for clearer explanation. Note that the block (A) and (B) in the figure show that the algorithm is implemented in sensor node and controller node respectively. The maximum allowable

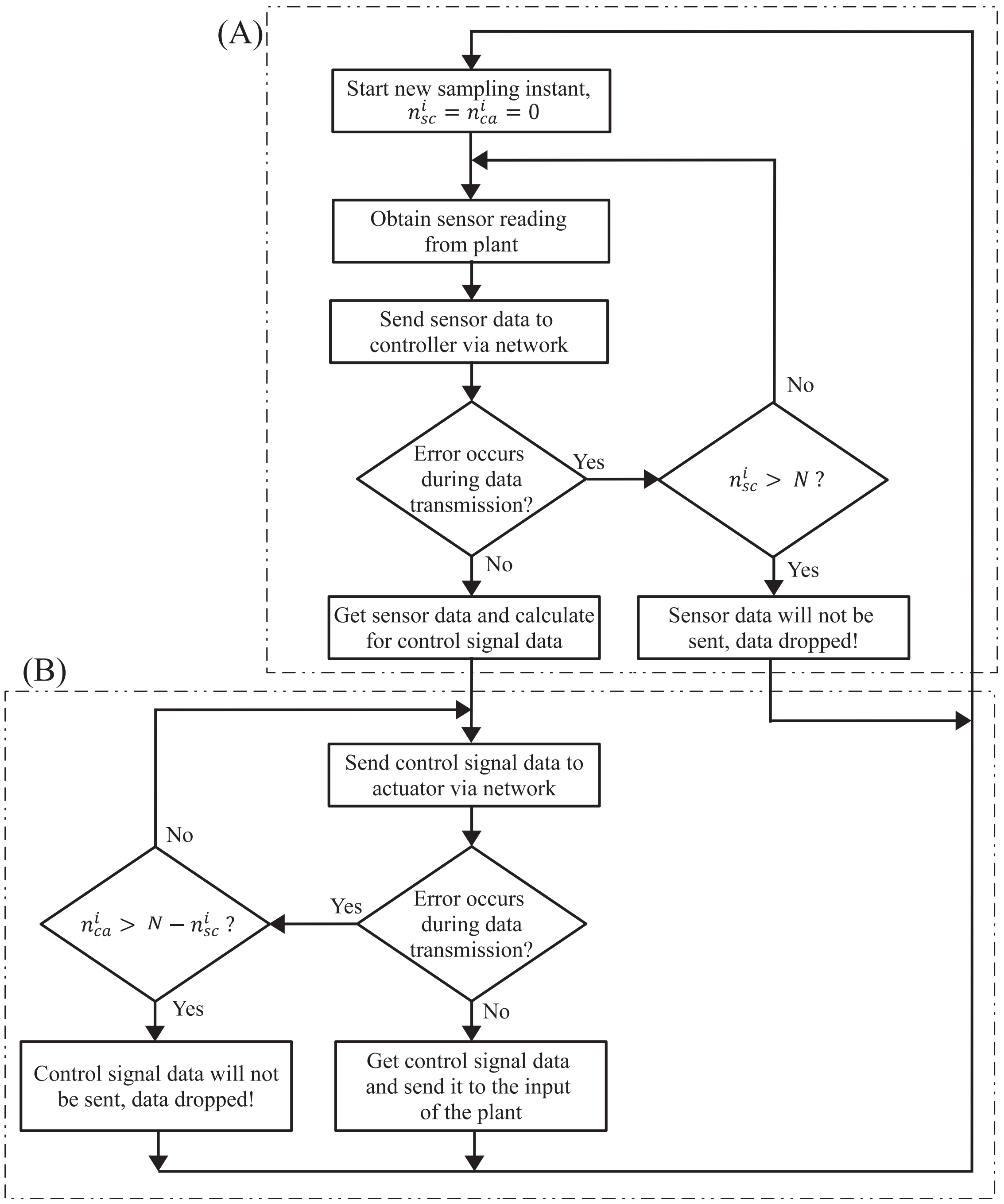
(A)
Start new sampling instant, $n_{sc}^i = n_{ca}^i = 0$
Obtain sensor reading from plant
Send sensor data to controller via network
Error occurs during data transmission?
Yes
No
$n_{sc}^i > N$?
No
Yes
Get sensor data and calculate for control signal data
Sensor data will not be sent, data dropped!
(B)
Send control signal data to actuator via network
Error occurs during data transmission?
Yes
No
$n_{ca}^i > N - n_{sc}^i$?
No
Yes
Control signal data will not be sent, data dropped!
Get control signal data and send it to the input of the plant

Fig. 4. Flowchart of PSeC algorithm.

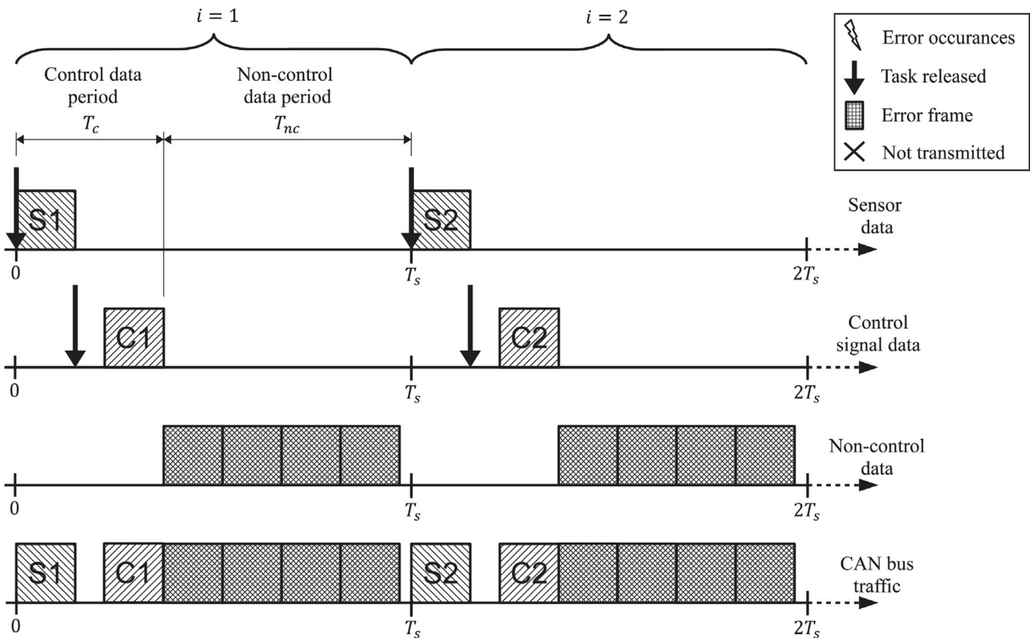

Fig. 5. Windows of control data period and non-control data period in error free situation.

number of error bursts (MAEB), N will be derived in the next section. In order to perform this proposed mechanism at sensor and control nodes, it is require that both nodes should be in single-shot transmission mode. This can be achieved by disabling the automatic retransmission mechanism in CAN protocol. Therefore, in order to use this algorithm, it is required to choose CAN controllers that have this particular feature, e.g. Atmel T89C51CCO2, Philips SJA1000 or Microchip MCP2515 [23]. Also, in native error handling of CAN, all transmitted error frame will be counted and recorded as Receive Error Counter (REC) and Transmit Error Counter (TEC) by the affecting node. If value of TEC exceeds 255, the node will going to bus-off to prevent the node to transmit or receive any frame. However, in this new error handling mechanism, this feature should be disabled.

