NETWORK TOPOLOGIES FOR LONG ARMATURE LINEAR MOTORS

by

KUBRA KARAYAGIZ

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of

the requirements for the degree of

Master of Science

Sabanci University

Fall 2013

NETWORK TOPOLOGIES FOR LONG ARMATURE LINEAR MOTORS

APPROVED BY

Assoc. Prof. Dr. Ahmet ONAT (Thesis Supervisor)	………………………………………..
Assoc. Prof. Dr. Ayhan BOZKURT	……………………………………….
Assoc. Prof. Dr. Ozgur ERCETIN	……………………………………….

DATE OF APPROVAL: ……………………………………….

Acknowledgments

First, I would like to express my special appreciation and thanks to my supervisor Ahmet Onat for his guidance, support and good nature. I feel fortunate to have had the opportunity to work with such a kind and considerate supervisor.

Secondly, I would like to thank my friends in Sabanci University for their support and interior friendship.

Next, I would also like to thank my family, especially my mother and my father for all of their sacrifices. Words cannot express how grateful I am to them. And finally, I would like to express my gratitude to my husband, Omer, not only for his encouragement but also for his patience and understanding.

Abstract

Long armature linear motor is a type of distributed control system including a large number of motor drivers implemented in a distributed fashion.

In such a system, data exchange between the motor drivers must be accurate and lossless. However, delays and packet losses can be constant, bounded, or even random, depending on the network protocol adopted. Although it is hard to avoid delay and packet loss problems in communication networks, it is possible to decrease these disadvantages to the minimum level by choosing a suitable communication protocol and a network topology.

In this thesis, CAN protocol that is a reliable real-time communication protocol for control systems is used and two different network topologies entitled as topology A and topology B are introduced for the communication of motor drivers of long armature linear motor.

Topology A is a hierarchical system composed of motor drivers, a main computer that coordinating the overall motion of the linear motor movers (with connected elevator cars), and gateway computers in between. The unit that is composed of one gateway and a fixed number of motor drivers connected to it is called one “motor section”. The structure of this topology requires using gateway computers to communicate with adjacent motor sections while the mover is passing the borders of motor sections. Due to this, an extra delay occurs in the communication of the adjacent motor sections.

Similar to topology A, topology B is a hierarchical system composed of motor drivers, a main computer and gateway computers as well. However, thanks to the structure of this topology, gateway computers are not required to be used to communicate with adjacent motor sections.

The proposed topologies are simulated via TrueTime, a toolbox for simulation of distributed real-time control systems. Since the system consists of many computers, it is not easy and inefficient to use the standard Matlab graphical user interface based mouse operations to create a model. Moreover, while analyzing the topologies, the system parameters must be changed several times to observe their effects in the system performance or for other purposes. In order to easily create and modify that kind of large and complicated models, we use a new Matlab script based method to build Simulink models of large repetitive systems.

The advantages and disadvantages of both topologies are discussed and their performances are evaluated based on comparisons in delay amount. The results indicate that compared to topology A, topology B has a better performance with less delay as expected. Therefore, topology B is suggested for the communication of motor drivers of long armature linear motor.

Özet

Uzun armatürlü lineer motor, büyük sayıda motor sürücü içeren bir çeşit dağıtık kontrol sistemidir.

Bu gibi sistemlerde, motor sürücüler arasındaki bilgi alışverişi doğru ve kayıpsız olmalıdır. Fakat, kullanılan ağ protokolüne göre sabit, sınırlı veya rastgele zaman gecikmeleri ve kayıplar oluşabilmektedir. Haberleşme ağlarında, bu gibi zaman gecikmesi ve paket kaybı problemlerinden tamamen kurtulmak mümkün olmasa da, seçilen uygun bir haberleşme protokolü ve ağ topolojisine göre bu dezavantajları en düşük seviyeye indirmek mümkündür.

Bu çalışmada, uzun armatürlü lineer motorun motor sürücülerinin haberleşmesinde kontrol sistemleri için güvenilir bir gerçek-zamanlı haberleşme protokolü olan CAN protokolü kullanılmış, topoloji A ve topoloji B isimli iki farklı ağ topolojisi önerilmiştir.

Topoloji A, motor sürücüleri, ana bilgisayar ve bu ikisi arasındaki gateway bilgisayarlardan oluşan hiyerarşik bir sistemdir. Bir gateway bilgisayar ve belirli sayıda motor sürücüden oluşan bölüme bir "motor bölümü" denilmektedir. Bu topolojinin yapısı, itici motor bölümlerinin sınırlarını geçerken, komşu motor bölümlerle haberleşmek için gateway bilgisayarları kullanmayı gerektirir. Buna bağlı olarak, komşu motor bölümler ile haberleşmelerde daha fazla zaman gecikmesi meydana gelmektedir..

Topoloji A'ya benzer olarak, topoloji B de motor sürücüler, ana bilgisayar ve gateway bilgisayarlardan oluşan hiyerarşik bir sistemdir. Fakat, bu topolojinin yapısı sayesinde, komşu motor bölümlerle haberleşmelerde gateway bilgisayarların kullanılması gerekmemektedir.

Amaçlanan topolojiler, gerçek-zamanlı dağıtık kontrol sistemlerinin simülasyonu için geliştirilmiş olan TrueTime yazılımı ile simüle edilmiştir. Sistem pek çok bilgisayardan oluştuğu için, simülasyon modelini oluştururken, standart grafiksel kullanıcı arayüzüne dayalı Matlab işlemlerini kullanmak kolay değildir ve verimsiz bir metottur. Ayrıca, topolojileri analiz ederken, sistem parametrelerinin performansa etkisini gözlemlemek için veya başka amaçlarla pek çok defa değiştirilmeleri gerekmektedir. Bu gibi büyük ve karmaşık modelleri kolaylıkla oluşturmak ve değiştirmek için, Matlab kodlarına dayalı bir metot kullanılmıştır.

Her iki topolojinin avantaj ve dezavantajları tartışılmış ve zaman gecikme miktarları kıyaslanarak da performansları değerlendirilmiştir. Sonuçlar, beklenen gibi topoloji B'nin daha iyi sonuçlar verdiğini göstermiştir. Bu yüzden, uzun armatürlü lineer motorun haberleşmesi için topoloji B ağ yapısı önerilmektedir

Table of Contents

List of Figures

Chapter 1

Introduction and Background

Thanks to the improvements in building technology, high-rise buildings can be constructed successfully. However, traditional cable driven elevators are not desirable in such buildings . The most important elevator problems encountered in such high-rise buildings are that the weight of the traction cable limits the payload and its elasticity degrades control performance . Since only one elevator car can use the hoist way, transportation from the ground to the top floors takes a long time causing a high passenger waiting time and its elasticity degrades control performance [1], [2], [3]. A solution to these problems is using multiple cars in the same hoist-way. However, traditional cable driven elevators have some limitations on the number of cars placed in the same hoist way; because of the cables, it is not mechanically possible to include several elevator cars in the same hoist way. However, for obtaining significant improvements over single–car systems, we should use more than two cars in the same hoist way spanning several hundred meters; the height of the building, which can be realized using elevator cars being directly driven by linear motor. [2], [1], [3], [7], [8].

In multi–car elevator systems, since the travelling cable is undesirable the elevator car must be free of any connections, mechanical or electrical. This causes some interesting design requirements on safety devices, position measurement, supplying power, communication of call buttons which must all be implemented externally, from the stator side. [10], [11], [12]. Because of these limitations, the design must be a long armature type where the stator contains the coils and the mover contains permanent magnets.

Since the propulsion method is the core component of new multi-car elevator (MCE) systems, it needs to be designed to manage not only lifting the elevator car, but also several other requirements. Therefore, selection and design of the linear motor has a big importance. In addition to the major properties such as power consumption, torque and size additional requirements for mechanical and control purposes, must be included as design criteria.

In order to satisfy all requirements, the linear motor that will be used for MCE systems must have these properties: high thrust force, low force ripple, brake operation, spanning whole hoist way, independent control of sections, unlimited length, modularity, position sensing, power and signal transfer. Our group member Ender Kazan discussed all these properties in detail and proposed a linear motor meeting all the requirements. In this research, we will only discuss about the properties related to our studies among all the properties mentioned above. These are:

- Spanning whole hoist way – Since traction cable is absent, it is not possible to transfer lifting force through the hoist way. Instead of ropes, the linear motor mover will transfer that force directly. So that, we need a motor spanning the whole hoist way.

- Independent Control of Sections – MCE systems require independent control. Therefore, at least one mover for each elevator car is needed. However, it is not feasible to build different actuators for each car. In order to control each car independently, the motor can be divided into different sections electrically and each section is assigned to different cars. Therefore, each car can be controlled independently from the section assigned to it.

- Unlimited Length - The motor should not limit the height of the building. Therefore, this criterion also must be taken into consideration while the motor is designing.

- Modularity – Since the linear motor for MCE systems is constructed for very high buildings, it is not possible to construct it at once. Therefore, the motor must be designed as parts and constructed part by part to be assembled into each other within the hoist way. Moreover, if a replacement of a part is needed, it is possible to replace only the related module can be replaced instead of whole motor.

- Power and Signal Transfer – Cables for power or signal transfer to elevator car must be removed. In this case, the motor itself should be used for power and signal transfer to elevator car.

- Position Sensing – In order to control and drive the motor accurately, feedback from the system is needed. However, the position measurement method should not be based on any cable travelling with the elevator car. Therefore, it is not suitable to use conventional position encoders for position sensing. The motor itself should be able to measure the position of the car.

In order to design a linear motor with a good performance, all these properties must be examined in detail. Some of these properties have been studied before by our group members. A significant research on position sensing methods has been done by one of our group member Cagri Gurbuz. He investigated position sensing methods and introduced a method called as the linear-motor active position sensing method (LIMAP) having the best performance for the linear motors used in MCE systems up to now. This research discussed on the following subjects. It is desired to drive the motor without any electrical connections between the mover and stator. Therefore, there are some limitations on safety devices, supplying power and position measurement [10], [11], [12]. Especially, limitations on position measurement complicate to drive the RLMs. In this case, sensorless motor drive method can be a solution to this problem. Sensorless motor drive is usually realized by measuring the electrical variables of the motor [13], [14]. However, these methods do not work in low speeds because of the magnitude of back-EMF will be very small to provide enough voltage value compared to the noise. Therefore, incremental encoders are generally used for measuring the position. However, in this method the active part of the encoder is attached to the mover and cables of the encoder are travelling with the mover. This is an undesirable way in MCE systems. Therefore, a new method, that is sensitive to initial

position called linear drive with passive vehicle, was proposed. Although the previous studies [15], [16], [17], [18] on position sensing with the use of passive lightweight vehicles attached to the mover are accurate enough for linear motors, these methods contradicts with some constraints of the linear motor designed for MCE. Therefore, our group member Cagri Gurbuz proposed the idea of the passive position sensing for ropeless linear motors, which utilized its own stator. In this method, it is taken advantages of the segmented construction of the linear motor.
Segmented construction provides separated segments to do different works. During normal drive operation, while active segments are used for motor driving, the passive segments can be used for other operations, such as braking, signal transfer or position sensing. The position measurement method called Linear Motor Active Position Measurement Method (LIMAP), introduced by Cagri Gurbuz, aims to use the passive segments of the stator, for position sensing of the mover. The sensing is based on the principle of measuring the variation of the mutual inductance between the coils of the stator caused by a magnetic shunt which is fixed at a predetermined distance from the mover [19]. By combining the proposed position sensing method with sensorless drive method, it is aimed to obtain a linear motor completely ropeless and working independently from the zero velocity.

The working principle of LIMAP system is explained as follows. The motor coils are placed on the stator. The stator is electrically divided into segments and each segment has its own motor driver and controller. The magnetic shunt used for position measurement and the mover are mounted on both sides of the stator and connected to each other mechanically. The position of the magnetic shunt is calculated by the drivers close to the shunt during the movement. Since the distance between the magnetic shunt and the mover is known, it is possible to determine the position of the mover from measured position of the magnetic shunt. Finally, the calculated position value is sent to the other drivers which are used for motor driving in order to continue the movement. The method explained above requires a communication between the motor drivers for transfering the position information. Moreover, it is also necessary for commanding target position, velocity etc. of the mover and other purposes.

The control method that is applied to the linear motor must be designed as enabling the movers to move independently. Also, it must be independent of the motor length. The simplest method to achieve that is to implement a centralized control scheme where each module is directly connected to a central controller. However, this method can be applied to a limited number of movers in the system. Therefore, control of the movers should be shared between local controllers instead of one main controller. In this approach, the motor modules are driven separately with a certain coordination. While a mover is passing a certain part of the motor, relevant segments must be allocated and deallocated for a predetermined duration to avoid collisions. While the motor is driven, the synchronization of the electrical phases of several modules can cause a problem in terms of timing requirements. Especially the adjacent drivers have more time restrictions since the duty of motor driving is shared between the adjacent drivers. If a time delay occurs between these adjacent drivers, the electrical signal can not transmit well and this causes oscillation and driving of the mover improperly. In order to solve this problem, a real-time computer network is used so that travelling of the movers can be coordinated by distributed control [20].

Distributed control systems include a set of control systems implemented in a distributed fashion using an appropriate communication network and protocol. A networked control system seen in Figure 1 is a control system whose functions are shared between separate modules each implemented in a distributed fashion on separate computer nodes connected by a real-time computer network. Each node in the computer network must perform computations in a bounded amount of time to meet sampling constraints of the control system. The computing system and the communication network have several features designed to satisfy the requirements and constraints of the control system. A control system is characterized by one or more feedback control loops, and associated control algorithms, sensors, and actuators. The various controllers, sensors, and actuators can take place in different nodes of the communication network and communicate among one another using the network services. Thus in a distributed control system the applications involve controllers, sensing functions, actuation functions, and communication functions in the context of control loops.

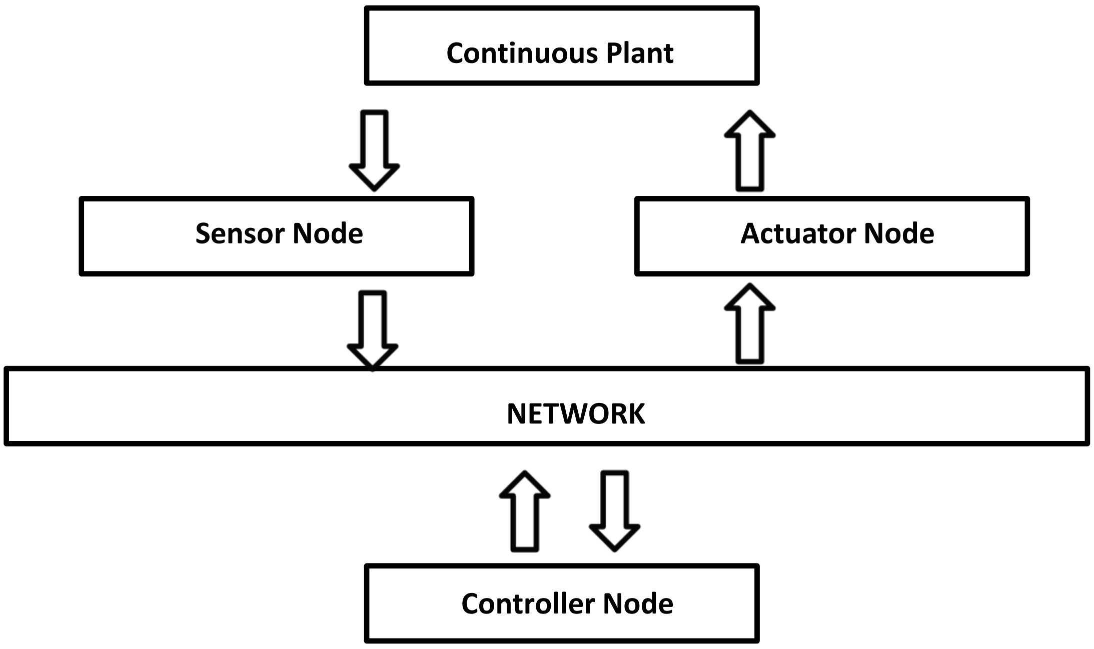


Figure 1. A networked control system

In a communication network managing traffic, timely execution of tasks and delivery of messages within their deadlines are of prime importance. These time constrained messages are the basis for applications that operate in a real-time environment.