In NCS, non-control data may exist where it is required to be transmitted via the same network. Example of non-control data is the notification of event or sensor data for monitoring purpose. Non-control data can be transmitted after control data transmission period. Therefore, in every sampling time T_s of NCS, the bandwidth will be decomposed into two segments: control data period and non-control data period. The control data period T_c is allocated at the beginning of every interval T_s. Therefore, there is a residual bandwidth $T_s - T_c$, denoted as T_{nc}, which can be allocated to transmit non-control data. The window for control data period T_c and non-control data period T_{nc} in every interval T_s is illustrated in Fig. 5.

Non-control data can be generated by any nodes in network and the data should be assigned to lower priority than control data. Note that the non-control data might not be transmitted when the error occurred in control data transmission in order to give access for PSEC mechanism to perform data retransmission as shown in Fig. 6. Hence, non-control data that is scheduled at period T_{nc}

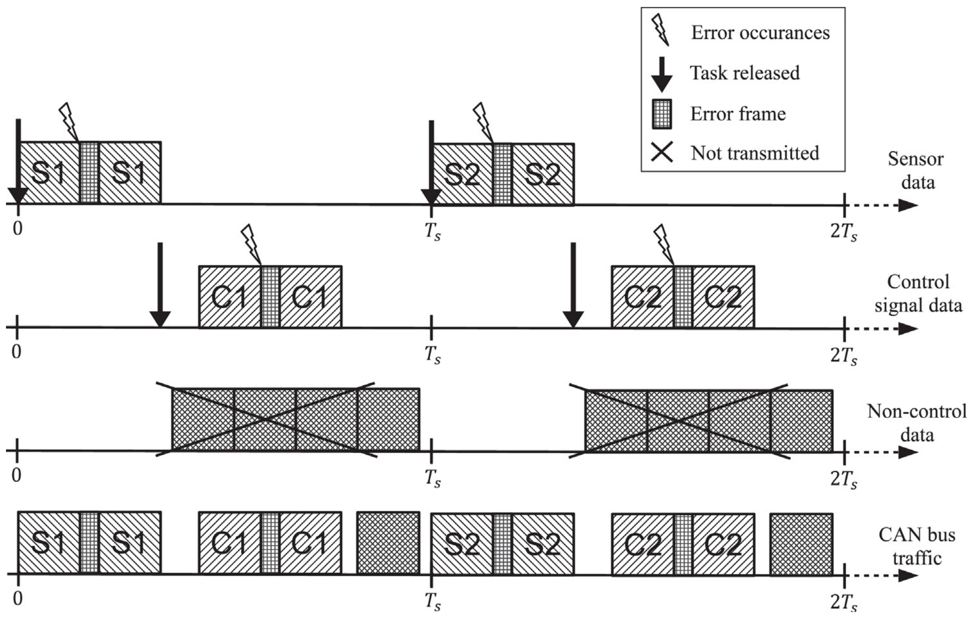

Fig. 6. Data transmission in CAN under error occurrences situation. Some of non-control data is not transmitted.

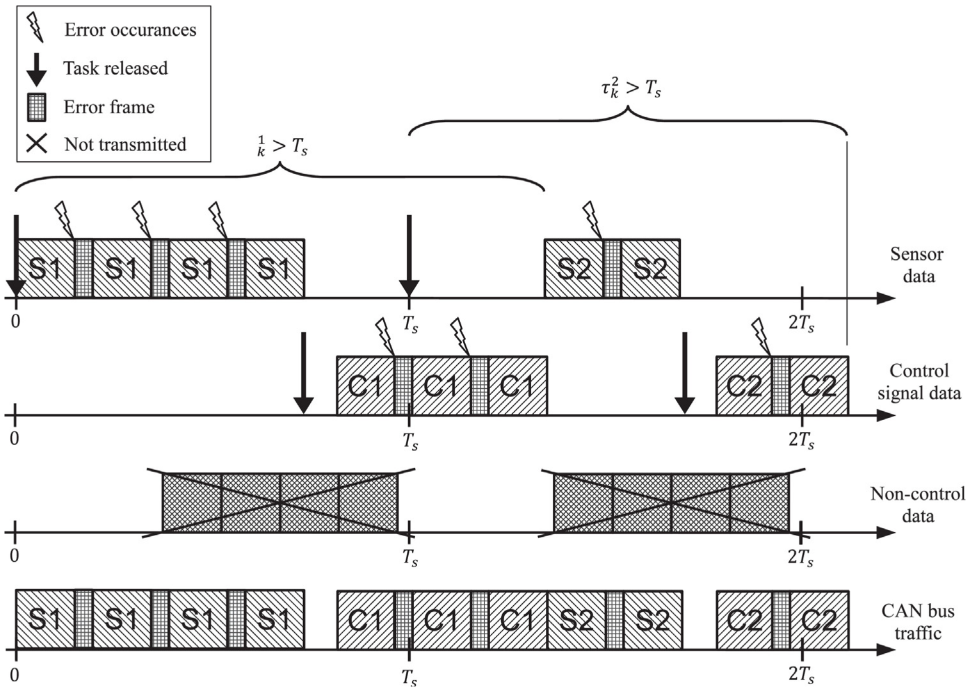

Fig. 7. Data transmission in CAN when error occur at control data using native error handling of CAN.

should be a non-critical message which does not induce undesired consequences if the data is not transmitted.

Data recovery mechanism that is provided by this PSeC algorithm is different from the native error handling of CAN protocol, where this built-in CAN protocol will attempt to retransmit previous sensor data when there is an error in sensor data transmission. However, the drawback is that the CAN mechanism does not provide the facility to monitor the number of consecutive data error bursts in every sampling instant and it keeps retransmitting the data until it is successfully transmitted. This may cause the control data to be transmitted after sampling period of T_s and it will lead to an increasing loop delay on the next sampling instant as illustrated in Fig. 7. On the other hand, PSeC will obtain an updated value of sensor data when there is an error in transmitting sensor data and also keep track the number of error occurrences of n_{sc}^i and n_{ca}^i in every sampling instant. If sensor and control data are unable to be transmitted in the period of T_s, these data will be dropped, hence the data transmission on next sampling instant is not affected as shown in Fig. 8. This will lead to a better performance since in controlling dynamical systems, a greatly delayed data is more harmful than no data at all [32].

6. Delay analysis and stability condition

In NCS, delay can degrade the performance of NCS and in the worst case, it can destabilize the system. Therefore, the analysis of delay is important in an attempt to preserve the system stability. There are some assumptions have to be made to perform the

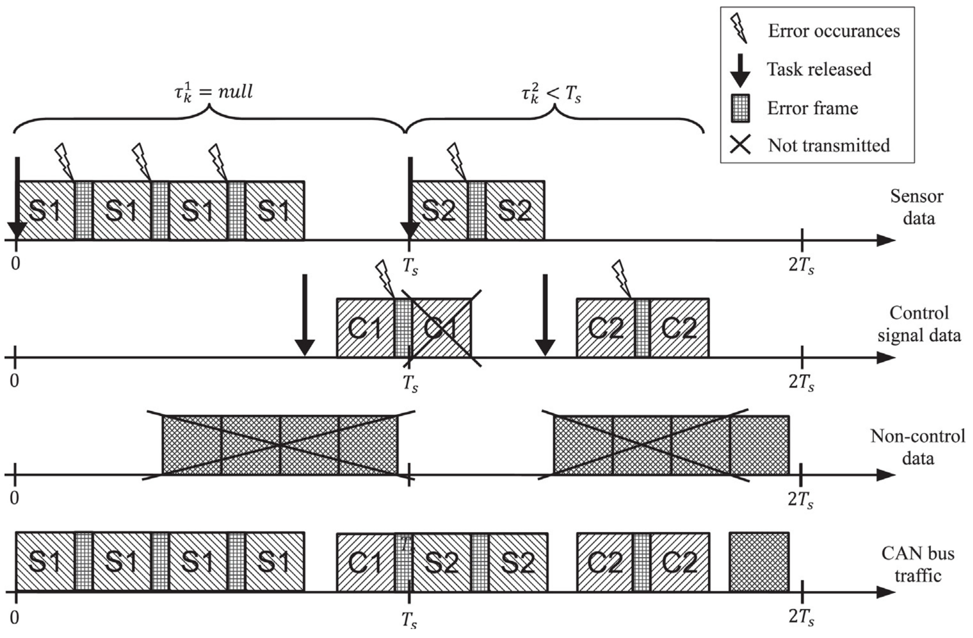

Fig. 8. Data transmission in CAN when error occurs at control data using proposed error handling mechanism.

analysis. Some of the assumptions have been established in the previous section, however, for clarity purposes, they are re-iterated with other newly proposed assumptions, where the lists can be stated as follows:

(i) The analysis is done based on worst case scenario. Thus, sensor data and control data are assumed to have maximum length size of 135 bits. The worst case error frame size is 31 bits.
(ii) Filtering, buffering and packetizing delays are neglected.
(iii) All error occurrences can be detected.
(iv) No clock drifts in the system.
(v) All non-control data are non-critical messages.

The NCS configuration could be connected to a system with multi-input multi-output (MIMO) model. Thus, there are few sensors and actuators could be connected to sensor node and actuator node. For this case, all obtained sensor values are packed into one frame and be transmitted to the controller node. The calculated control signal for multiple input are also transmitted in one frame. In some cases especially in safety critical applications, several sensors are required to measure each state variable of the system. The readings of these sensors are then filtered by filtering circuit or algorithm in order to get stable and reliable measurement values before packetizing into data frame. Filtering circuit or algorithm will contribute some delay in the network, however, in this analysis, this type of delay is assumed relatively small and can be ignored. Fig. 9 illustrates the sensor configuration of the system, and sensor and control data that are being packetized into single CAN frame.

The analysis requires that the clock of all nodes in the network to be synchronized in order to minimize the influence of jitter in final results. Clock synchronization scheme on CAN protocol is can be done by means of hardware implementation as proposed in [33] or by means of clock synchronization in which two clock synchronization messages are transmitted successfully, as proposed in [34]. In this work, it is reasonably assumed that the period of clock synchronization is much larger than sampling time T_s in order to avoid the interference between the algorithm and clock synchronization process.

The delay analysis is divided into two situations: (1) normal network condition and (2) in the event of error occurrence situation. The delay terms τ_{sc}^i and τ_{ca}^i which describe this situation can be established as

$$\tau_{sc}^i = L_{sc} + n_{sc}^i(L_e + L_{sc}) \quad (12)$$

$$\tau_{ca}^i = C_c + C_a + L_{ca} + n_{ca}^i(L_e + L_{ca}) \quad (13)$$

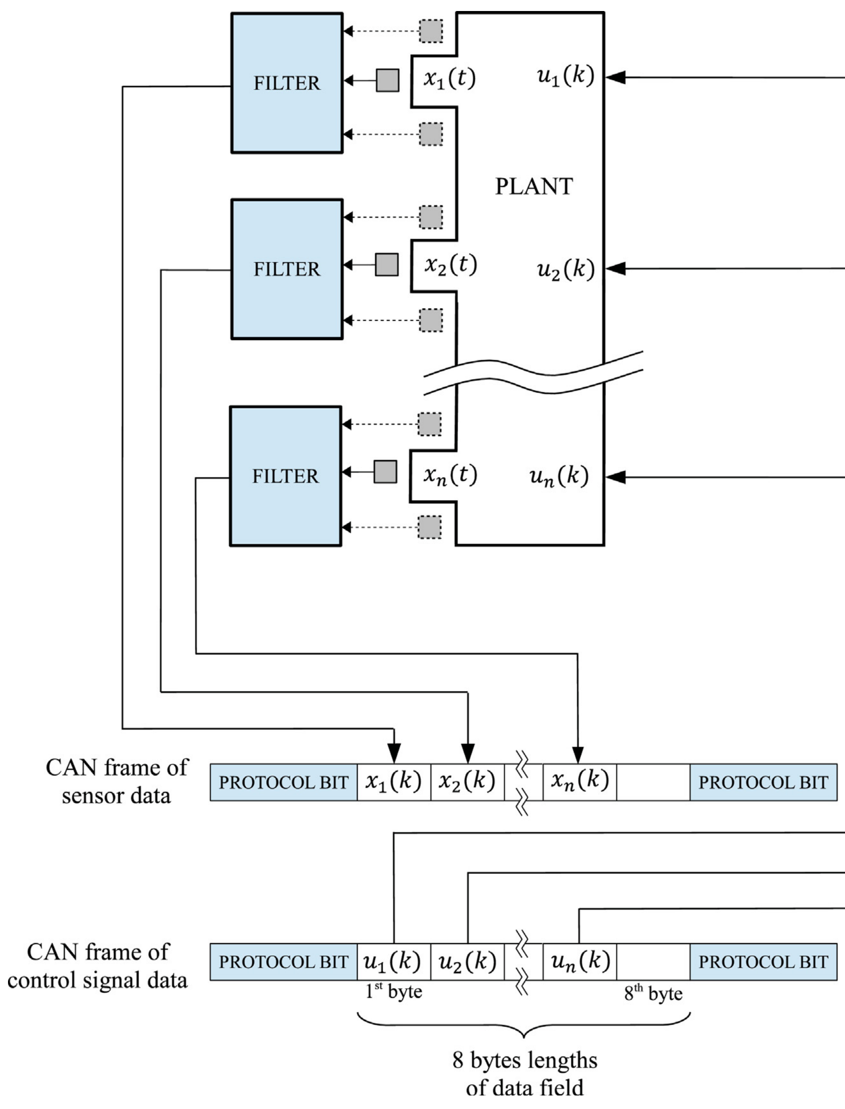

Fig. 9. Sensor configuration of system and the packetizing of sensor data and control single data into single CAN frame.

where n_{sc}^i and n_{ca}^i is a positive integer. L_{sc}, L_{ca} and L_e are transmission time for sensor to controller, controller to actuator and error frame, respectively, while C_c is execution time of controller tasks and C_a is execution time of actuator task.

Based on assumption (i) in Section 6, the length of sensor data and control data are identical, hence $L_c = L_{sc} = L_{ca}$. Also, for the sake of simplicity, it can be established that the relationship of L_e in term of L_c, subjected to (1), can be deduced to

$$L_e \approx \frac{L_c}{4} \tag{14}$$

From (10), (12) and (13), loop delay τ_k^i can be obtained as

$$\tau_k^i = C_c + C_a + 2L_c + \frac{5L_c}{4}(n_{sc}^i + n_{ca}^i) \tag{15}$$

Based on (15), it is obvious that in normal condition (i.e. $n_{sc}^i = n_{ca}^i = 0$), the delay τ_{sc}^i and τ_{ca}^i should be constant for every instant ith. However when there are occurrences of transmission error in network, the delay τ_{sc}^i and τ_{ca}^i will become random. Error occurrences in some actual systems are usually assumed to be governed by probability distribution, e.g. Poisson distribution [17,19], however in our simulation, errors are set to occur periodically with constant error bursts length where it reflects error occurrence in most control system with repetitive processes [35].