Besides timely delivery of the messages, another important requirement in such a system is data exchange between the physical process and the controllers to be accurate and lossless. However, transmission delays and packet losses can be constant, bounded, or even random, depending on the network protocol adopted. Although they are hard to avoid in communication networks, it is possible to decrease these disadvantages to the minimum level by choosing a suitable communication protocol and a network topology.

In this research, our aim is to find a network topology suitable for the long armature linear motor as a distributed real time system and evaluate their performance. Two different topologies are introduced and simulated in TrueTime that is a Matlab/Simulink-based simulator for networked and embedded control systems. The advantages and disadvantages of both topologies are discussed and their performances are evaluated according to the simulation results. Moreover, the suitable communication protocols are introduced and applied in TrueTime system models.

The work done in this thesis is laid out as follows. Firstly, in Chapter 1, some information about the background of the project is presented and the motivation of our study is declared. Later, in Chapter 2, communication protocols suitable for real time communication are explained. In Chapter 3, proposed topologies and protocols are introduced and their advantages and disadvantages are discussed. In Chapter 4, after background information on TrueTime is provided, the steps for creating a script-based TrueTime simulation models are explained. In Chapter 5, the simulation results of the proposed topologies are presented and performances are compared. Chapter 6 summarizes and concludes the thesis and indicates possible future works.

Chapter 2

Communication Protocols Suitable for Real Time Communication

2.1. Contention-Based Protocol

In contention based protocols, there is no centralized control and when a node wants to send data, it competes for gaining control of the medium. The main advantage of contention based protocols is their simplicity. They can be easily implemented in each node. The techniques work efficiently under light load, but performance falls under heavy load [21].

2.1.1. CSMA

Carrier sense multiple access (CSMA) is a probabilistic media access control (MAC) [22]. In this protocol, before transmitting data, a node first listens to the medium to check whether another transmission is in progress or not. The node starts sending only when the channel is free, that is there is no carrier. That is why the scheme is also known as listen-before talk. [21]

2.1.1.1. CSMA/CD

CSMA/CD is used to improve CSMA performance by terminating transmission as soon as a collision is detected. In this protocol, nodes transmit only when they detect that the channel is idle. If more than one node starts to transmit at about the same time, a collision of packets occur. In this case, the transmissions have to be aborted and then retried.

2.1.1.2. Virtual-Time Carrier-Sensed Multiple Access (VTCSMA)

VTCSMA is designed to avoid collision generated by nodes transmitting signals simultaneously, used mostly in Hard Real-Time systems. The VTCSMA algorithm uses two different clocks; a real-time clock and a virtual-time clock, at each node. If the channel is idle, VT clock runs. If it is behind the real-time clock, it runs at a faster rate. When a node generates a message, its time is stamped with the (real) time of its generation. In order to transmit a message, the virtual clock reading should reach the time stamp of a message.

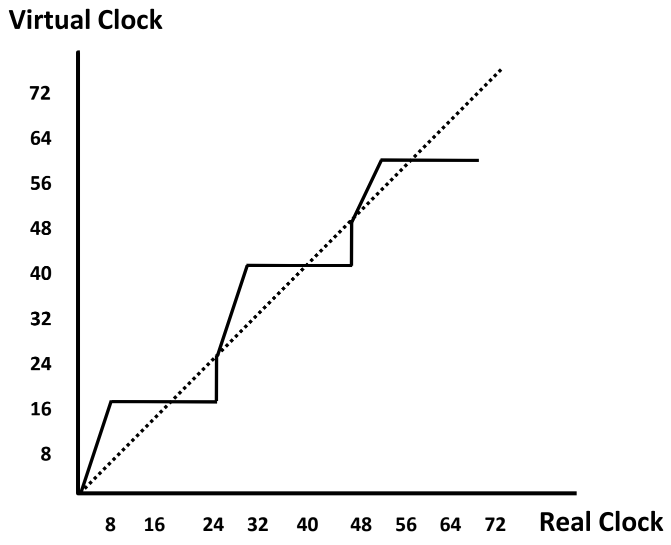


Figure 2. VTCSMA Protocol

In order to satisfy soft real-time constraints, latest time can be used to send as the time stamp. Whenever the bus becomes idle virtual clock is reset to real clock and then runs at a faster rate. This results in a significantly improved probability of missing message deadlines and results in a network-wide minimum-laxity-first policy.

2.1.2. Window Protocol

Similar to the VTCSMA protocol, the window protocol is based on collision sensing and it can not be guaranteed that messages will be transmitted in time to meet their deadlines. Therefore, this protocol is only suitable for soft real-time systems.

The system consists of a set of nodes connected on a bus like VTCSMA. In order to receive any messages that may be addressed to a node, each node continuously monitors the bus. All activity on the bus is equally visible to all the nodes. Therefore, it can be assumed that each node knows when multiple simultaneous transmissions collide or when a transmission has succeeded, even if the transmissions in question does not originate from or not destinated for that node. Thus, for synchronizing the actions of the nodes, events on the bus become a mechanism.

The protocol owes its name to the window maintained at each node. The window is a time interval, and the windows of all the nodes are identical. The packet is eligible to be transmitted when the latest-time-to-transmit (LTTT) of a packet falls within the window and the channel is idle. If a node has more than one packet that are eligible for transmission, one of them is chosen based on some criterion (e.g. LTTT). [23]

2.2. Token-Based Protocols

A node must hold the token to transmit its packets on the network. A token is a grant of permission to a node to transmit. When the node with the token completes its transmission, it surrenders the token to another node.

The ratio of the end-to-end delay time to the time taken in putting out a packet on the ring is large in optical networks. Therefore, token-based protocols are more suitable for optical networks than collision-sensing.

2.2.1. IEEE 802.5 Token-Ring Protocol

Token Ring is a LAN protocol that is defined in the IEEE 802.5. All stations are connected in a ring network and each station can receive transmissions from only its instant neighbor. A token that circulates around the ring gives the permission to transmit.

IEEE 802.5 Token Ring protocol is originated from the IBM Token Ring LAN technologies. Both of them are based on the Token Passing technologies. They generally compatible with each other but differ in minor ways.

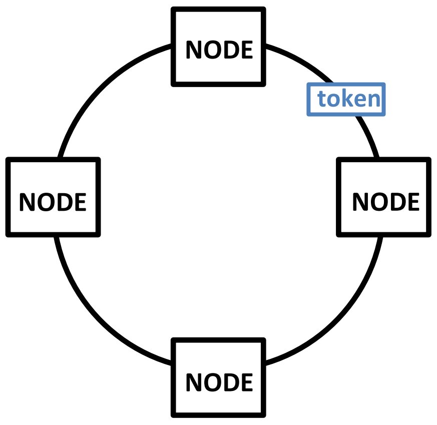


Figure 3. Token Ring Protocol

Token-passing networks move a small frame, called a token, around the network. Possession of the token grants the right to transmit. If a node receiving the token does not have any information to send, it seizes the token, alters 1 bit of the token (which turns the token into a start-of-frame sequence), attaches the information of wanting to transmit, and sends this information to the next station on the ring. While the information frame is circling the ring, no token is on the network. It means that other stations wanting to transmit must wait. So that, collisions do not occur in Token Ring networks.

The information frame circulates the ring until it reaches the intended destination station. Then the station copies the information for next processing. The information frame maintains circling the ring and when it reaches the sending station, it is finally removed. The sending station can check the returning frame to see if the frame was seen and copied by the destination.

In contrast with Ethernet CSMA/CD networks, token-passing networks are deterministic. It means that it is possible to calculate the maximum time that will pass before any end station will be capable of transmitting. This feature and several reliability features make Token Ring networks ideal for applications requiring delay to be predictable and robust network operation [24].

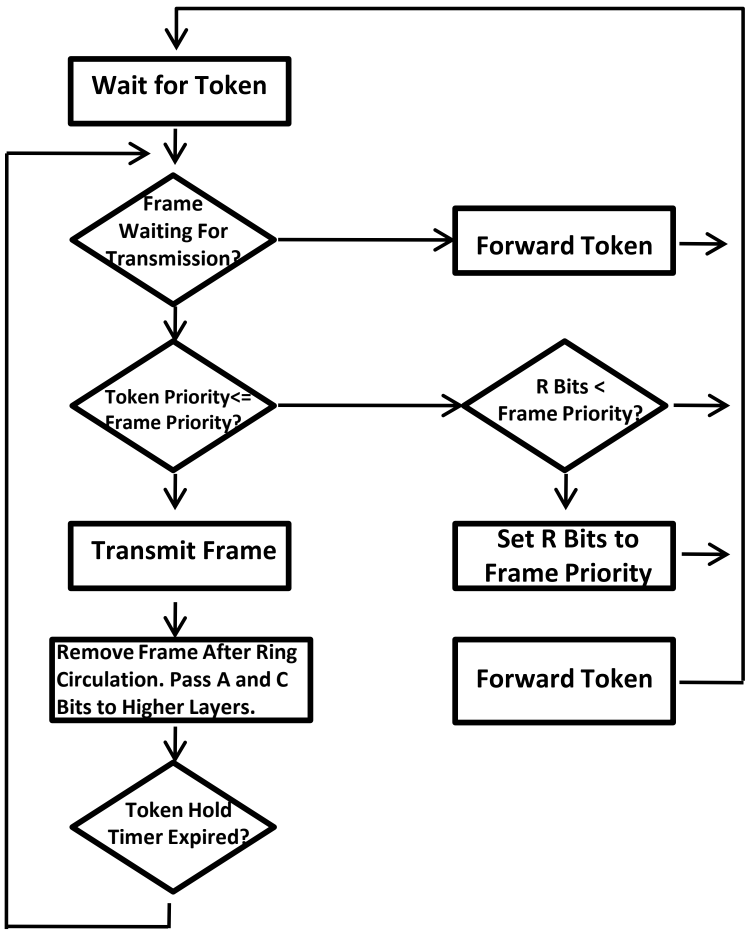

Wait for Token
Frame Waiting For Transmission?
Forward Token
Token Priority<= Frame Priority?
R Bits < Frame Priority?
Transmit Frame
Set R Bits to Frame Priority
Remove Frame After Ring Circulation. Pass A and C Bits to Higher Layers.
Forward Token
Token Hold Timer Expired?

Figure 4. Token Ring Send Algorithm

2.2.1.1. Timed-Token Protocol

The timed-token protocol provides each node to be guaranteed timely access to the network. It distinguishes between two basic classes of traffic, synchronous and asynchronous.

Synchronous traffic is the real-time traffic which guarantees that each node can send out up to a certain amount of synchronous traffic every T time units. Asynchronous traffic is nonreal-time traffic. It takes up any bandwidth left unused by the synchronous traffic. It can consist of multiple priority classes [23].

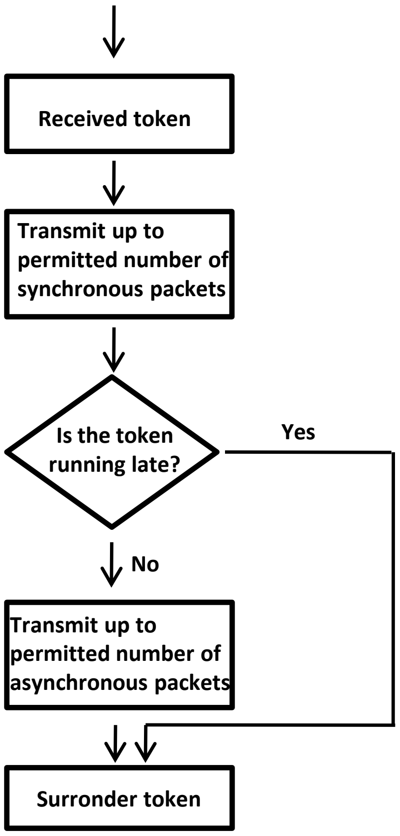

Received token
Transmit up to permitted number of synchronous packets
Is the token running late?
Yes
No
Transmit up to permitted number of asynchronous packets
Surronder token

Figure 5. Flowchart of the timed-token protocol after the second cycle

2.3. The Polled Bus Protocol

There is a bus network with a bus-busy line. When a processor broadcasts on the bus, it also makes this line high. After finishing broadcasting, this line is reset. This can be done very easily if the line executes a *wired-OR operation.* If two signals a and b are put out on the line simultaneously, the resultant signal is a OR b.

The polled bus protocol assumes all the processors to be tightly synchronized. The time axis is divided into slots. Each slot duration is equal to the end-to-end propagation time of the bus.

If a processor wants something to transmit on the bus, it checks the bus-busy line to see if it is busy. If the bus-busy line is busy, the processor waits until the transmission ceases. If it is not busy, it monitors the bus for one slot. During that slot, if no other processor makes a request, the processor starts to transmit a poll number on the bus. This poll number is directly proportional to the priority of the message.

The poll number is transmitted slowly. The transmission rate is one bit per slot. After the processor transmits its bit, it monitors the bus to see whether the signal on the bus is the same as its own output. If it is not the same, we can understand that there is higher-priority processor asking for access, and this processor drops out of contention and stops transmitting its poll number. If the bus signal is the same, the processor proceeds during the next slot to broadcast the next bit of the poll number. This process continues until it sends out its entire poll number successfully or until it has to drop out of contention [23].

2.4. CAN Bus Protocol

CAN (Controller Area Network) protocol was developed to be used for conveying information between the subassemblies in an automobile, truck or vessel in automotive industry. However, since it has a good performance in other process control systems, it is used in many other industrial applications.

CAN is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs). Each node can send and receive messages, but not simultaneously. A message is composed of an ID (identifier), which represents the priority of the message, and data up to eight bytes. In the improved CAN (CAN FD), the length of the data section can be extended up to 64 bytes per frame. It is transmitted serially onto the bus. This signal pattern is encoded in non-return-to-zero (NRZ) and is sensed by all nodes.

The devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are not connected directly to the bus, but through a host processor and a CAN controller.

If the bus is idle which is represented by recessive level (TTL=5V), any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits, i.e., zeroes) will overwrite less dominant IDs of other nodes, so that finally only the dominant message remains and is received by all nodes. This mechanism is referred to as priority based bus arbitration. Messages with numerically smaller values of IDs have higher priority and are transmitted first.

Each node requires a host processor, CAN controller and a transceiver. Their duties re explained below:

- ❖ Host processor: decides what received messages mean and which messages it wants to transmit itself. Sensors, actuators and control devices can be connected to the host processor.
- ❖ CAN controller:
  - ➢ *Receiving*: the CAN controller stores received bits serially from the bus until a whole message is available. Thereafter, it can be fetched by the host processor usually after the CAN controller has triggered an interrupt.
  - ➢ *Sending*: the host processor stores its transmit messages to a CAN controller, which transmits the bits serially onto the bus.
- ❖ Transceiver:

- *Receiving*: it adapts signal levels from the bus to levels that the CAN controller expects and has protective circuitry that protects the CAN controller.
- *Transmitting*: it converts the transmit-bit signal received from the CAN controller into a signal that is sent onto the bus.

The CAN protocol is optimized for short messages. It can carry up to 8 bytes of data in each frame. Bit rates can change according to the network length. It can increase up to 1 Mbit/s at network lengths below 40 m. Decreasing the bit rate allows longer network distances (e.g., 500 m at 125 kbit/s). The improved CAN (CAN FD) extends the speed of the data section by a factor of up to 8 of the arbitration bit rate.