In order to maximize MALD and to prevent network overload at every sampling instant, it is proposed that $T_s = \Phi$, and thus sampling time T_s can be determined as follows:

$$T_s = \frac{\eta}{2} \tag{16}$$

Also in order to preserve the stability of NCS, loop delay τ_k^i should satisfy the following condition:

$$\tau_k^i \le T_s \tag{17}$$

From (15) and (17), the sum of error burst n_{sc}^i and n_{sc}^i, denoted as N should be restricted to the following inequality

$$N = n_{sc}^i + n_{ca}^i \le \left\lfloor \frac{4(T_s - C_c - C_a - 2L_c)}{5L_c} \right\rfloor \tag{18}$$

If the number of error burst n_{sc}^i and n_{ca}^i violate the inequality (18), it indicates that the control data is not able to be transmitted within T_s.

The control data period T_c can be obtained from (15), when the network is operated in normal operating condition without any error occurred in data transmission, i.e. $n_{sc}^i = n_{ca}^i = 0$. Thus,

$$T_c = C_c + C_a + 2L_c \tag{19}$$

Hence the period for non-control data transmission T_{nc} is

$$T_{nc} = T_s - T_c \tag{20}$$

7. Simulation result and discussion

In order to show the effectiveness of the proposed method, inverted pendulum mounted on a cart as shown in Fig. 10 is used as the testbed. Inverted pendulum on a moving cart is a fourth order system that serves as a very good example to illustrate the control performance. Rotational and linear encoders are attached to measure the control variables of the system which are the pendulum angle, $\theta(t)$, and the cart linear displacement, $z(t)$. It is assumed that the values of gravity acceleration $g = 10\ \mathrm{m\,s^{-2}}$, mass of the pendulum $m = 0.1$ kg, mass of the cart $M = 67$ kg and distance from the mass m to the pivot point $l = 1$ m. It is also assumed that the variation of pendulum angle from vertical $\theta(t)$, is relatively small so that the equation is linear. $z(t)$ is the position of

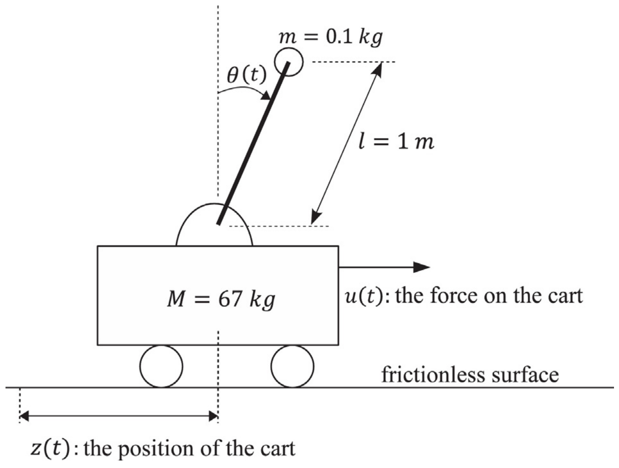

Fig. 10. An inverted pendulum system mounted on a cart.

the cart from reference point and $u(t)$ is the force that applied to the cart. By choosing the state variables as $x_1(t) = y(t)$, $x_2(t) = \dot{y}(t)$, $x_3(t) = \theta(t)$, $x_4(t) = \dot{\theta}(t)$, and the outputs of interest are $x_1(t)$ and $x_3(t)$, one can obtain the following state space model (more discussion about this model can be found in [36]):

$$\dot{x}_1(t) = x_2(t) \tag{21}$$

$$\dot{x}_2(t) = \frac{1}{M}x_3(t) + \frac{1}{M}u(t) \tag{22}$$

$$\dot{x}_3(t) = x_4(t) \tag{23}$$

$$\dot{x}_4(t) = 0x_3(t) + \frac{1}{M}u(t) \tag{24}$$

By rearranging Eqs. (21)–(24), the matrices of the system as described in (6) becomes

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -0.015 & 0 \\ 0 & 0 & 0 & 01 \\ 0 & 0 & 10 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0.015 \\ 0 \\ -0.015 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The initial conditions for system are $x_3(0) = 0.1$ and $x_1(0) = x_2(0) = x_4(0) = 0$. With CAN speed set to $B = 125$ kbps and using the assumption (i) at Section 6, length of sensor and control data L_c can be calculated using (2) or (3), which yields $L_c = 1.08$ ms.

The desired discrete closed loop poles for controller design are chosen as 0.0498, 0.0183, 0.0067 and 0.0025. It is found that the maximum allowable equivalent delay bound for chosen poles is $\eta = 90$ ms and using (16), sampling time of the system can be determined such that $T_s = 45$ ms. Then, the continuous time system (6) can be transformed to discrete system of (7), yields

$$A_d = \begin{bmatrix} 1 & 0.045 & 0 & 0 \\ 0 & 1 & -0.0007 & 0 \\ 0 & 0 & 1.01 & 0.0452 \\ 0 & 0 & 0.4515 & 1.01 \end{bmatrix}, \quad B_d = \begin{bmatrix} 0 \\ 0.0007 \\ 0 \\ -0.0007 \end{bmatrix}$$

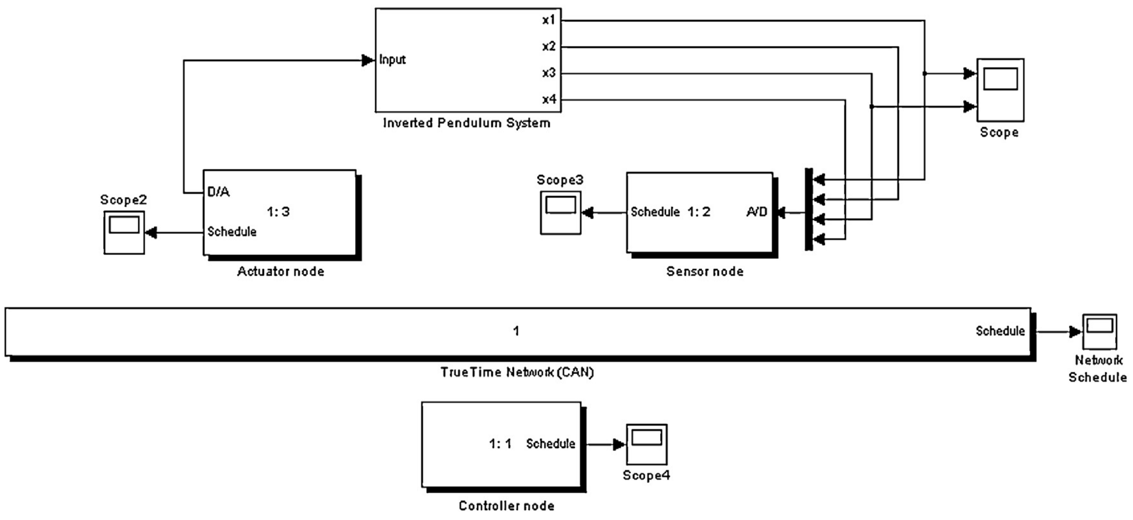

Fig. 11. The screenshot of TrueTime simulation environment.

$$C_d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad D_d \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