2.4.1. Data Transmission

In CAN protocol, data transmission is based on priority. A message with highest priority will succeed, and the node transmitting the lower priority message will sense this and wait. In order to decide on the message with higher priority, CAN transmitting data is checked. It includes binary numbers 1 or 0. If it is logical 0 then it is dominant bit, if it is logical 1 then it is recessive bit. If one node transmits a *dominant* bit and another node transmits a *recessive* bit then the dominant bit "wins". If any node sets a voltage difference, all nodes will see it. Thus there is no delay to the higher priority.

It uses a CSMA/CD+AMP (Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority) access protocol. If more than one node start transmitting at the same time, there is a priority based arbitration scheme to decide which one will be permitted to continue transmitting. The CAN solution to this is prioritized arbitration, making CAN very suitable for real time prioritized communications systems.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting. Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID. When their ID is a larger number (lower priority) they will send 1 and see 0, so they cancel transmitting. At the end of ID transmission, all nodes with lower priorities back off, and the highest priority message gets the chance to be sent.

Chapter 3

Proposed Topologies and Protocols

Real-time systems can support the execution of applications with time constraints. There are two classifications of real-time systems based on their properties: soft and hard real-time systems. In hard real-time systems, like our linear motor system, delay is accepted only up to a certain level. If a deadline is not met, the controlled device may suffer a failure. The proposed topologies aim to decrease the delay to the minimum levels to satisfy hard real time requirements. Two different topologies are introduced and their advantages and disadvantages are discussed in this section.

3.1. Topology A

The structure of the topology can be seen in Figure 6. It is a hierarchical system composed of motor drivers, a main computer that coordinating the overall motion of the linear motor movers (with connected elevator cars), and gateway computers in between. The "Main computer" labeled as 'M' is connected to "gateway computers" labeled as 'G' and gateway computers are connected to "motor drivers" labeled as 'MD'. The unit that is composed of one gateway and a fixed number of motor drivers connected to it is called one "motor section" shown in a grey colored square. There are several different types of networks in the system. We can classify them into three groups: one is between main computer and gateway computers labeled as "network1", another one is between a gateway computer and motor drivers labeled as "network2", and the last one is between adjacent gateway computers labeled as "network3" in the Figure 6. As seen in the Figure 6, main computer is only connected to network1, motor drivers are only connected to network2 motor of their section. Unlike the main computer and motor drivers, gateway computers are connected to at least three types of networks, network1, network2 and one or two different network3. While inner gateways are connected to two different network3, first and last gateways are connected to only one network3 based on their positions in the system.

The tasks and real-time requirements of the three types of computers are divided in the following way:

- Main computer's job is to set the position reference and generate the motion profile of the mover and communicate this to motor drivers through movers, when a passenger pushes the button to call the elevator. In order the mover to arrive to the correct floor, the main computer sends the necessary messages to the related motor drivers.

- Gateway computers are responsible for motion control, calculation of current reference based on position measurement sent from motor drivers. Besides data transfer from main computer to motor drivers, gateway computer has an important role that is receiving the position information from its own motor drivers and sending to motor drivers of adjacent group or vice versa. This data transfer is required to communicate with adjacent motor sections while the mover is passing the borders of the groups.

- Motor drivers have two different tasks during position measurement and during driving.
    - During position measurement: generation of sinusoidal coil excitation signal, measurement of induced coil voltages.
    - During driving: current control of motor and similar low level control and measurement duties. Motor drivers receive the position information from analog input and send to the gateway computers for necessary calculations. In order to drive the motor, adjacent drivers have to communicate. While the mover is passing through one motor section, motor drivers of that section can easily communicate through their own group network. However, while crossing from one motor group to the other, they communicate via gateway computers as explained above.

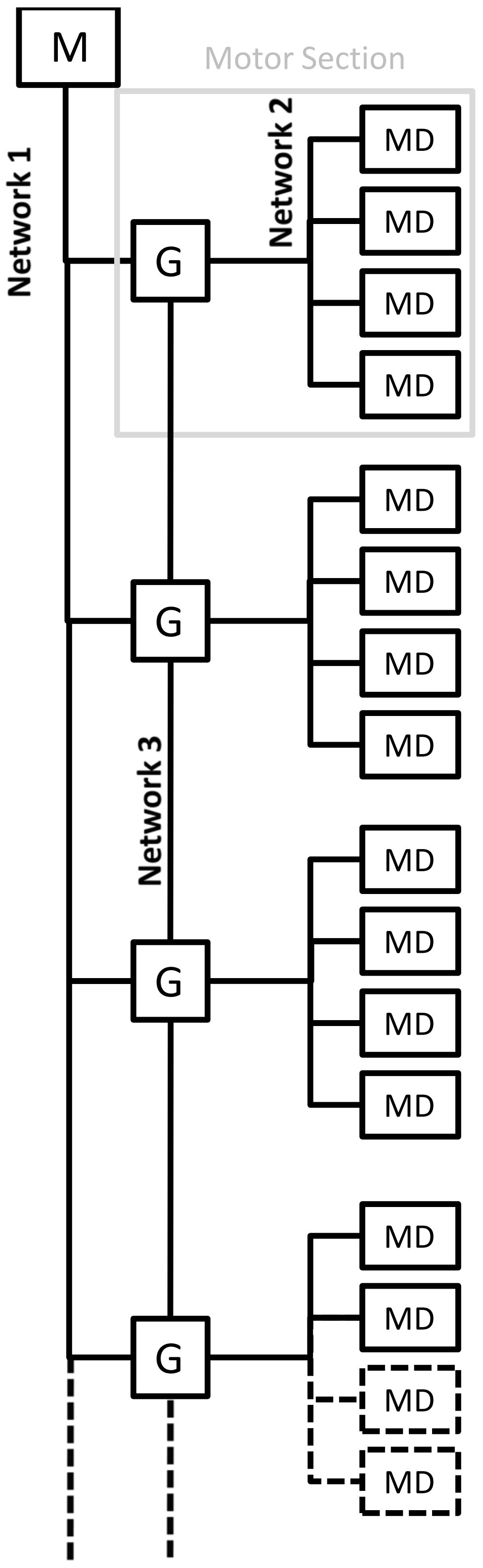


Figure 6. Topology A

3.1.1. Suitable Communication Networks for Topology A

Since the linear motor is designed for high rise buildings, the communication protocols must be chosen to meet its requirements. Considering that requirement as well, the communication protocol for network 1 is chosen as Ethernet, with a high communication and addressing capacity that is sufficient for a large number of gateway nodes, and no hard real-time capability requirements. Network 2 and network 3 can be simpler but with low latency and high reliability. Therefore, CAN protocol is suitable for these networks.

3.1.2. Advantages of the Method

Motor drivers can be simple computers. They can be used either as position sensors or inverters with current regulation. If a motor driver is used for position measurement, it will only produce sinusoidal excitation voltage for the LVDT on one phase and send the measured position value to neighboring drivers. However, while the mover is passing the section borders gateway computers must be used to relay the position information, since the position measurement and drive tasks are shared between nodes of adjacent sections.

3.1.3. Disadvantages of the Method

Due to the structure of topology A, extra delays occur in the last motor drivers of each motor section. These extra delays in the last motor drivers may cause torque fluctuations and vibrations while the mover is passing the motor sections. Moreover, since gateway computers are used for the communication of the adjacent motor drivers, if one of these gateway computers breaks down, problems in driving and controlling the motor will be encountered.

3.2. Topology B

The structure of the topology can be seen in Figure 7. “Main computer” labeled as ‘M’ is connected to “gateway computers” labeled as ‘G’ and gateway computers are connected to “motor drivers ” labeled as ‘MD’. The unit that is composed of a fixed number of motor drivers is called one “motor section” shown in a grey colored square. As distinct from topology A, a gateway computer to relay the information between motor sections is not required. Here, this task is carried out by a particular motor driver. So, in this topology the motor drivers are more complex than the motor drivers of topology A. In topology B, motor sections communicate directly through the common networks shared with the adjacent motor sections without any need for such gateway computers. In this topology, gateway computers are used for relaying messages from main computer to motor drivers and for necessary motor control calculations.

There are two types of networks in the system: one is between main computer and gateway computers labeled as “network 1”, another one is between motor drivers of a motor section and gateway computers labeled as “network 2” seen in Figure 7. Main computer is only connected to network 1, gateway computers are connected to network 1 and related network 2, motor drivers are connected to related network 2 both from right and left sides. The tasks and real-time requirements of the three types of computers are divided in the following way:

- Main computer generates position reference and motion profile and sends to the related motor drivers. This is necessary when a passenger pushes the button to call the elevator. In order for the mover to arrive at the correct floor, the main computer sends the necessary messages to the related motor drivers through network 1.

- Gateway computers relay messages from main computer to motor drivers and do the necessary motor control calculations.

- Motor drivers of this topology have the same responsibilities with the motor drivers of topology A, such as generation of sinusoidal coil excitation signal, measurement of LVDT position sensor, current regulation and similar low level control and measurement duties. These duties are explained in detail in Section 3.1.

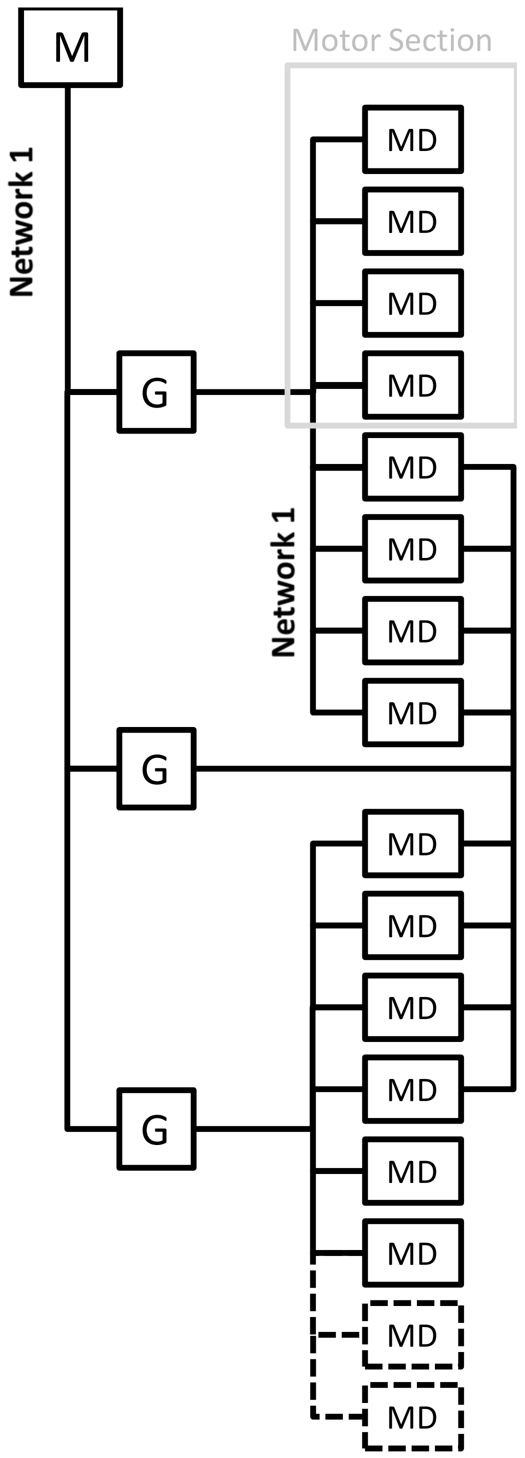


Figure 7. Topology B

3.2.1. Suitable Communication Networks for Topology B

Similar to topology A, a network with a high communication and addressing capacity that is sufficient for a large number of motor drivers and no real-time capability is required for network 1. Therefore it is chosen as Ethernet. Network 2 is chosen as CAN, a simpler and low latency network.

3.2.2. Advantages of the Method

The main advantage of this topology compared to topology A is the elimination of the time delay that is caused by using the gateway computers. Using the gateway computers makes the motor driver tasks easier since most of the complex calculations are achieved by the gateway computers. However, in order for the motor drivers to communicate with the main computer and the other motor sections, the only way is using gateway computers. And, since the gateway computers have to perform many tasks, there may be traffic and time delays may occur. Actually, the process of sending the messages using gateway computers causes time delays itself even if there is no traffic in the system. For some systems, time delays can be acceptable, but since the motor driving and control is a hard real time process time delays are more crucial and are not acceptable if they are more than a limit. In order to avoid time delays caused by using the gateway computers, topology B uses the motor drivers themselves for communicating with the other motor sections.

3.2.3. Disadvantages of the Method

In topology B, we observe uniformly distributed delays in the communication of the motor drivers. Although they do not cause torque fluctuations and vibrations in the system like seen in topology A, these uniform delays lead to deceleration in the system. However, since it is hard to avoid delay problem in communication networks, it is acceptable.

Chapter 4

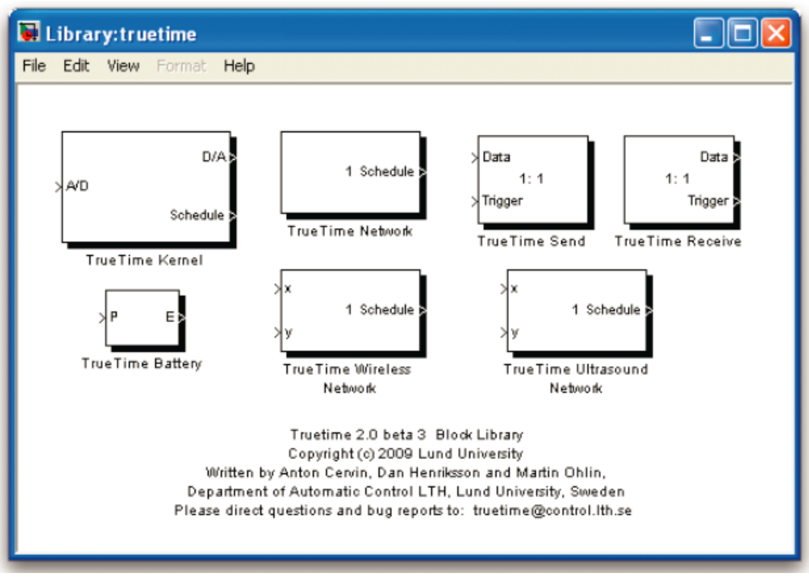
Simulation Models and Environment

Networked control systems are hybrid systems where continuous time-driven dynamics and discrete event-driven dynamics interact. Delays can lower the performance. Software tools are needed to analyze and simulate how the timing affects the control performance [26]. Using a simulation tool during the development stage is very helpful for such systems. It is required to simultaneously simulate the computations within the nodes, the communication between the nodes, the sensor and actuator dynamics [27].

4.1. Truetime

TrueTime is a Matlab/Simulink-based simulator for networked and embedded control systems developed at Lund University since 1999. The simulator software is composed of a Simulink block library seen in Figure 5 and a collection of MEX files. The kernel block (computer node) simulates a real-time kernel on an embedded computer executing user-defined tasks and interrupt handlers. The network blocks allow kernel blocks to communicate over simulated wired or wireless networks [29]. The TrueTime blocks can be connected with ordinary Simulink blocks to create a real-time control system. The main advantage of TrueTime is the possibility of co-simulation of the interaction between the real-world continuous dynamics and the computer architecture in the form of task execution and network communication.

The Truetime block library consists of the TrueTime Kernel blocks simulating real-time kernels that execute user defined tasks and interrupt handlers, the Network blocks enabling nodes to communicate over simulated network, a couple of standalone interface blocks and the Battery block that allows modeling of battery driven operation. Before a simulation can be run, it is necessary to initialize kernel blocks and network blocks, and to create tasks, interrupt handlers, timers, events, monitors, etc. TrueTime allows the initialization code and the code that is executed on the computer nodes during simulation to be written either as Matlab Mfiles or as C++ code.



Truetime 2.0 beta 3 Block Library
Copyright (c) 2009 Lund University
Written by Anton Cervin, Dan Henriksson and Martin Ohlin,
Department of Automatic Control LTH, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

Figure 8. The TrueTime Block Library

4.1.1.Truetime Kernel Blocks

A computer node that is simulated as TrueTime kernel block has a generic real-time kernel, A/D and D/A converters, and network interfaces. An initialization script is used to configure the block. In this function, it is possible to create several objects as task, timers, interrupt handlers, semaphores, etc. These objects establish the software executing in the kernel block. The kernel continuously calls the code functions of the tasks and interrupt handlers. To write initialization scripts and the code functions either Matlab m-files code or C++ language may be used. The main advantage of using C++ is the speed, but creating m-file code is easier for the user. A variety of scheduling policies can be used in TrueTime kernel blocks, these can be fixed-priority scheduling, earliest-deadline-first scheduling or custom scheduling policies [29].

The kernel block is configured through the block mask dialog, see Figure 6, with the following parameters; name of init function, init function argument, number of analog inputs and outputs, number of external triggers, network and node numbers, local clock offset and drift. Init function argument is an optional argument to the initialization script. It can be any Matlab struct. The following points must be taken into consideration while Network and Node Numbers parameter is entered:

- ❖ If there is only one network in the simulation model, there is no need to specify the network number. It is considered as “1” that is the default network number.

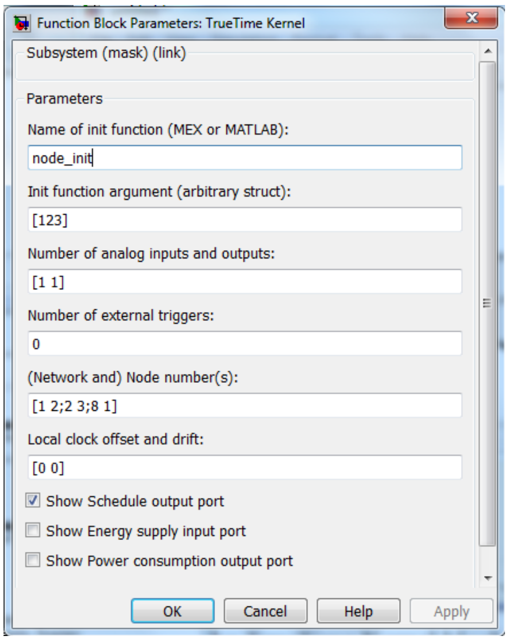
  An example of this case: [2]

  Here number “2” represents the node number. Network number is “1” as default.

- ❖ If there is more than one network in the model, the network numbers must be specified as well. A node can be connected to more than one network and can take different node numbers in each of these networks. In order to define a node that is connected to different networks, we should specify each network and node numbers of this node.

An example of this case: [1 2; 3 4]

It means the node is connected to network 1 with a node number of 2 and connected to network 3 with a node number of 4.



Function Block Parameters: TrueTime Kernel

Subsystem (mask) (link)

Parameters

Name of init function (MEX or MATLAB): node_init

Init function argument (arbitrary struct): [123]

Number of analog inputs and outputs: [1 1]

Number of external triggers: 0

(Network and) Node number(s): [1 2;2 3;8 1]

Local clock offset and drift: [0 0]

[✓] Show Schedule output port

[ ] Show Energy supply input port

[ ] Show Power consumption output port

Figure 9. The TrueTime Kernel Function Block Parameters

The *task* in the kernel is used to simulate periodic and aperiodic activities. A set of characteristics and a code function define a task. These characteristics are release time, worst-case execution time, relative and absolute deadlines, priority, period [30]. An example of the definition of a task is shown below:

```
function controller_init(arg)
% Distributed control system: controller node
% Initialize TrueTime kernel
ttInitKernel('prioDM') % deadline-monotonic scheduling
% Periodic dummy task with higher priority
starttime = 0.0;
period = 0.007;
data = period*arg;
ttCreatePeriodicTask('dummy_task', starttime, period, 'dummy_code', data);
```

The kernel primitive `ttInitKernel()` initializes a sensor node. The kernel is initialized by specifying the number of A/D and D/A channels and scheduling policy. The built-in priority function prioFP specifies fixed-priority scheduling. Rate monotonic prioRM, earliest deadline first prioEDF, and deadline monotonic prioDM scheduling are additional predefined scheduling policies [26].

Interrupts can be internal and external interrupt. The Truetime blocks are event-driven and support external interrupt handling. An external interrupt is associated with one of the external interrupt channels of the computer block. When the signal of the corresponding channel changes value the interrupt triggers. This type of interrupt is useful to simulate distributed controllers that execute when measurements arrive on the network. A user-defined interrupt handler is scheduled when an interrupt occurs. An interrupt is scheduled on a higher priority level. An interrupt handler is defined by name, a priority and a code function [30]. An example of a definition of a interrupt handler is as follows:

```
% Create and attach network interrupt handler
data = 'controller_task';
ttCreateHandler('network_handler', 1, 'nwhandler_code', data)
ttAttachNetworkHandler('network_handler')
```

Simulated execution can be preemptive or non-preemptive. The execution occurs at three distinct *priority* levels: the interrupt (highest priority), kernel and task (lower priority) levels. At interrupt level, interrupt handlers are *scheduled* according to fixed priorities. At kernel and task level, dynamic-priority scheduling may be used. At each scheduling point, the priority of task is given by user-defined priority function.This makes it easy to simulate different scheduling policies. Predefined priority functions exist for most of the commonly used scheduling schemes [29], [30].

While simulation is running the *code* associated with task and interrupt handlers is executed by the kernel. The code may be divided in segments, which can interact with other tasks. After the simulation is finished, the execution time of each segment is returned by the code function as an output [30]. The kernel saves the current segment during the simulation and calls the code functions. Execution pursues in the next segment when the task has been running for the time regarding to previous segment [32]. An example of a sensor code is given bellow:

```
function [exectime, data] = sensor_code(seg, data)

persistent y

switch seg
 case 1
  y = ttAnalogIn(1);
  exectime = 0.0005;
 case 2
  ttSendMsg(3, y, 80); % Send message (80 bits) to node 3 (controller)
  exectime = 0.0004;
 case 3
  exectime = -1; % finished

end
```

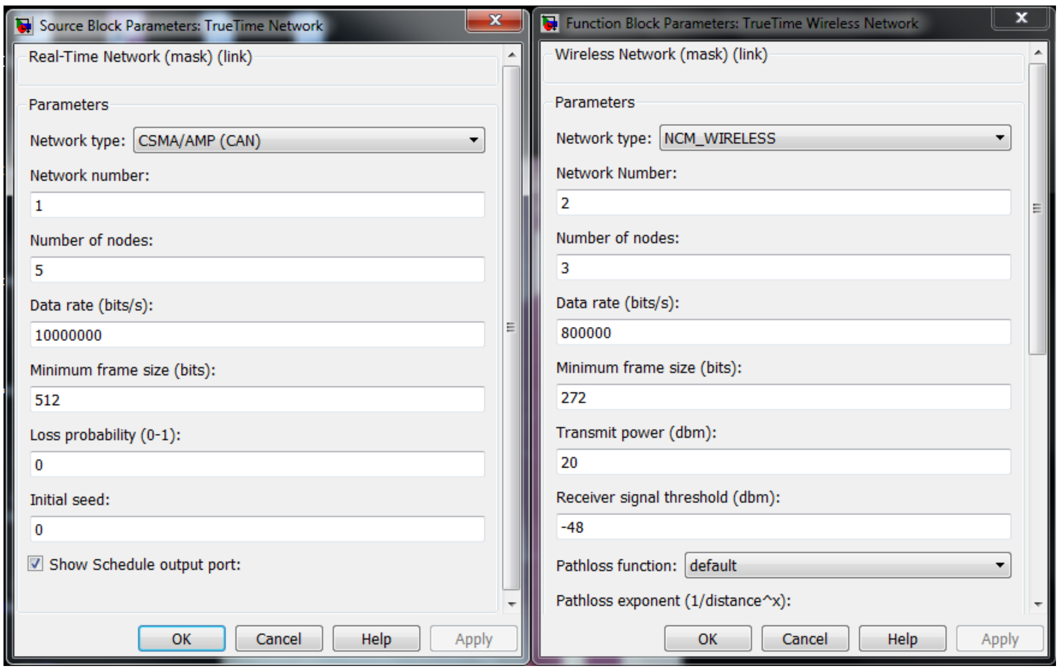
This function explain a simple sensor mechanism. In the first segment, data is received from analog input and the plant is sampled using a execution time of 0.5 ms. In the second segment, the message (control signal) is sent to the controller node. The third segment shows the end of execution by returning a negative execution time. The structure *data* is the local memory and used to store the measured variable between calls to the different segments [33].

Truetime blocks generate different *output graphs*; computer graphs and monitor graphs. A computer graph will display the execution of each task and interrupt handler. If the signal is high, it means that the task is running. If there is a medium signal,it means that the task is ready but not running.A low signal indicates that the task is idle. On the other hand, a monitor graph displays which tasks are holding and waiting on the different monitors during simulation [30].

4.1.2. Truetime Network Block

The TrueTime network block simulates medium access and packet transmission in a local area network (LAN). When a node tries to transmit a message, a triggering signal is sent to the related network block. When the transmission of the message is finished, a new triggering signal is sent by the network block to the receiving node. The transmitted message is saved in a buffer at the receiving computer node [33].

The network blocks are mainly configured using their block mask dialogues. There are some common parameters for all networks such as network type, network number, number of nodes, data rate or minimum frame size. Some of the parameters are specific to the network type like transmit power or receiver signal threshold in wireless networks. Wired and wireless network block masks are seen in the Figure 7. There can be several network blocks in a model. The ID numbers of each network is used to identify these networks. Connected nodes also have their ID numbers. The node ID numbers must be specific to their networks.


Source Block Parameters: TrueTime Network

Real-Time Network (mask) (link)

Parameters

Network type: CSMA/AMP (CAN)

Network number: 1

Number of nodes: 5

Data rate (bits/s): 10000000

Minimum frame size (bits): 512

Loss probability (0-1): 0

Initial seed: 0

[✓] Show Schedule output port:

Function Block Parameters: TrueTime Wireless Network

Wireless Network (mask) (link)

Parameters

Network type: NCM_WIRELESS

Network Number: 2

Number of nodes: 3

Data rate (bits/s): 800000

Minimum frame size (bits): 272

Transmit power (dbm): 20

Receiver signal threshold (dbm): -48

Pathloss function: default

Pathloss exponent (1/distance^x):

Figure 10. The TrueTime Wired and Wireless Network Function Block Parameters

The nodes send and receive messages over the networks belonging to them using `ttSendMsg()` and `ttGetMsg()` kernel primitives. An overview of all Truetime's primitives can be found on Truetime Manual.

The types of networks supported by Truetime are CSMA/CD (Ethernet), CSMA/AMP (CAN), Round Robin (Token Bus), FDMA, TDMA (TTP), Switched Ethernet, WLAN(802.11b), and ZigBee (802.15.4). The networks are simulated down to the data transport layer.

Truetime models of CSMA/CD (Ethernet) and CSMA/AMP (CAN) protocols used in this project are explained and their comparison with those used in real applications are done in the following sections.

4.1.2.1. CSMA/CD (Ethernet)

CSMA/CD stands for Carrier Sense Multiple Access with Collision Detection. If the network is busy, the sender node must wait until it is free. A collision occurs if a message is transmitted within 1 microsecond of another. When a collision occurs, the sender node backs off for a time defined by:

$$t_{backoff} = minimum\,frame\,size \div data\,rate \times R$$

Where $R = rand(0.2^{K} - 1)$ (discrete uniform distribution) and K is the number of collisions in a row. K can be maximum 10 and minimum frame size cannot be 0. After waiting time is finished, the node will try to retransmit.

4.1.2.2. CSMA/AMP (CAN)

As the same as real CAN applications, if the network is busy, the sender will wait until it occurs to be free. If a collision occurs, the message with the highest priority (the lowest priority number) will continue to be transmitted. On the other hand, there is a difference on message choice when two messages with the same priority seek transmission simultaneously. In that case an arbitrary choice is made as to which is transmitted first. However, in real CAN applications, all sending nodes have a unique identifier, which serves as the message priority and the choice is made according to these priorities [33].

4.2. Building Simulink Models using MATLAB Scripts

The linear motor is made up of a large number of sections and modules that are repeated. Since we wish to analyze the performance of several different network topologies and communication protocols, models of different motors must be prepared. One contribution of this thesis is the development of a Matlab script based method to create a Simulink model. Such an approach makes it easier to generate a model and change parameters of the model.

It is possible to build a Simulink model using Matlab scripts, without using the standard Matlab graphical user interface based mouse operations. Although building a Simulink model with the usual operations is easier and preferred by most of the users, using Matlab script to generate models is more advantageous as follows:

* Ease of building and modification- The linear motor that will be modeled in Simulink is composed of repeated modules, consisting e.g., a fixed number of motor drivers connected to one gateway. These modules are called “motor sections”. Number of motor sections may

change depending on the length of the motor, or design considerations. Therefore, it is necessary to modify the model frequently. However, it is difficult and complicated to build such a large model with GUI operations. Also, it is not easy to modify even if the model has been created with the GUI method since it requires several repetitive changes in each system block. Using Matlab scripts for building a model enables the model to be created and modified easily.

* Ease of parameter change- The Simulink model is used to simulate the system and to determine the performance of the selected topologies and network protocols. In order to compare their performances, we need to change the parameters and decide on the optimum one. It is easy to change the parameters of several blocks with a few modifications in the MATLAB script.

This section discusses how to build and manipulate a Simulink model using only m-scripts. The main steps to create a model automatically are explained:

1. The parameters are introduced.

2. The scripts then checks to see if a model with the specified name already exists and if it does then it deletes it.

3. A new model is created using the MATLAB function `new_system`.

4. The model is filled in using MATLAB functions such as `add_block`, `add_line` and `set_param`, etc. `set_param` is necessary when the default values of the block properties are to be modified. After the model is constructed, it is saved using `save_system`. Creating a model will be explained in detail in the following paragraphs.

Since TrueTime has its own library, there are a few differences while using MATLAB to create a TrueTime model.

The scripts written for creating our linear motor model is described in detail below:

i. The name of the model that will be created is specified.

```
% Specify the name of the model to create
fname = 'Topoloji_A';
```

ii. The parameters are introduced at the beginning. In order to easily modify the model, the numbers such as number of gateways, number of drivers must be defined as constants. The parameters dependent on these numbers are assigned automatically in the scripts.

```
% Introduce the parameters
Outer_Network_No=50000; %outer network number
Num_of_Gate=3;  %number of gateway computers
Num_of_Drivers=10; %number of motor drivers
Const=1000; %max group number can be 1000
Const2=40000; %parameter used for naming the 2nd scope of the Gateway
init_function='Tum_sistemi_baslatan';  %initialization function for
all Truetime blocks
```

iii. It is checked to see if a model with the specified name already exists and if it does then it is deleted.

```
% Check if the file already exists and delete it if it does
if exist(fname,'file') == 4
    % If it does then check whether it's open
    if bdIsLoaded(fname)
        % If it is then close it (without saving!)
        close_system(fname,0)
    end
    % delete the file
    delete([fname,'.mdl']);
end
```

iv. A new model is then created using the MATLAB function `new_system`.

```
% Create the system
new_system(fname);
```

v. In contrast with adding blocks from Simulink Library, it is required to open TrueTime Library and the created model to add blocks. This operation is based on copying the blocks from TrueTime Library and pasting to the new model.

```
% Open truetime library and the created system to copy(add) blocks
open_system('truetime');
open_system('Topoloji_A');
```

vi. The MATLAB functions used to create the model are `add_block`, `add_line` and `set_param`. `set_param` is necessary when the default values of the block properties must be modified. Usage of these functions are introduced below:

- `add_block('src', 'dest', 'param1', value1,'param2', value2,...)` creates a copy of the `'src'` block, with the named parameters having the specified values.
- `set_param(object,param1,value1,...,paramN,valueN)` sets parameters and values on the specified model or block object. Some properties are *read-only* and hence cannot be modified. Block parameter names used in `set_param` is different from the dialog box prompt. For instance, the parameter name of the dialog box prompt "Sample time (-1 for inherited)" is "SampleTime". For Simulink blocks, the detailed information about *the parameter names* with their *dialog box prompt* and *the values* showing the type of value required (scalar, vector, variable), the possible values, and the default value can be reached from Matlab Simulink Help under "Block-Specific Parameters" section. For TrueTime blocks, such information can be found as described: Bring the cursor on to the block, then right click and choose *View Mask* and the *Mask Editor* is opened automatically. The dialog prompts, variable names belonging to the parameters and types can be seen under "Parameters" section.
- `add_line('sys', 'oport', 'iport')` adds a straight line to a system from the specified block output port `'oport'` to the specified block input port `'iport'`. `'oport'` and `'iport'` are strings composed of a block name and a port identifier in the form 'block/port'. Most block ports are identified by numbering the ports from top to bottom or from left to right, such as 'Gain/1' or 'Sum/2'.

vii. In our model both commonly used Simulink blocks and TrueTime blocks are used together. Adding Simulink blocks and adding TrueTime blocks are explained below in detail with their usage in the scripts.