By using pole placement method and the calculated sampling time, the gains of state feedback controller can be determined as $K = 1 \times 10^3[-1.5747 - 1.3816 - 6.4567 - 2.3844]^T$. By setting the controller execution time $C_c = 4$ ms and actuator execution time $C_a = 0.5$ ms, the non-control data period can be determined from (19) and (20), yields $T_{nc} = 38.34$ ms. Also the value of maximum consecutive error burst N can be calculated from (18), such that

$$N = n_{sc}^i + n_{ca}^i \le 28 \tag{25}$$

The simulation is performed until 10 s, which constitutes 222 sampling instances (i.e. $t = 10$, $i = 1, 2, 3, \ldots, 222$, $n_t = 222$). The control objective of this system is to drive $z(t)$ and $\theta(t)$ from their initial conditions to zero with least amount of overshoot in less than 4 s. The performance of the proposed error handling mechanism is compared to native error handling of CAN where the performance measure is defined as integral square error (ISE)

$$ISE = \int_0^\infty [r(t) - c(t)]^2 dt \tag{26}$$

where $r(t)$ is desired trajectory and $c(t)$ is output parameter where the value need to be measured to show their performance. In this case, $r(t) = 0$. The simulation results are categorized into 4 cases:

(1) 25% data error rate with $n_{sc}^i = n_{ca}^i = 7$.
(2) 75% data error rate with $n_{sc}^i = n_{ca}^i = 7$.
(3) 25% data error rate with $n_{sc}^i = n_{ca}^i = 15$.
(4) 75% data error rate with $n_{sc}^i = n_{ca}^i = 15$.

Simulation is performed by using Matlab/TrueTime simulator. TrueTime is a Matlab/Simulink-based simulator for real-time control systems which facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics [37]. Using TrueTime, one can easily verify the analysis of NCS under influence of different scheduling scheme, task execution, network delay and sampling time. Fig. 11 shows the screenshot of simulation environment where sensor, actuator and controller node are constructed from TrueTime Kernel block. The network is however developed from TrueTime network block. The details procedure to setup each block can be referred to [38]. The pseudocode for the sensor, controller and actuator node are listed in Pseudocodes 1, 2 and 3, respectively. Lines 12–19 at Pseudocode 1 and lines 11–17 at Pseudocode 2 reflect the PSeC mechanism, and variable *nsc* and *nca* should be declared as global variables.

Pseudocode 1
Algorithm for sensor node.

```
1:   SamplingTime = 0.045
2:   FrameSize = 135
3:   $N = 28$
4:   $i = 0$
5:   REPEAT every SamplingTime
6:   {
7:     nsc = 0;
8:     $i = i + 1$;
9:     READ SensorsValue
10:    SEND SensorsValue to controller node
11:    WHILE CurrentSimulationTime < $i$*SamplingTime
12:      IF error frame detected
13:        READ SensorsValue
14:        SEND SensorsValue to controller node
15:        nsc = nsc + 1;
16:        IF nsc > $N$
17:          BREAK
18:          ENDIF
19:        ENDIF
20:      ENDWHILE
21:  }
```

Pseudocode 2
Algorithm for controller node.

```
1:   SamplingTime = 0.045
2:   FrameSize = 135
3:   $N = 28$
4:   $i = 0$
5:   IF receive sensor data
6:     nca = 0
7:     $i = i + 1$;
8:     Calculate ControlSignal
9:     SEND ControlSignal to actuator node
10:    WHILE CurrentSimulationTime < i*SamplingTime
11:      IF error frame detected
12:          SEND ControlSignal to actuator node
13:          nca = nca + 1;
14:          IF nca > N – nsc
15:          BREAK
16:        ENDIF
17:        ENDIF
18:      ENDWHILE
19:  ENDIF
```

Pseudocode 3
Algorithm for actuator node.

```
1: IF receive control data
2:    WRITE ControlSignal
3:    SEND ControlSignal to input of system
4: ENDIF
```

Figs. 12 and 15 show the simulation results when number of consecutive error bursts is $n^i_{sc} = n^i_{ca} = 7$. Under 25% error rate, it can be noticed that the difference in the responses are very minimal. However, under 75% data error rate, the PSeC method gives better performance than native error handling of CAN in term of overshoot size. The different value of $z(t)$ and $\theta(t)$ at overshoot time are $\Delta z(t) = 0.0315$ m and $\Delta\theta(t) = 0.0195$ = rad. This is predictable since in this algorithm, in the occurrence of errors, the updated data will be transmitted instead of previous value of sensor data, as shown in block (A) in Fig. 4. The loop delay of the system for both error handling mechanisms under 25% data error rate are varied from 16.11 ms to 6.66 ms as shown in Figs. 13 and 14, while under 75% data error rate, loop delay plot is also changed from 16.11 ms to 6.66 ms but different plot pattern as shown in Figs. 16 and 17.

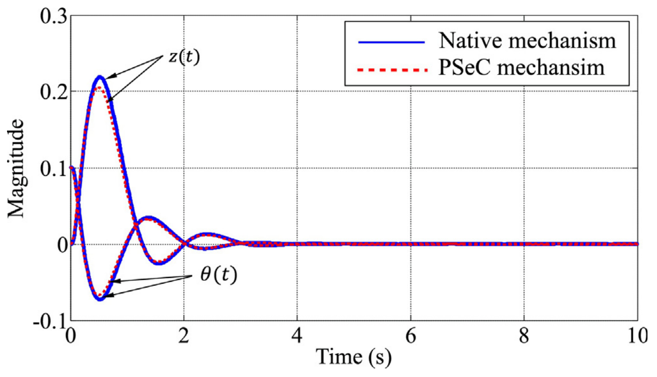

Fig. 12. Response of the system for case 25% data error rate with $n^i_{sc} = n^i_{ca} = 7$.

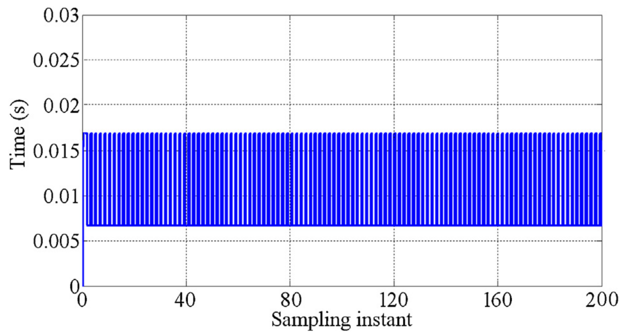

Fig. 13. Loop delay of the system for case 25% data error rate and $n^i_{sc} = n^i_{ca} = 7$ using native error handling of CAN.

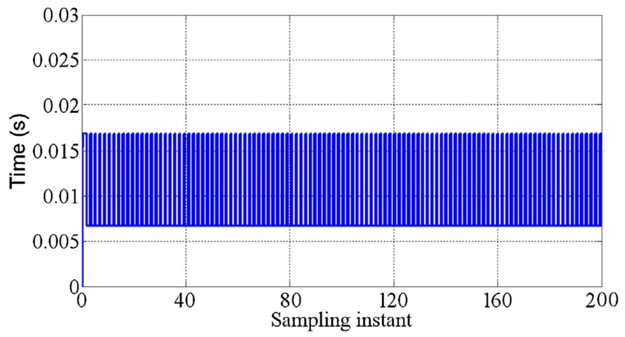

Fig. 14. Loop delay of the system for case 25% data error rate with $n^i_{sc} = n^i_{ca} = 7$ using PSeC mechanism.