- Adding Simulink Blocks- In our model, clock, display, gain, sum, sine wave and scope blocks are used. The inputs of `add_block` and `add_line` functions must be entered in the order that is given as `add_block('src', 'dest', 'param1', value1,'param2',... value2,...)` before. For example in order to add a clock to the system, first the source ('`src`') of the block is defined as `'built-in/Clock'` and the destination ('`dest`') is defined as `[gcs,'/Clock']`. Then all the parameters of the block are defined with their values. In the first example below, there is only one type of parameter that is `'Position'` and its value is `[800 20 830 50]`. `add_block` function achieves its task by copying the block from the source and pasting it to the destination. More examples are presented below:

```
% Add Clock
add_block('built-in/Clock', [gcs,'/Clock'],'Position', [800 20 830
50]);

% Add Display
add_block('built-in/Display', [gcs,'/Display'],'Position', [850 20
900 50]);

% Connect Clock and Display
add_line('Topoloji_A','Clock/1','Display/1');

% Add Gain
add_block('built-in/Gain', [gcs,'/Gain'],'Position', [850 150 880
180], ...  'Gain','100','SampleTime','-1');

% Add Sum
add_block('built-in/Sum',[gcs,'/','Sum'])

% Set Sum Block Parameters
set_param([gcs,'/','Sum'],'inputs','++','position',[900,150,930,18
0])

% Add Constant
add_block('built-in/Constant', [gcs,'/Constant'],'Position', [850
250 880 280],...
'Value','26','SampleTime','inf');

% Connect First Output of Constant Block to Second Input of Sum
Block
add_line('Topoloji_A','Constant/1','Sum/2');

% Add Scope
add_block('built-in/Scope', [gcs,'/Scope'],'Position', [100 (10)
130 (40)],... 'LimitDataPoints','off');
```

- Adding TrueTime blocks-In our model, TrueTime kernel and network blocks are used. Since the linear motor consists of several motor sections, including one gateway computer and fixed number of motor drivers, for loops and if-else conditions are used to create these similar blocks. Since `set_param` and `add_block` functions requires these parametric numbers to be as string type, `num2str` is used to change the numbers to string. The following examples show their usage:

```
% Generate OUTER NETWORK BLOCK
add_block('truetime/TrueTime Network','Topoloji_A/Outer
Network',...
'Position', [10 (20) 70 (70)]);

set_param('Topoloji_A/Outer Network',...
'nnodes',num2str(Num_of_Gate+1),'nwnbr',num2str(Outer_Network_No),
'nwtype','CSMA/AMP (CAN)','rate','10000000','minsize','80');

add_block('built-in/Scope', [gcs,['/Scope'
num2str(Outer_Network_No)]],...
'Position', [100 (10) 130 (40)],'LimitDataPoints','off');

add_line('Topoloji_A','Outer Network/1',['Scope'
num2str(Outer_Network_No) '/1']);
```

While using `add_block` function, “TrueTime Network” block is copied from “truetime” library to the model where model name is “Topoloji_A” and the name of this block in new system is determined as “Outer Network”,then the position of the block is specified. In order to set block properties of TrueTime Network block, `set_param` function is used. First, the name of the model, “Topoloji_A”, and the block name, “Outer Network” are introduced then the parameters such as number of nodes (`nnodes`), network number (`nwnbr`), network type (`nwtype`) , data rate (`rate`) and minimum frame size (`minsize`) are introduced as below. Name of the parameters such as nnodes, nwnbr, nwtype can be learned as explained previous section. Number of nodes change according to number of gateway computers. Therefore, its value is introduced as parameter. Also, to make network number as variable, it is also introduced as parameter.

Since Matlab does not allow to use the same block name more than once, the block names are specified as unique to each block. For similar blocks, the same name but with different numbers such as “Truetime Network 1”, “Truetime Network 2”, “Truetime Network 3” etc. are used. The total number of the motor drivers of the designed linear motor are assumed to be maximum 1000. Therefore, since each motor section elements (the motor drivers and the gateway computer of the section) communicate over their own network, the number of networks for these sections will be the same amount. The gateway computers and the networks for the sections have the same number such as for the first motor section the name of network is “Truetime Network 1” and the name of the gateway computer is “Gateway Computer 1”. The number of motor drivers is constant for each motor section. In the real motor, there must be 10 motor drivers in each section. In order to easily distinguishing the motor drivers of each section, they are named depending on this simple formula:

The node IDs (d_{ID}) for the motor drivers (m_d) in each motor section (m_s) should be unique. This was accomplished using a simple coding method: d_{ID}=m_s*K+m_d, where K is an appropriate constant taken as 1000. For example; when the number of the motor section is m_s=2, the name of the motor driver m_d=5 of this section becomes d_{ID}=2005. Here, Scope block is named with the Outer_Network_No that is "Scope 50000".

Scope blocks automatically show the last 5000 data. However, it prevents the results to be evaluated fairly. Therefore, the default value of "LimitDataPoint" parameter is changed from "on" to "off" to see all data.

```
% Generate MAIN COMPUTER BLOCK

add_block('truetime/TrueTime Kernel', 'Topoloji_A/Main
Computer',...
'Position', [60 110 120 160]);

set_param('Topoloji_A/Main Computer','sfun',init_function,'args',
... ['['num2str(Outer_Network_No) ']'],'ninputsoutputs','[1
1]',...
'nwnodenbr',['[', num2str(Outer_Network_No),' ',
num2str(Num_of_Gate+1),']']);
```

`add_block` function is used as similar to adding TrueTime Network block. Here, TrueTime Kernel block is copied from truetime library and added to the model with "Main Computer" name. The parameters name of init function (`sfun`), init function argument (`args`), number of analog input and outputs (`ninputsoutputs`), network and node numbers (`nwnodenbr`) are set in `set_param` function.

Nested for loops are used to create Driver Networks, Gateway Networks, Motor Drivers and Gateway Computers. Whole scripts creating the model is attached at the end of this report in Appendix section.

4.3. Scripts Used in TrueTime Blocks

It is necessary to initialize kernel blocks and network blocks. In order to define the parameters, to create tasks and interrupt handlers these initialization functions are used. The initialization scripts and the scripts that is executed during simulation may be written either as Matlab M-files or as C++ code (for increased simulation speed). In this project, the Matlab M-files are preferred since they are easy to use. How the code functions are defined and what must be provided during initialization will be described below.

4.3.1. Writing the Initialization Function

Initialization of a TrueTime kernel block involves specifying the number of inputs and outputs of the block, defining the scheduling policy, and creating tasks, interrupt handlers, events, monitors, etc for the simulation. This is done in an initialization script for each kernel block.

In our model, initialization script is named as “`initialization`”. The initialization script used in Topology A model will be introduced step by step:

1. Initialization of TrueTime Kernel: The kernel is initialized by providing the scheduling policy using the function `ttInitKernel`.

```
function initialization(mode)

% Initialize TrueTime kernel
ttInitKernel('prioDM');   % deadline-monotonic scheduling
```

2. Definition of Parameters: All parameters are defined in the initialization script. So that, their values can be changed from one place easily. Struct data type is preferred to use since it enables several parameters grouped together under one name . The data structure, consisting of the data sent as messages between the nodes, is named as `msg`. The data structure, consisting of the parameters used to create the simulink model, is named as `data.` The related scripts are presented below:

```
% Define the parameters
msg.sender_ID=0; %% node number of the sender node
msg.outer=0; %% the message sent by main computer via outer network
msg.pos_of_sensor=0;  %% position of the sensor
msg.des_ID=0;  %% node number of the destination node
num_of_gateways=3; %%number of gateways
num_of_drivers=6; %%number of drivers
driver_length=25; %% the length of one motor driver
sensor_offset=26; %%sensor position is 26 cm far from the bottom of the
mover.
data.kernel_ID = mod(mode,100); %%used to determine the type of the kernel
as main computer,gateway computers, motor drivers
data.network_ID= (mode-data.kernel_ID)/1000;   %%network number
data.first_driver_ID=2; %%in each motor section, gateway has the node
number "1". The node number of the drivers start from 2 and goes on as
3,4,5,...
data.last_driver_ID=num_of_drivers+1; %%since the node numbers of the
drivers start from "2", the last driver has the node number: (number of
drivers + 1)
data.num_of_drivers=num_of_drivers;
data.num_of_gateways=num_of_gateways;
data.sensor_offset=sensor_offset;
data.driver_length=driver_length;
data.pos_down=(data.network_ID - 1)*num_of_drivers*driver_length +
(data.kernel_ID - 1)*driver_length; %%lower limit of the position of the
driver
data.pos_up= (data.network_ID - 1)*num_of_drivers*driver_length +
(data.kernel_ID)*driver_length;     %%upper limit of the position of the
driver
```

3. Calculation of the Network IDs of Upper and Lower Gateway Networks: The gateway computers communicate with each other via the networks called “Gateway Network” between them and every two adjacent gateways have a different network between each other as seen in Figure 6. The network IDs of upper and lower gateway networks are calculated using a simple coding method. In order to calculate the upper network number (net_no_{up}) of a gateway, the following code is used: $net\_ID_{up}=K+gwID-1$ where K is appropriate constant taken as 1000, *gwID* is the ID number of the gateway and *gwID-1* is the ID number of the previous (upper) gateway. For example; for the gateway with ID number of *gwID=2* , the ID of upper gateway network becomes $net\_ID_{up}=1001$. In order to calculate the lower network number (net_no_{low}) of a gateway, the following code is used: $net\_ID_{low=K+gwID}$ where K is appropriate constant taken as 1000, *gwID* is the ID number of the gateway and *gwID+1* is the ID number of the next (lower) gateway. For example ; for the gateway with ID number of *gwID=2* , the ID of lower gateway network becomes $net\_ID_{low}=1002$.

Instead of using the node number of each node in the scripts, we generalize them as upper or lower gateway based on their position in the system. Therefore, we do not need to calculate their node numbers everytime when we use them in the scripts.

```
%%calculate upper and lower gateway network numbers
if data.network_ID<data.num_of_gateways
   data.lower_gw_network_no=1000+(data.network_ID);
end
if data.network_ID>1
data.upper_gw_network_no=1000+(data.network_ID-1);
end
```

4.Creating Periodic and Aperiodic Tasks: In order to create periodic activities such as controller and I/O tasks, `ttCreatePeriodicTask` function is used. The matlab syntax for this function is :

```
ttCreatePeriodicTask(name, starttime, period, codeFcn, data)
```

The arguments of the function are stated below:

`name`: Name of the task. It must be unique.
`starttime`: Release time for the first job of the periodic task.
`period`: Period of the task.
`codeFcn`: The code function of the task, where codeFcn is a name of an m-file in the Matlab.
`data`: An arbitrary data structure representing the local memory of the task.

In order to create aperiodic activities such as communication tasks and event-driven controllers, `ttCreateHandler` and `ttAttachNetworkHandler` functions can be used together.

The matlab syntax for the `ttCreateHandler` function is:

```
ttCreateHandler(name, priority, codeFcn)
```

where `name`, `priority`, `codeFcn` stand for name of the handler, priority of the handler, the code function of the handler, an arbitrary data structure, respectively. The code function of the handler includes the jobs of this handler.

This function is used to create a handler that will be executed in response to interrupts. Interrupt handlers may be associated with timers, network interfaces, external interrupt channels, or attached to tasks as overrun handlers. In this project, networked interfaces are used a to trigger the interrupt handlers. Therefore, the code function of the handler is invoked when a message arrives over the network. A handler may be associated with many interrupt sources.

The matlab syntax for the `ttAttachNetworkHandler` function is:

```
ttAttachNetworkHandler(network, handlername)
```

where `network` and `handlername` stand for The number of the TrueTime network block and The name of the interrupt handler, respectively.

Since the same initialization function is used by all type of kernels in the system (main computer, gateway computers, motor drivers), if-else statements are used in the script. Therefore, each type of kernel will execute the condition related to them. Each kernel in the simulation system has unique "mode numbers" and these numbers are used to distinguish the computers. These mode numbers are defined as "init function arguments" in the "Source Block Parameters" of each kernel blocks.

The scripts below, written for creating periodic and aperiodic tasks, clarify the usage of the TrueTime functions explained previously:

```
%%%%Since this initialisation code is used for all type of kernels, each
%%%%type of kernels (main computer,gateway computer,motor drivers) will run
%%%%the condition related to them.

if mode ==50000 % (main computer)
    starttime = 0.0;
    period = 0.010;
    ttCreatePeriodicTask(['Main_computer_task' num2str(mode)], starttime,
period, 'Main_computer',data);
end


if data.kernel_ID==1 % (gateway computers)
    deadline = 10.0;
    % Network handler triggered by Outer Network (node number=50000)
    prio = 1.0;
    ttCreateHandler(['Gateway_computers_outer_network_Task' num2str(mode)],
prio, 'Gateway_computers_outer_network', data);
    ttAttachNetworkHandler(50000,['Gateway_computers_outer_network_Task'
num2str(mode)])

    % Network handler triggered by TrueTime Network Blocks used for
connecting the motor section elements (node number=data.network_ID)
    ttCreateHandler(['Gateway_computers_drivers_network_Task'
num2str(mode)], prio, 'Gateway_computers_drivers_network', data);
```

```
ttAttachNetworkHandler(data.network_ID,['Gateway_computers_drivers_network_
Task' num2str(mode)])

    %%for all motor section networks except the last one
    if data.network_ID<data.num_of_gateways
        ttCreateHandler(['Gateway_computers_lower_network_Task'
num2str(mode)], prio, 'Gateway_computers_lower_network', data);

ttAttachNetworkHandler(data.lower_gw_network_no,['Gateway_computers_lower_n
etwork_Task' num2str(mode)])
    end

    %%for all motor section networks except the first one
    if data.network_ID>1
        ttCreateHandler(['Gateway_computers_upper_network_Task'
num2str(mode)], prio, 'Gateway_computers_upper_network', data);

ttAttachNetworkHandler(data.upper_gw_network_no,['Gateway_computers_upper_n
etwork_Task' num2str(mode)])
    end

end


if data.kernel_ID>1 % (drivers)
    deadline = 10.0;
    % Network handler
    prio = 1.0;
    ttCreateHandler(['Motor_Drivers_Task' num2str(mode)], prio,
'Motor_Drivers',data);
    ttAttachNetworkHandler(data.network_ID,['Motor_Drivers_Task'
num2str(mode)])
end
```

4.3.2. Writing the Task Functions

The task functions are invoked by the initialization function. In our model, we have three different type of kernels that are main computer, gateway computers and motor drivers. And all these kernels have different tasks. One kernel may have more than one task, such as interrupt handlers. If these tasks are trigerred by different network handlers, it is useful to create different script files for each type of these tasks. For example, in our model gateway computer has to complete several tasks trigerred by different network handlers, so a different script file is created for each type of task. The two main TrueTime functions used in these task functions are `ttGetMsg` and `ttSendMsg`. `ttGetMsg` function is used to retrieve a message that has been received over the network. If no message exists, the function will return an empty value. `ttSendMsg` function is used to send a message over a network.

The matlab syntax for the `ttGetMsg` function is:

```
msg = ttGetMsg(network)
```

where “network” stands for the network interface from which the message should be received. The default network number is 1.

The matlab syntax for the ttSendMsg function is:

```
ttSendMsg([network receiver], data, length)
```

The meaning of the input arguments are stated below:

network: The number of the network on which the message is sent. The default network number is 1.
receiver: The ID of the receiving node.
data: An arbitrary data structure representing the contents of the message.
length: The length of the message, in bits. Determines the time it will take to transmit the message.

The following scripts clarify how to use these functions:

```
msg = ttGetMsg(data.network_ID);
        if isempty(msg)
            disp(msg)
            disp('Error nwhandler_code : no message received!');
            msg = 0.0;
        else
```

First the message sent through the network with number “data.network_ID” is received, then it is checked if it is an empty message. If it is an empty message, it returns 0 value.

```
ttSendMsg([data.network_ID 1 ], msg , 16);
```

16 bit length message is sent to the node with ID number “1” through the network with network number “data.network_ID” .

Chapter 5

Simulation Results

In simulations, due to some restrictions in Matlab, some real system requirements could not be realized as desired. However, it is created as similar as possible to the real motor. The real system consists of a main computer and 800 motor sections each with 10 motor drivers and one gateway computer. The simulation model is a prototype of the real system consisting of a main computer and 4 motor sections, each spanning 1m with 4 motor drivers and one gateway computer. The reason for selecting this size was that for a larger model with greater memory requirements, Matlab faces problems with data logging. However, with a suitably powerful computer the simulations are believed possible. The control loop has a frequency of 7 kHz and a period of 142 µs (0.000142s). The speed of the motor is 4 m/s in the model. Each motor driver has 0.25 m length. Therefore, in order the mover to rise up to the height of the whole motor (4 m), we need to run the model for a time duration of 1s. In real life 4 m/s is too fast and 1.5 m/s is more reasonable. However, since the control loop has a frequency of 7 kHz, the total number of messages sent and received is so many causing the simulation to slow down. In order to realize the real velocity of 1.5 m/s, we need to run the simulation for longer times. Therefore, mover velocity is chosen as 4 m/s in the simulations.

As explained in Section 3, two different communication protocols are used in the system. The communication on the inform network is carried out using Ethernet protocol and the communication on the synchronization network is carried out using CAN protocol. Ethernet communication is used to convey the messages containing various system related information sent by main computer to gateway computers with a period of 0.01s. For all other communications between the nodes, CAN protocol is used. It is assumed that the messages with 16 bits data length for CAN protocol (containing current mover position) and 1000 bits data length for Ethernet (containing various system related information) are enough for the system communication. In CAN protocol, each packet has 60 bits length consisting of 44 bits header and 16 bits data. The data rate of CAN protocol is set to 1Mbits/s that is the maximum data rate for CAN protocol. Therefore, the time required for the communication on CAN is 60µs. In Ethernet protocol, each packet has 1144 bits length consisting of 14 byte header, 4 byte CRC and 1000 bits data. The data rate for Ethernet is set to 10 Mb/s (10,000,000bits/s) as a commonly accepted value.

In Matlab scripts, task execution times are defined as 10µs, 14.2µs for gateway computers and motor drivers respectively. Messages are sent 3 µs later than the time that task executions start.

5.1.Topology A

Topology A as described in the Chapter 3 is investigated here. In the simulations, the messages between motor drivers are sent in one direction simulating the mover going up. The topology consists of several motor sections. Elements of each motor section communicate through their group network. The adjacent motor sections communicate via gateway computers relaying the messages. The messages sent by the main computer are also relayed to motor drivers via gateway computers. When more than one node demands to use the network at the same time, collision may occur and one node must wait until the network is free as in CSMA-CD protocol. In the following network traffic graphs, a high level signal means the node is sending, a medium level signal means waiting (e.g. for media access), and a low level signal means idle. In the simulation results, we rarely observe large positive delays due to the periodic messages sent by the main computer to the motor drivers through gateway computers causing a network traffic.

We used time stamps to analyze the delays in each node statistically. In order to observe the steps a message is following, we should examine the schedule output ports of the nodes.

As explained in Section 3, since gateways are used to relay messages between adjacent motor sections, extra delays occur in this transmission. In order to understand how these extra delays occur, we should examine each step of a message from sender to receiver. For example, a message sent from “motor section 2- motor driver 1” to “motor section 1- motor driver 4” follows these steps:

1. From “motor section 2- motor driver 1” to “gateway computer 2” through “network 2” that is the group network of motor section 2.

2. From “gateway computer 2” to “gateway computer 1” through “network 1-2” that is the network between “gateway computer 2” and “gateway computer 1”.

3. From “gateway computer 1” to “motor section 1- motor driver 4” through “network 1” that is the group network of motor section 1.

In order to observe each step of this transmission, schedule output ports of motor section 2- motor driver 1 (MS2-MD1), network 2 (N2), gateway computer 2 (GW2), network 1-2 (N12), gateway computer 1 (GW1), network 1 (N1), motor section 1- motor driver 4 (MS1-MD4) are combined in one graph seen in Figure 11. In the graph, motor section 2- motor driver 1, network 2 , gateway computer 2, network 1-2, gateway computer 1, network 1, motor section 1-motor driver 4 are represented with brown, brown, pink, turquoise, red, green, blue colors respectively, from down to up.

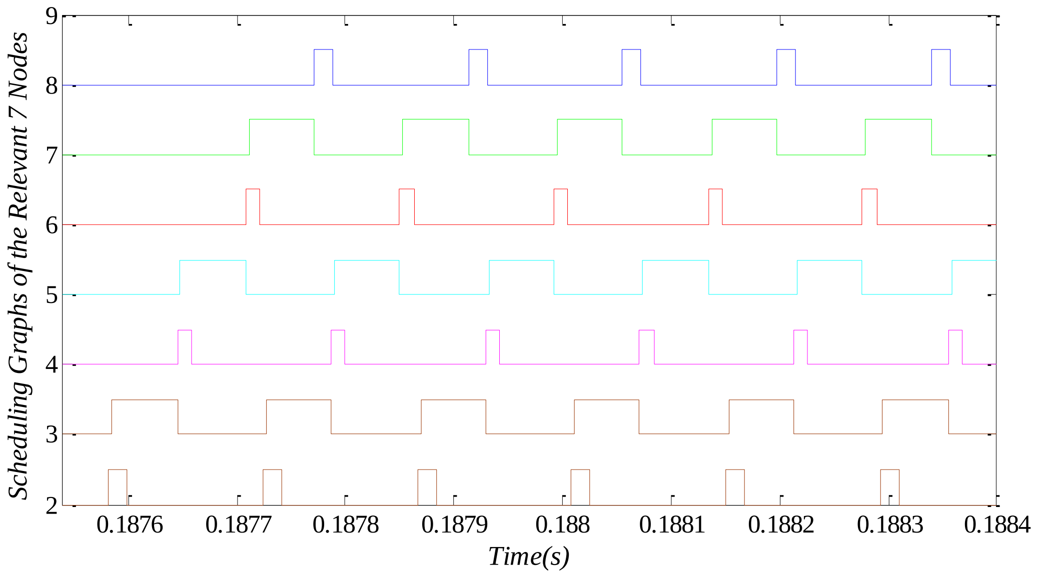


Figure 11. Message Transmission Between the Motor Drivers of the Adjacent Motor Sections
(Topology A) (motor section 2- motor driver 1, network 2 , gateway computer 2, network 1-2, gateway computer 1, network 1, motor section 1-motor driver 4 are represented as 1st, 2nd, 3rd, 4th, 5th, 6th, 7th scheduling graphs from down to up, respectively.)

Figure 12 represents all steps, marked with black colored squares, to transmit only one message. The brown signal in the bottom belongs to motor section 2-motor driver 1. It represents the task execution time of MS2-MD1. It is seen that this time is 14.2 µs as defined in Matlab scripts. Messages are started to sent to the other node 3 µs later than the time that task execution of this node start. The second brown signal belongs to network 2. It represents the message transmission time between MS2-MD1 and GW2. When we look the graph in detail, we observe that this transmission time is 60 µs that matches with the required time to communicate on CAN explained in the beginning of this chapter. Pink signal belongs to gateway computer 2. It represents the task execution time of gateway computer 2 that is 10 µs. It is seen that this execution time matches with the execution time of gateway computers defined in Matlab scripts. Messages are started to sent to the other node 3 µs later than the time that task execution of this node start. Blue signal belongs to network 1-2 and represents the message transmission time between GW2 and GW1. The duration is 60 µs as expected. Red signal belongs to gateway computer 1 and represents the task execution time of gateway computer 1. The duration is 10 µs as expected. Messages are started to sent to the other node 3 µs later than the time that task execution of this node start. Green signal represents the message transmission time between GW1 and MS1-MD4. The duration is 60 µs same as the other message transmission times. Blue signal belongs to MS1-MD4. It represents the task execution time. It is 14.2 µs as defined in Matlab scripts.

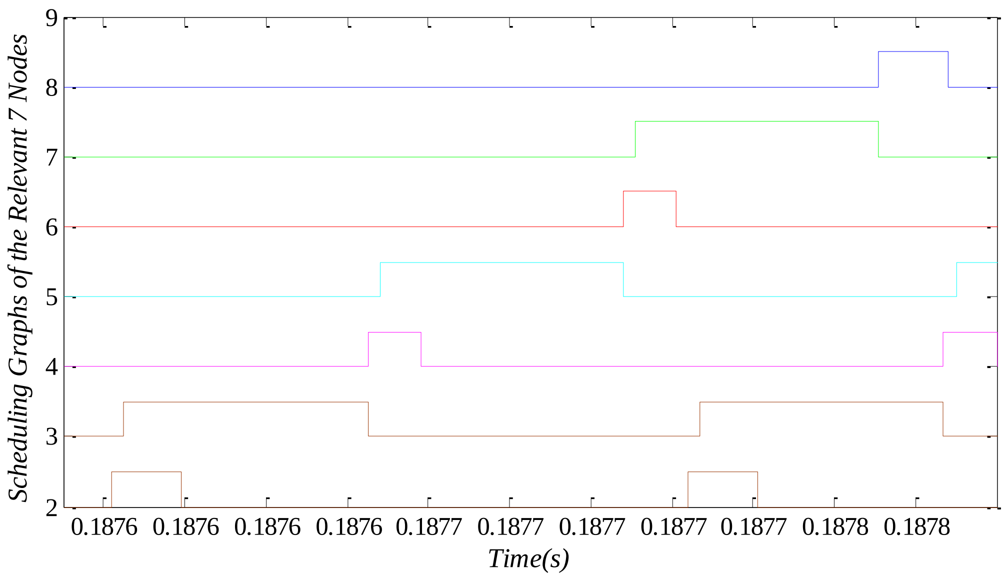


Figure 12. Message Transmission Between the Motor Drivers of the Adjacent Motor Sections (In detail) (Topology A) (motor section 2- motor driver 1, network 2 , gateway computer 2, network 1-2, gateway computer 1, network 1, motor section 1-motor driver 4 are represented as 1st, 2nd, 3rd, 4th, 5th, 6th, 7th scheduling graphs from down to up, respectively.)

As a result, it is seen all measured values of execution time and message transmission time match with the predefined values. As seen in Figure 12, messages are sent to the other nodes 3 µs later than their task execution starts. In that case, sending a message between the motor drivers of the adjacent motor sections will take $(60 \times 3) + (3 \times 3) = 189$µs that is the total amount of time spent during three Can communications and the delays caused by the task execution of the three nodes used in that communication . Here, we represent the steps to transmit one message between "motor section 2-motor driver 1" and "motor section 1-motor driver 4". Same steps are followed to relay a message between motor drivers of the other adjacent motor sections. Therefore, we will not show them separately. In general, three CAN communications and three nodes are used to relay a message between the motor drivers of adjacent motor sections. However, in order to relay a message between the motor drivers of the same motor section only one CAN communication and one node is used, so, 63 µs is spent in that communication. Due to these reasons, extra delays occur in the message transmission between the motor drivers of the adjacent motor sections.

Figure 13 is presented to show how delays are changed in each motor drivers of a motor section with respect to time. This graph belongs to motor section 1 as a representative, and all other motor sections have results similar to it. First three signals (pink, green, blue) represent the delays in MD1, MD2, MD3 respectively and the last signal represent the delays in MD4. It is obviously seen that the delay in MD4 is three times higher than the others. It means that the communication is carried out using the gateway computers. The spikes in the delays are caused by the periodic messages sent by the main computer to the motor drivers using the gateway computers. Since these messages are sent to the motor drivers via the same network used for the communication of the motor drivers, while these messages are received the communication of the motor drivers is disrupted and these spikes in the delay amounts occur periodically.

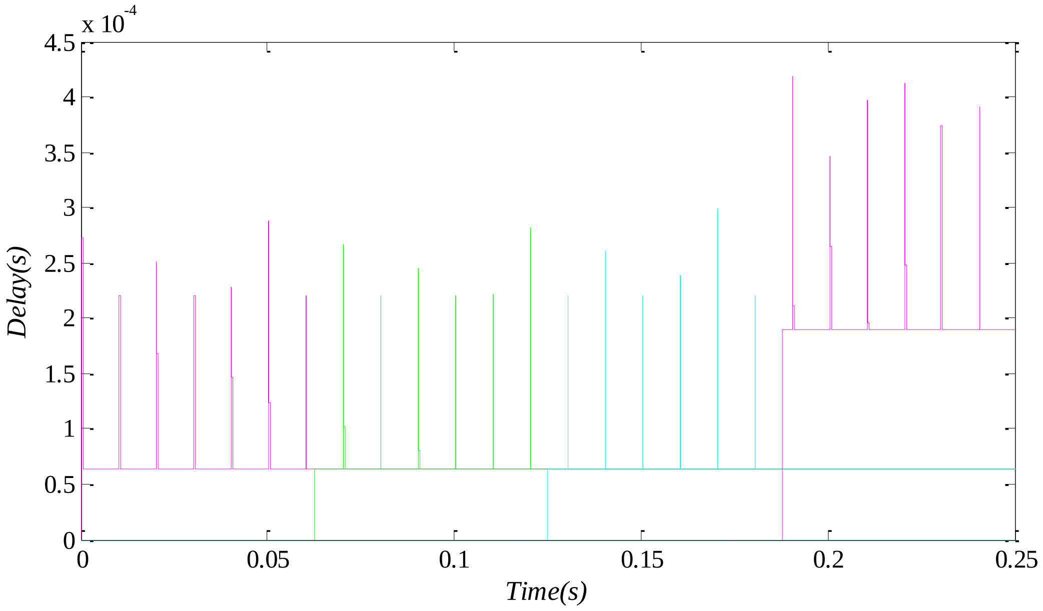


Figure 13. Motor Section 1 Delay-Time Graph (Topology A) (the signals with pink, green, blue, pink colors from left to right between 0s-0.25s belong to motor driver 1, motor driver 2, motor driver 3, motor driver 4, respectively)

In order to observe delay distributions in each motor section, histogram graphs are created. The results are explained in the following paragraphs. Small changes can be observed in each message transmission due to the network traffic and the arbitrary choosing process of TrueTime CAN protocol for two messages with the same priority seek transmission simultaneously as explained in Section 4.