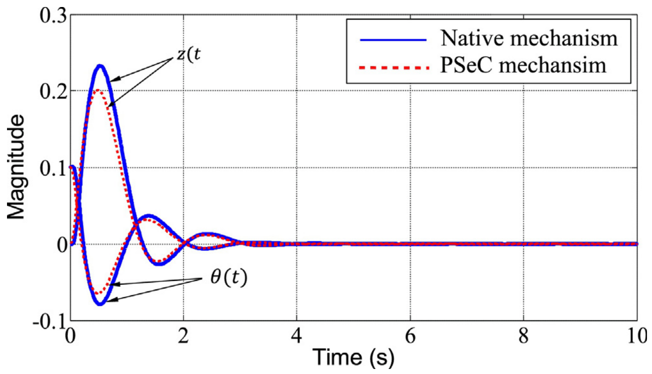

Fig. 15. Response of the sytem for case 25% data error rate with with $n^i_{sc} = n^i_{ca} = 15$.

Figs. 18 and 21 show the simulation when the number of consecutive error $n^i_{sc} = n^i_{ca} = 15$, and these values violate the stated bound (25). It is found that with 25% data rate error, both error handling mechanisms still can preserve the system stability but the performance of the system has degraded since the loop delay is larger than the delay in the case $n^i_{sc} = n^i_{ca} = 7$. The loop of the system for native error handling of CAN varying from 47.16 ms and 6.66 ms as shown in Fig. 19 while loop delay for the new error handling mechanism is altering from 26.91 ms, 6.66 ms and 0 ms, as shown in Fig. 20. It should be noted that 0 ms of loop delay means the data is dropped and will not be transmitted to the actuator of the inverted pendulum. The difference between the overshoot value of $z(t)$and $\theta(t)$ are $\Delta z(t) = 0.032$ m and $\Delta\theta(t) = 0.0143$ rad. Under 75% data error rate, native CAN error

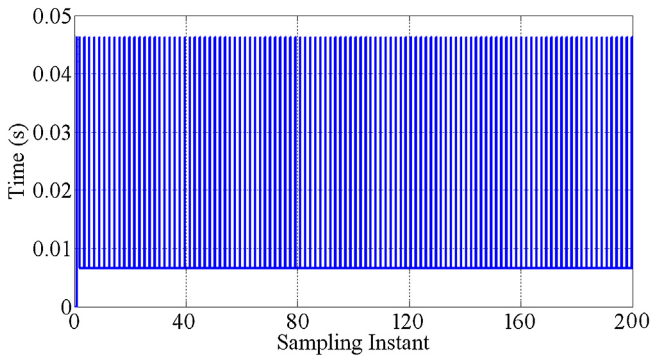

Fig. 16. Loop delay of the system for case 25% data error rate and $n^i_{sc} = n^i_{ca} = 15$ using native error handling of CAN.

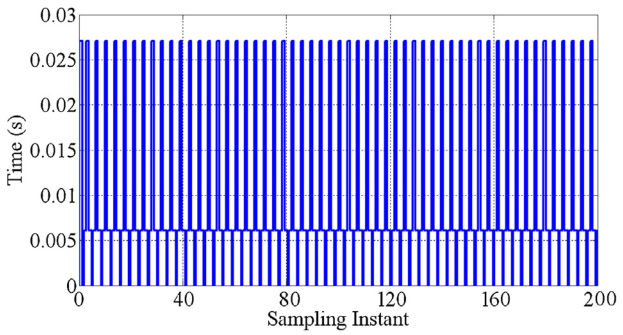

Fig. 17. Loop delay of the system for case 25% data error rate with $n^i_{sc} = n^i_{ca} = 15$ using PSeC mechanism.

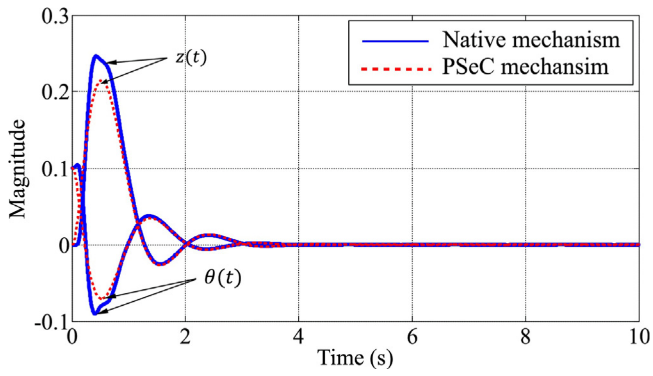

Fig. 18. Response of the system for 75% data error rate with with $n_{sc}^{i} = n_{ca}^{i} = 7$.

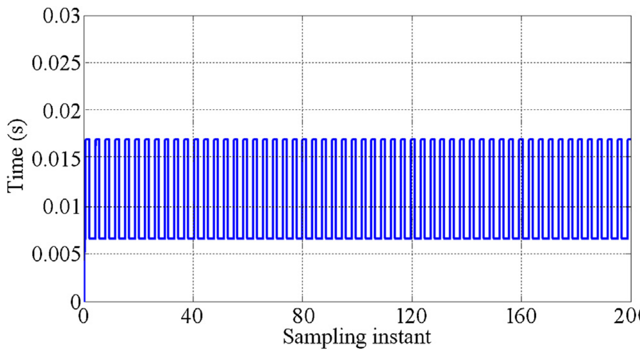

Fig. 19. Loop delay of the system for case 75% data error rate and $n_{sc}^{i} = n_{ca}^{i} = 7$ using native error handling of CAN.

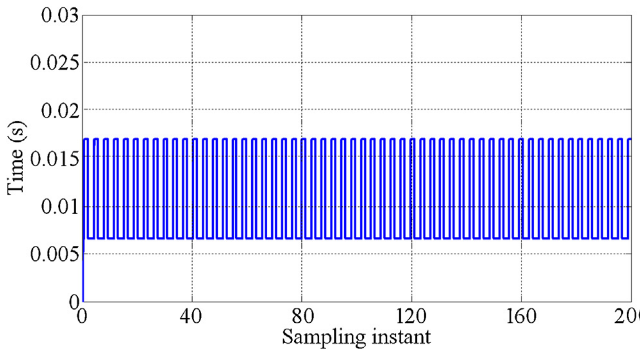

Fig. 20. Loop delay of the system for case 75% data error rate with $n_{sc}^{i} = n_{ca}^{i} = 7$ using PSeC mechanism.

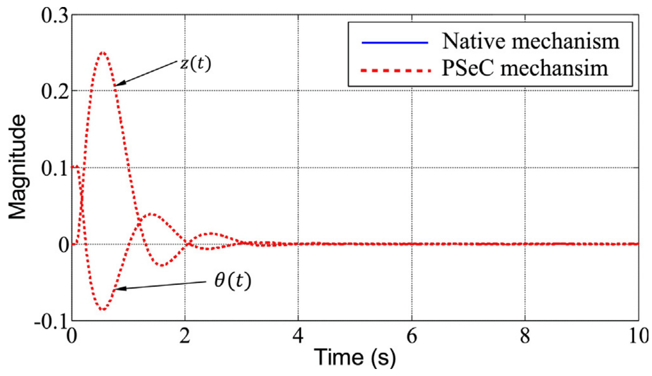

Fig. 21. Response of the system for case 75% data error rate with $n_{sc}^{i} = n_{ca}^{i} = 14$.

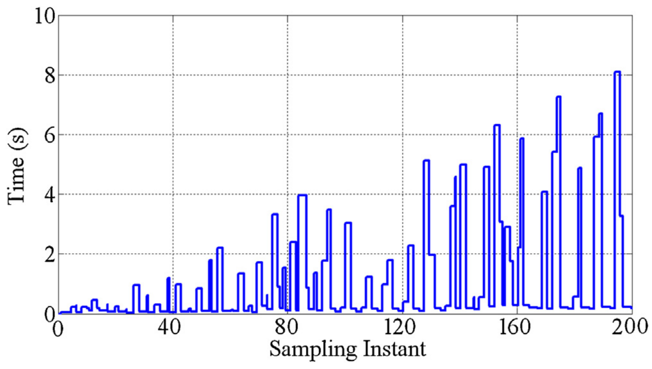

Fig. 22. Loop delay of the system for case 75% data error rate and $n_{sc}^{i} = n_{ca}^{i} = 15$ using native error handling of CAN.

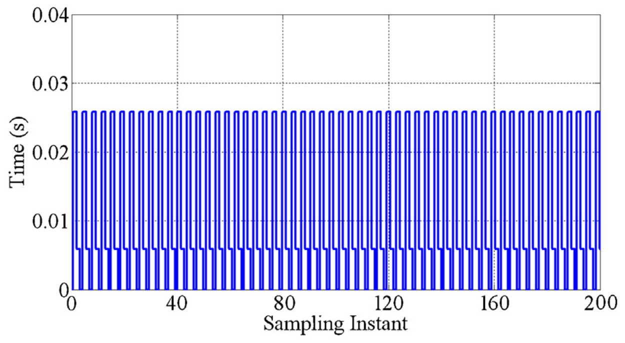

Fig. 23. Loop delay of the system for case 75% data error rate with $n_{sc}^{i} = n_{ca}^{i} = 15$ using PSeC mechanism.

handling mechanism is unable to preserve system stability due to network congestion (not shown in Fig. 21). As can be observed in Fig. 22, this condition has led to increasing loop delay over sampling instances, T_s that is caused by repetitive re-transmission of control data. This situation, however does not occur under PSeC algorithm and it is obvious that the stability of the system is still preserved. The mechanism will drop the data when the MAEB, N exceeds the maximum bound (25) and thus prevents network congestion. Fig. 23 shows the loop delay for PSeC mechanism is varying from 26.91 ms, 6.66 ms and 0 ms.

Table 1
Performance comparison of native error handling of CAN and PSeC mechanism for case $n_{sc}^{i} = n_{ca}^{i} = 7$.

Data transmission type	Error rate (%)	Error burst pattern (error occur at sampling instant i)	Native error handling of CAN		PSeC mechanism	
			$z(t)$	$\theta(t)$	$z(t)$	$\theta(t)$
Sensor to controller	25	1 of every 4 samples, e.g.: i = {3, 7, 11, 15, 19, 23, ..., 443}	25.85×10^{-3}	3.703×10^{-3}	22.81×10^{-3}	3.133×10^{-3}
Controller to actuator	25	1 of every 4 sample, e.g.: i = {1, 5, 9, 13, 17, 21, ..., 441}				
Sensor to controller	75	3 of every 4 samples, e.g.: i = {1, 2, 3, 5, 6, 7, 9, 10, 11, 13, ..., 443}	32.41×10^{-3}	5.535×10^{-3}	25.12×10^{-3}	3.501×10^{-3}
Controller to actuator	75	3 of every 4 samples, e.g.: i = {2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 15, ..., 444}				

Table 2
Performance comparison of native error handling of CAN and PSeC mechanism for case $n_{sc}^{i} = n_{ca}^{i} = 15$.

Data transmission type	Error rate (%)	Error burst pattern (error occur at sampling instant i)	Native error handling of CAN		PSeC mechanism	
			$z(t)$	$\theta(t)$	$z(t)$	$\theta(t)$
Sensor to controller	25	1 of every 4 samples, e.g.: i={3, 7, 11, 15, 19, 23, . . ., 443}	29.03×10^{-3}	4.373×10^{-3}	21.93×10^{-3}	2.974×10^{-3}
Controller to actuator	25	1 of every 4 sample, e.g.: i={1, 5, 9, 13, 17, 21, . . ., 441}				
Sensor to controller	75	3 of every 4 samples, e.g.: i={1, 2, 3, 5, 6, 7, 9, 10, 11, 13, . . ., 443}	*Unstable*	*Unstable*	33.5×10^{-3}	5.237×10^{-3}
Controller to actuator	75	3 of every 4 samples, e.g.: i={2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 15, . . ., 444}				

Tables 1 and 2 shows the performance comparison of both PSeC and native CAN error handling mechanism based on (26) where lower value of *ISE* means a better control system performance. For case $n_{sc}^{i} = n_{ca}^{i} = 7$, under 25% data error rate, the *ISE* different performance of $z(t)$ and $\theta(t)$ are $\Delta ISE(z(t)) = 3.04 \times 10^{-3}$ and $\Delta ISE(\theta(t)) = 5.7 \times 10^{-4}$, while under 75% data error rate, the difference would be larger, that are $\Delta ISE(z(t)) = 7.29 \times 10^{-3}$ and $\Delta ISE(\theta(t)) = 2.031 \times 10^{-3}$. For case $n_{sc}^{i} = n_{ca}^{i} = 15$, under 25% data error rate, the *ISE* different performance of $z(t)$ and $\theta(t)$ are $\Delta ISE(z(t)) = 7.29 \times 10^{-3}$ and $\Delta ISE(\theta(t)) = 1.399 \times 10^{-3}$. The *ISE* performance difference under 75% data error rate cannot be measured since the system response under the native CAN error handling is unstable.

8. Conclusion

This article discussed a newly error handling mechanism, denoted as PSeD in CAN by introducing a maximum allowable number of error bursts (MAEB) that occur within every sampling time unit. The effectiveness of this method is demonstrated by applying the algorithm to 4th order system of inverted pendulum. From the simulation results, it can be seen that PSeD promotes a better performance of the system as compared to native CAN error handling mechanism for single loop NCS and proven to be superior than native error handling of CAN. For future works, the analysis of this new error handling technique under multi-frame control data, multi-loop of NCS and the error occurrences that governed by probability distribution shall be investigated.

Acknowledgements

The first author would like to express his gratitude to Universiti Teknologi Malaysia (UTM) under UTM-GUP Grant No. Q.J130000.7108.03J37, Universiti Teknikal Malaysia Melaka (UTeM), Ministry of Higher Education (MOHE) Malaysia and Mälardalen University, Västerås, Sweden for their support.