In order to easily compare the results, x and y axes are set to the same ranges in each graph. X axis represents the delays in the range of 0-0.0005s with an increment of 0.00001s and y axis represents the number of the correlated delay in the range of 0-450.

Figure 14 represents the distribution in delay for the motor drivers of motor section 1. For the first three motor drivers, delay amount is intensely at 63µs and for the last motor driver, it is intensely at 189µs. These values are the same as the values expected as 63µs and 189µs for the first three motor drivers and the last motor drivers, respectively.

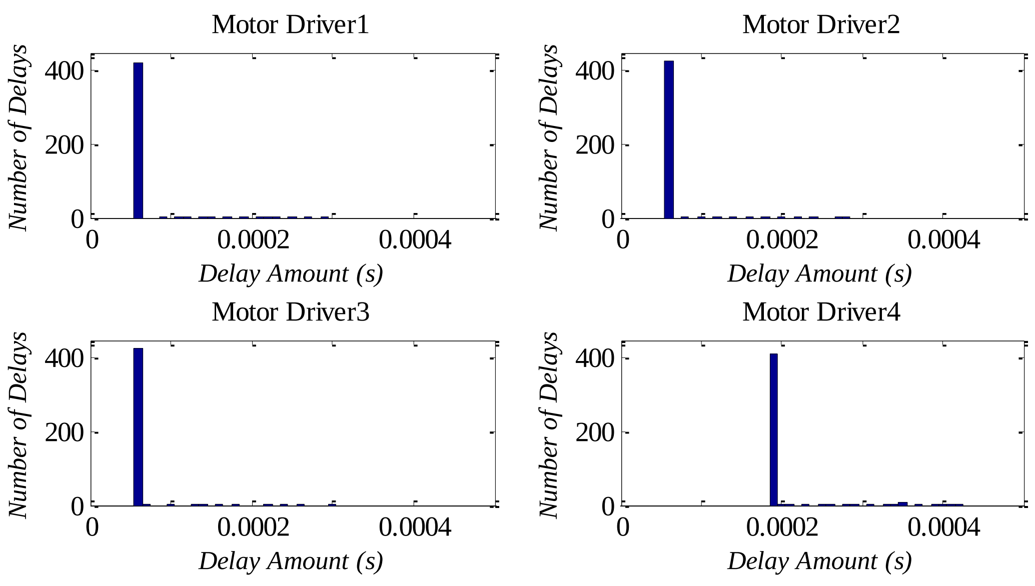


Figure 14. Delay Distribution for Motor Section 1(Topology A)

Figure 15 belongs to motor drivers of motor section 2 in the same experiment. Similar results are observed for motor section 2, as well. For the first three motor drivers, delay amount is intensely at 63µs and for the last motor driver, it is intensely at 189 µs as expected.

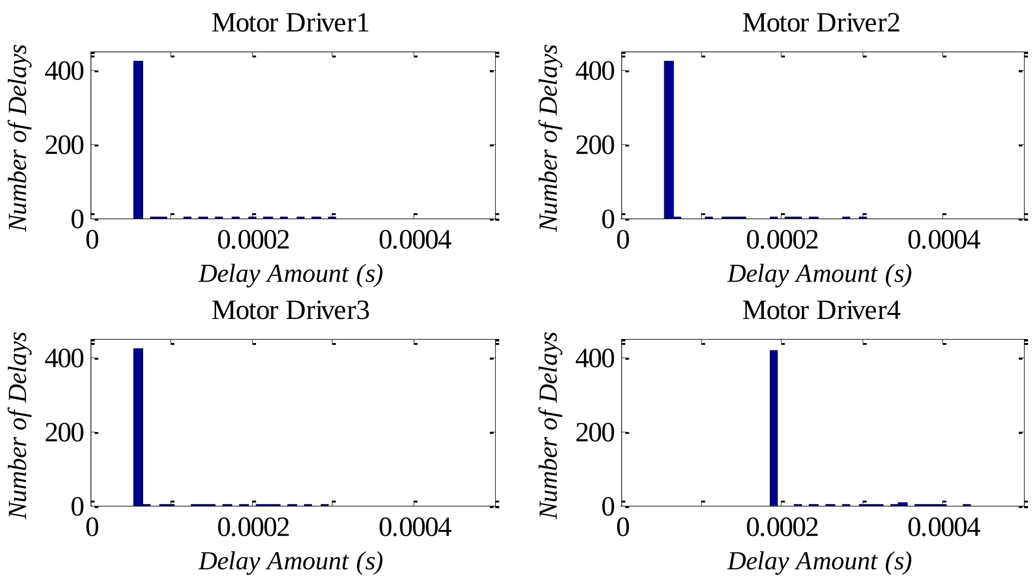


Figure 15. Delay Distribution for Motor Section 2 (Topology A)

The results for motor section 3 are seen in Figure 16. For the first three motor drivers, delay amount is intensely at 63µs and for the last motor driver, it is intensely at 189µs as expected.

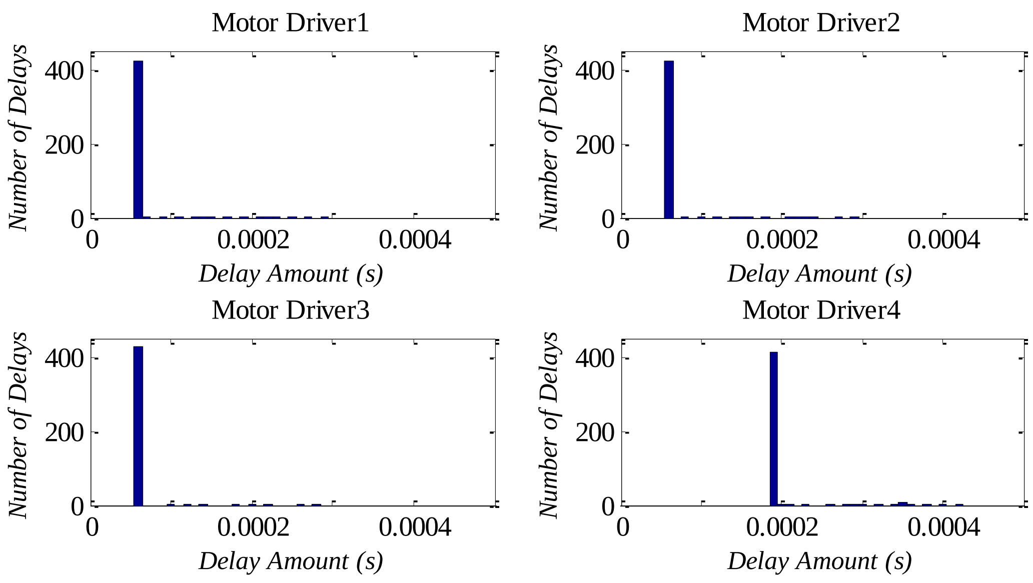


Figure 16. Delay Distribution for Motor Section 3 (Topology A)

The results for motor section 4 are seen in Figure 17. Since this is the last group in the system, MD4 of the motor section 4 does not receive any message in this direction. Hence, we only have 3 graphs for MD1, MD2, MD3 with a delay intensely at 63μs.

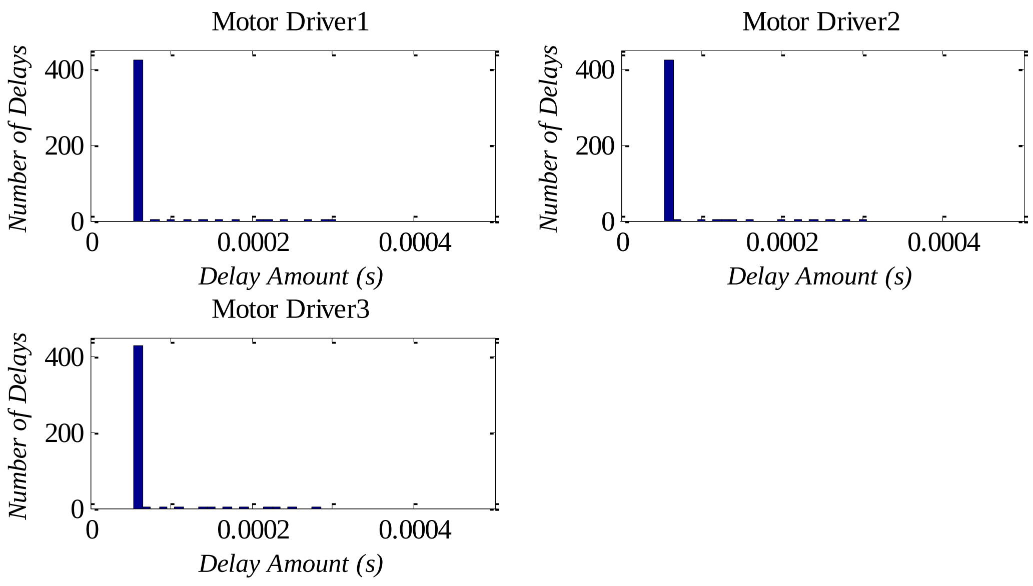


Figure 17. Delay Distribution for Motor Section 4 (Topology A)

In all the results presented in previous paragraphs, different delay amounts are observed due to the network traffic, caused by the periodic messages sent by the main computer via the gateway computers.

We observe that for all motor sections, while the delay amount of the first three motor drivers is mostly 63 µs, the delay amount of the last motor drivers increase to 189 µs with an extra delay of 126 µs. Since these extra delays cause torque fluctuations and vibrations while the mover is passing the motor sections, topology A is not a good choice.

5.2. Topology B

In these simulations, the chosen protocols and the values such as data rate, minimum frame size etc. are the same as topology A. Unlike to Topology A, gateway computers are not used to relay information between motor drivers. So, we expect to decrease the delay occurred in the message transmission between adjacent motor sections. The message transmission between the motor drivers is carried out on either the left or right connection network, whichever is suitable. Therefore, the motor drivers of the adjacent motor sections communicate without any need to gateway computers. The simulation results below show that we have almost the same amount of delays for each node, as expected. Small differences are ignored. This can be obviously seen in the Figure 18 representing the distribution of delay for the motor drivers of motor section 1 with respect to time. It is seen that there is a constant delay at 63 µs, and spikes caused by the messages sent from the main computer to the motor drivers through the gateway computers. The communication between the motor drivers of each motor section in topology B is almost the same as the communication of the first three motor drivers of each motor section in topology A. Because similar to the communication of the first three motor drivers of each section in topology A, all motor drivers of topology B communicate without any need of using the gateway computers. Therefore, the same explanations about the first three motor drivers of each motor sections in topology A are valid for the all motor drivers of topology B.

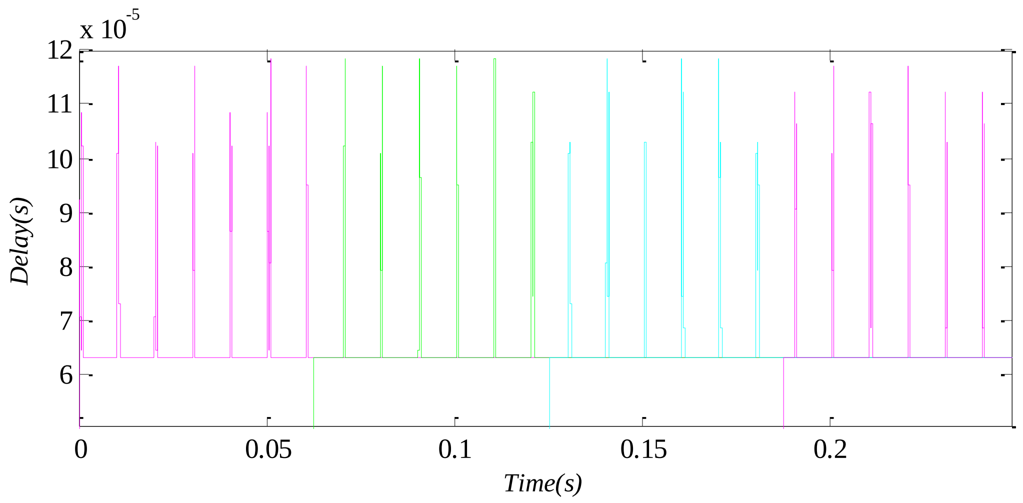


Figure 18. Motor Section 1 Delay-Time Graph (Topology B) (the signals with pink, green, blue, pink colors from left to right between 0s-0.25s belong to motor driver 1, motor driver 2, motor driver 3, motor driver 4, respectively)

Figure 19 represents the delays in the motor drivers of the motor section 1. It is seen that delay amount is intensely at 63 µs for all motor drivers.

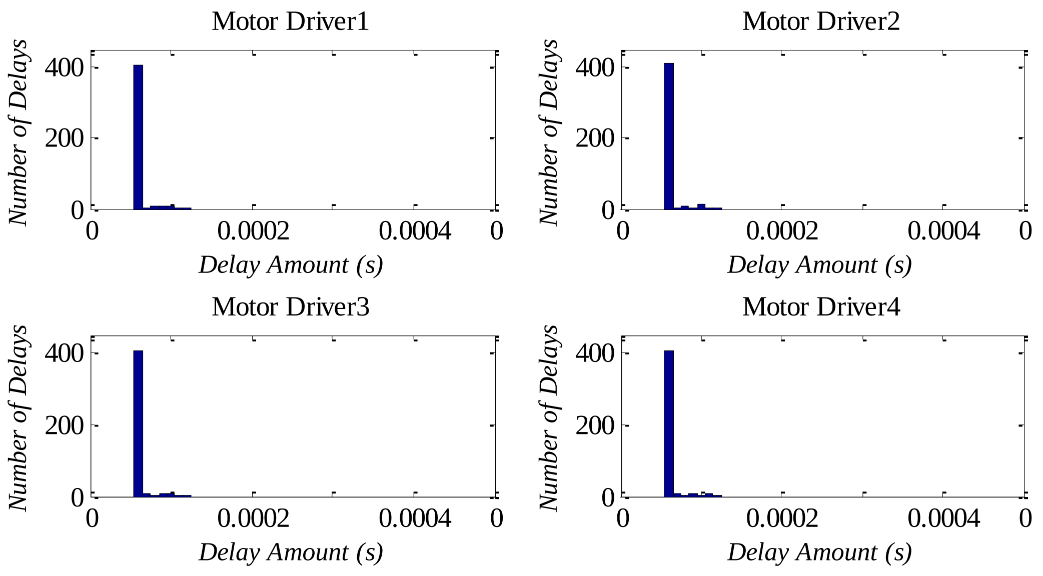


Figure 19. Delay Distribution for Motor Section 1 (Topology B)

Figure 20 represents the delays in the motor drivers of the motor section 2. It is seen that delay amount is intensely at 63 µs for all motor drivers.