References

[1] P. Antsaklis, J. Baillieul, Special issue on technology of networked control systems, Proceedings of the IEEE 95 (2007) 5–8.
[2] S.H. Hong, I.H. Choi, Experimental allocation of bandwidth allocation scheme for foundation fieldbus, IEEE Transaction on Instrumentation and Measurement 52 (6) (2003) 1787–1791.
[3] L.J. Xu, C.Y. Dong, Q. Wang, The fuzzy variable structure control of spacecraft attitude networked control systems, Yuhang Xuebao/Journal of Astronautics 29 (2) (2008) 590–595.
[4] F.E.W. Frew, C. Dixon, J. Elston, B. Argrow, T.X. Brown, Networked communication, command and control of unmanned aircraft system, Journal of Aerospace Computing, Information and Communication 5 (4) (2008) 84–107.
[5] W. Zheng, G. Han, Design of network control system for car lights based on CAN bus, in: Proceedings of the 2nd International Conference on Electronic and Mechanical Engineering and Information Technology, 2012, pp. 180–183.
[6] J. Greifeneder, G. Frey, Optimizing quality of control in networked automation systems using probabilistic models, in: IEEE Symposium on Emerging Technologies and Factory Automation, 2006, 372–379.
[7] R.M. Murray, K.J. Åström, S.P. Boyd, R.W. Brockett, G. Stein, Future direction in control in an information rich world, IEEE Control System Magazine 23 (2) (2003) 22–33.
[8] R.A. Gupta, M.Y. Chow, Networked control system: overview and research trends, IEEE Transactions on Industrial Electronics 57 (2010) 2527–2535.
[9] Y. Tipsuwan, M.Y. Chow, Control methodologies in networked control systems, Control Engineering Practice 11 (2003) 1099–1111.
[10] J.P. Hespanha, P. Naghshtabrizi, Y. Xu, A survey of recent results in networked control systems, Proceedings of the IEEE 95 (2007) 138–172.
[11] R.W. Mitchell, Profibus: a pocket guide, The Instrument, System and Automation Society (2003).
[12] G. Liang, H. Wang, W. Li, D. Li, Communication performance analysis and comparison of two patterns for data exchange between nodes in WorldFIP fieldbus network, ISA Transactions 49 (2010) 567–576.
[13] J. Zhang, ControlNet control system network design and optimization, Advanced Materials Research 586 (2012) 399–403.
[14] G. Li, C. Xiao, Z. Wu, Development and application control network based on DeviceNet, in: International Conference on Information Science and Technology, 2011, 516–519.
[15] L. Urli, S. Murgia, Use of Ethernet communications for real-time control systems in the metals industry, in: IEEE International Conference on Automation Science and Engineering, 2011, 6–11.
[16] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto, C. Zunino, Real-time Ethernet networks for motion control, Computer Standards and Interfaces 33 (5) (2011) 465–476.
[17] I. Broster, A. Burns, Timing analysis of real-time communication under electromagnetic interference, Real-Time System 30 (1–2) (2005) 55–81.
[18] M.D. Natale, H. Zeng, P. Giusto, A. Ghosal, Understanding and Using the Controller Area Network Communication Protocol, Springer, New York, 2012.
[19] N. Navet, Controller area network: CANs use within automobile, IEEE Potentials 17 (4) (1998) 12–14.
[20] G. Bruno, N. Navet, Fault confinement mechanisms on CAN: analysis and improvements, IEEE Transactions on Vehicular Technology 54 (3) (2005) 1103–1113.
[21] H. Huangshui, Q. Guihe, Online fault diagnosis for controller area network, in: 4th International Conference on Intelligent Computation Technology and Automation, vol. 1, 2011, 452–455.
[22] G. Buja, J.R. Pimentel, A. Zuccollo, Overcoming Babbling-Idiot in CAN networks: a simple and effective bus guardian solution for the FlexCAN architecture, IEEE Transactions on Industrial Informatics 3 (3) (2007) 225–233.
[23] H. Aysan, A. Thekkilakattil, R. Dobrin, S. Punnekkat, Efficient fault tolerant scheduling on controller area network (CAN), in: 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2010), 2010.
[24] K.C.S. Emani, Application of hybrid ARQ to controller area network, University of Missouri-Rolla, 2007 (Master's Thesis).
[25] M.P. Kumar, P. Prabhat, An efficient forward error correction scheme for wireless sensor network, Procedia Technology 4 (2012) 737–742.
[26] P.D. Amer, C. Chassot, T.J. Connolly, M. Diaz, P. Conrad, Partial-order transport service for multimedia and other applications, IEEE/ACM Transactions on Networking 2 (5) (1994) 440–455.
[27] P. Ramanathan, Overload management in real-time control application using (m,k)-firm guarantee, IEEE Transaction on Parallel and Distributed Systems 10 (6) (1999).
[28] F. Flavia, Impact of a (m,k)-firm drata dropout policy on the quality of control, IEEE International Workshop on Factory Communication Systems (2006) 353–359.
[29] J. Unruh, H.J. Mathony, K.H. Kaiser, Error detection analysis of automotive communication protocols, SAE (Society of Automotive Engineers) Transactions 99 (1990) 976–985.
[30] ISO-11898, Road Vehicle—Interchange of digital information—Controller Area Network (CAN) for high speed communication, 1993.
[31] C. Peng, D. Yue, Maximum allowable equivalent delay bound of networked control systems, in: 6th World Congress on Intelligent Control and Automation (WCICA 2006), 2006, 4547–4550.
[32] Q. Liang, M.D. Lemmon, Robust performance of soft real time networked control system with data dropout, Proceedings of the IEEE Conference on Decision and Control 2 (2002) 1225–1230.
[33] M.G. Rodd, K. Dimyati, L. Motus, The design and analysis of low-cost real-time fieldbus systems, Control Engineering Practice 6 (1998) 83–91.
[34] M. Gergeleit, H. Streich, Implementing a distributed high-resolution real-time clock using the CAN-bus, in: International CAN Conference, vol. 94, 1994.
[35] N.S. Nise, Control Systems Engineering, 6th ed., Wiley, New Jersey, 2010.
[36] R.C. Dorf, Modern Control System, 12th ed., Addision-Wesley, New York, 2010.

[37] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K.E. Årzèn, How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime, IEEE Control Systems Magazine 23 (3) (2003) 16–30.
[38] A. Cervin, D. Henriksson, M. Ohlin, TrueTime 2.0 beta 5—Reference manual, Department of Automatic Control, Lund University, Sweden, 2010.

M. B. Nor Shah received the M. Eng degree in 2011 in Mechatronic and Automatic Control from Universiti Teknologi Malaysia, where he is currently working toward the Ph.D. degree in Electrical Engineering (Control). He is also fellow of Universiti Teknikal Malaysia, Melaka. His current research interests are networked control system, real-time control system, robust control, controller area network (CAN) and fault in network.

A. R. Husain received the B.Sc. degree in electrical and computer engineering from The Ohio State University, Columbus, Ohio, U.S.A., in 1997, M.Sc. degree in Mechatronics from University of Newcastle Upon Tyne, U.K., in 2003, and Ph.D. in Electrical Engineering (Control) from Universiti Teknologi Malaysia (UTM) in 2009. Before joining UTM, he worked as an engineer in semiconductor industry for several years specializing in precision molding and IC trimming process. He has taught courses in introduction to electrical engineering, microcontroller based system, modeling and control, and real-time control system. His research interests include control of dynamic and network control system, real-time control system, and system with delay.

S. Punnekkat received the Master of Statistics degree and the Master of Technology in Computer Science degree with honors from the Indian Statistical Institute, New Delhi, India, in 1982 and 1984, respectively and the Doctor of Philosophy degree in computer science from the University of York, U.K., in 1997. He is currently a Professor in dependable software engineering at Mälardalen University, Västerås, Sweden and the leader of the Dependable Software Engineering research group. He has more than 15 years industrial experience as a scientist at the Indian Space research Organization, and was the Head of the Software test and reliability engineering. He was recipient of the prestigious Commonwealth Scholarship and was awarded Doctor of Philosophy in Computer Science by the University of York, UK in 1997 for his research on schedulability analysis of fault-tolerant systems. He is the program director of the Master Programs in Software Engineering at MDH. His research interests include multiple aspects of Real-time Systems, Dependability, and Software Engineering.

R. Dobrin is a Senior Lecturer at the Department of Computer Science and Engineering at Mälardalen University, Västerås, Sweden and the Chair of the Software Engineering Division. He has a background in scheduling of dependable real-time systems and is currently involved in both research and education-oriented projects. Currently, his research is focused in the area of dependable embedded real-time systems build using component based development.