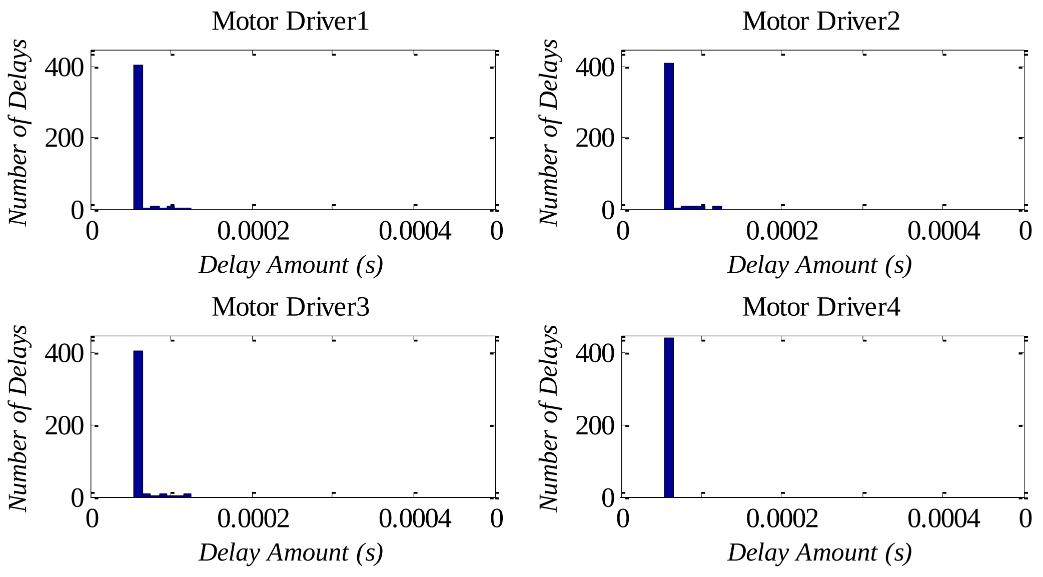


Figure 20. Delay Distribution for Motor Section 2 (Topology B)

Figure 21 represents the delays in the motor drivers of the motor section 3. It is seen that delay amount is intensely at 63 µs for all motor drivers.

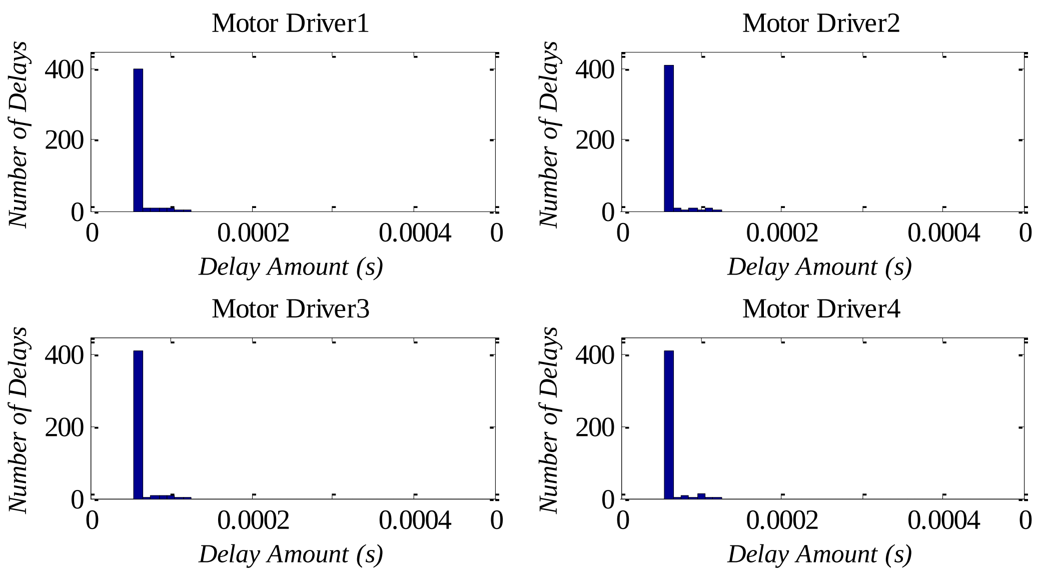


Figure 21. Delay Distribution for Motor Section 3 (Topology B)

Figure 22 represents the delays in the motor drivers of the motor section 4. It is seen that delay amount is intensely at 63 µs for all motor drivers.

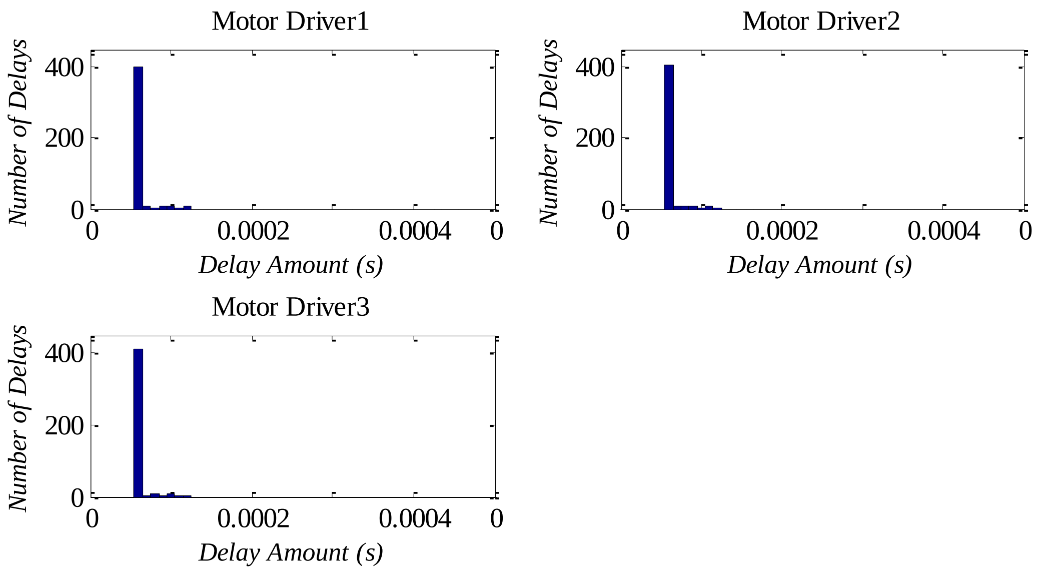


Figure 22. Delay Distribution for Motor Section 4 (Topology B)

The statistical results explained in the previous paragraphs for both topologies indicate that topology B is better than topology A with respect to delay amounts. Due to the structure of topology A, extra delays occur in the last motor drivers of each motor section. These extra delays in the last motor drivers may cause torque fluctuations and vibrations while the mover is passing the motor sections. On the other hand, the structure of topology B enables all motor drivers to have almost the same amount of delays. Thanks to these uniformly distributed delay amounts, the problems such as torque fluctuations and vibrations are not encountered in topology B. Moreover, it has a higher speed since there is no extra delay like topology A. As a result, topology B can be considered as a better choice for the communication of long armature linear motor compared to topology A.

Chapter 6

Conclusions

In this thesis, two different network topologies for the control of long armature linear motor are introduced and simulated, and a general purpose simulation environment was written in Matlab to enable the simulation of other topologies. Simulations were performed using TrueTime, a toolbox for simulating of distributed real-time systems in low level detail. The advantages and disadvantages of both topologies are discussed and their results are compared in terms of communication delay.

In topology A, the communication between the motor drivers of the same motor section can be carried out easily and expeditiously. However, while the mover is passing the borders of the motor sections, the gateway computers must be used to communicate with the motor drivers of the adjacent motor sections. This causes an extra delay in the communication between adjacent motor sections. These extra delays cause torque fluctuations and vibrations while the mover is passing the motor sections.

In topology B, the gateway computers are used only to relay messages from main computer to the motor drivers. It is not required to use the gateway computers to relay messages between the motor drivers of the adjacent motor sections. This structure of topology B enables all motor drivers to have almost the same amount of communication delays. Thanks to these uniformly distributed delay amounts, the problems such as torque fluctuations and vibrations are not encountered in topology B. Therefore, we can recommend topology B for the communication of long armature linear motor. Still, we have spikes in delays caused by the messages sent from the gateway computers carrying the necessary information to manage the mover, sent from the main computer. Since these messages are crucial for the mover to arrive to the correct floor, it is not possible to eliminate them. However, the spikes can be decreased to the lower levels by reducing the amount of these messages sent by the main computer to the sufficiently minimum level.

As another crucial contribution of this thesis, we used a Matlab script based method in order to easily create and modify the simulation models in Simulink and TrueTime toolboxes. For large models, it is nearly impossible to use the standard Matlab graphical user interface based mouse operations to create a model. Since such kind of models consist of a large number of sections and modules that are repeated, we can easily create and modify the repeated blocks from the scripts. Moreover, after creating a Matlab scripts for the model, it is very easy to modify it if it is required. Even enlarging the model is possible with a few changes in the scripts. Moreover, since the changes in the scripts are applied to the all related computers automatically, it is more reliable than the standard Matlab graphical user interface based mouse operations requiring that changes to be accomplished one by one in each node in the system. Although developing such an algorithm may be time consuming in the beginning, it saves time in the future works.

Appendix

Matlab Scripts Creating Simulation Model

```
% Specify the name of the model to create
fname = 'topology_A';

% Introduce the parameters
Outer_Network_No=50000; %outer network number
Num_of_Gate=4;  %number of gateway computers
Num_of_Drivers=4; %number of motor drivers
Const=1000; %max group number can be 1000
Const2=40000; %parameter used for naming the 2nd scope of the Gateway
init_function='initialization';  %initialization function for all Truetime
blocks

% Check if the file already exists and delete it if it does
if exist(fname,'file') == 4
    % If it does then check whether it's open
    if bdIsLoaded(fname)
        % If it is then close it (without saving!)
        close_system(fname,0)
    end
    % delete the file
    delete([fname,'.mdl']);
end

% Create the system
new_system(fname);

% Open truetime library and the created system to copy(add) blocks
open_system('truetime');
open_system('topology_A');

% Add Clock and Display
add_block('built-in/Clock', [gcs,'/Clock'],'Position', [800 20 830 50]);
add_block('built-in/Display',[gcs,'/Display'],'Position', [850 20 900 50]);
% Connect Clock and Display
add_line('topology_A','Clock/1','Display/1');

% Add Gain
add_block('built-in/Gain', [gcs,'/Gain'],'Position', [850 150 880 180],...
    'Gain','4','SampleTime','-1');
% Connect Clock and Gain
add_line('topology_A','Clock/1','Gain/1');

% Add Sum
add_block('built-in/Sum',[gcs,'/','Sum'])
```

```
% Set Sum Block Parameters
set_param([gcs,'/','Sum'],'inputs','++','position',[900,150,930,180])

% Connect Gain and Sum
add_line('topology_A','Gain/1','Sum/1');

% Add Constant
add_block('built-in/Constant', [gcs,'/Constant'],'Position', ...
    [850 250 880 280],'Value','0.25','SampleTime','inf');

% Connect First Output of Constant Block to Second Input of Sum Block
add_line('topology_A','Constant/1','Sum/2');

% Generate OUTER NETWORK BLOCK (Set Block Parameters, Add Scope, Connect
Blocks)
add_block('truetime/TrueTime Network','topology_A/Outer Network',...
    'Position', [10 (20) 70 (70)]);
set_param('topology_A/Outer Network','nnodes', num2str(Num_of_Gate+1), ...
    'nwnbr',num2str(Outer_Network_No),'nwtype','CSMA/CD (Ethernet)', ...
    'rate','10000000','minsize','1144');
add_block('built-in/Scope', [gcs,['/Scope' num2str(Outer_Network_No)]],...
    'Position', [100 (10) 130 (40)],'LimitDataPoints','off');
add_line('topology_A','Outer Network/1',...
    ['Scope' num2str(Outer_Network_No) '/1']);


% Generate MAIN COMPUTER BLOCK (Set Block Parameters, Add Sine Wave Block,
Add Scope, Connect Blocks)
add_block('built-in/Sin', [gcs,'/Sine Wave'],...
    'Position', [10 120 30 140],'SampleTime','0' );
add_block('truetime/TrueTime Kernel', 'topology_A/Main Computer',...
    'Position', [60 110 120 160]);
set_param('topology_A/Main Computer','sfun',init_function,'args', ...
    ['[' num2str(Outer_Network_No) ']'],'ninputsoutputs','[1 1]',...
    'nwnodenbr',['[',num2str(Outer_Network_No) ,' ', ...
    num2str(Num_of_Gate+1),']']);
add_line('topology_A','Sine Wave/1','Main Computer/1' );
add_block('built-in/Scope', [gcs,['/Scope'num2str(Outer_Network_No+1)]],...
    'Position', [160 120 190 150],'LimitDataPoints','off');
add_line('topology_A','Main Computer/2', ...
    ['Scope'num2str(Outer_Network_No+1) '/1']);

% Nested for loop is used to create Driver Networks,Gateway Networks,Motor
% Drivers and Gateway Computers.if-else statement is necessary for
% satisfying the conditions related to different blocks.
for j=0:1:(Num_of_Gate-1)
    Network_ID=j+1;  %network number
    Gate_ID=j*11+1;  %used for naming Gateway Computers

    % Generate DRIVER NETWORKS (Set Block Parameters, Add Scope, Connect
Blocks)
    add_block('truetime/TrueTime Network',['topology_A/Driver Network'...
        num2str(Network_ID)],'Position',[210 (400+900*j) 270 (450+900*j)]);
    set_param(['topology_A/Driver Network' num2str(Network_ID)],...
        'nnodes',num2str(Num_of_Drivers+1),'nwnbr',num2str(Network_ID), ...
        'nwtype','CSMA/AMP (CAN)','rate','1000000','minsize','60');
    add_block('built-in/Scope', [gcs,['/Scope' num2str((j+1)*1000)]],...
        'Position',[310 (390+900*j) 330 (410+900*j)],'LimitDataPoints', ...
        'off');
    add_line('topology_A',['Driver Network' num2str(Network_ID) '/1'],...
        ['Scope' num2str((j+1)*1000) '/1']);


    % Generate GATEWAY NETWORKS (Set Block Parameters, Add Scope, Connect
Blocks)
    if j<(Num_of_Gate-1)
        add_block('truetime/TrueTime Network', ...
            ['topology_A/Gateway Network'num2str(Network_ID)],'Position',
            [300 (850+900*j) 360 (900+900*j)]);
        set_param(['topology_A/Gateway Network' num2str(Network_ID)],...
            'nnodes',num2str(2),'nwnbr',num2str(Const+Network_ID), ...
            'nwtype','CSMA/AMP (CAN)','rate','1000000','minsize','60');
        add_block('built-in/Scope', [gcs, ...
            ['/Scope' num2str(Const+Network_ID)]],'Position', ...
            [400 (840+900*j) 420 (860+900*j)],'LimitDataPoints','off');
        add_line('topology_A',['Gateway Network' num2str(Network_ID) '/1'],
            ['Scope' num2str(Const+Network_ID) '/1']);
    end


    % Generate GATEWAY(GW) COMPUTERS
    add_block('truetime/TrueTime Kernel', ['topology_A/Gateway Computer'...
        num2str(Network_ID)],'Position', ...
        [(370) (400+900*j) (430) (450+900*j)]);

    % Set Block Parameters of First GW
    if j==0
        set_param(['topology_A/Gateway Computer' num2str(Network_ID)],...
            'sfun',init_function,'args',['[' num2str(Network_ID*Const+1)
            ']'],'ninputsoutputs','[0 9]',...
            ['[', num2str(Network_ID),' ', '1',';' ...
            num2str(Outer_Network_No),' ', '1',';' ...
            num2str(Network_ID*Const+1),' ', '1]']);
    end

    % Set Block Parameters of Inner GWs
    if j>0 && j<(Num_of_Gate-1)
        set_param(['topology_A/Gateway Computer' num2str(Network_ID)],...
            'sfun',init_function,'args',['[' num2str(Network_ID*Const+1)
            ']'],'ninputsoutputs','[0 9]',...
            'nwnodenbr',['[', num2str(Network_ID),' ', '1' , ';' ...
             num2str(Outer_Network_No),' ', num2str(Network_ID), ';'...
             num2str(Const+Network_ID-1),' ', '2', ';'...
             num2str(Const+Network_ID),' ','1',']']);
    end

    % Set Block Parameters of Last GW
    if j>0 && j==(Num_of_Gate-1)
        set_param(['topology_A/Gateway Computer' num2str(Network_ID)],...
            'sfun',init_function,'args',['[' num2str(Network_ID*Const+1)
            ']'],'ninputsoutputs','[0 9]',...
            'nwnodenbr',['[', num2str(Network_ID),' ', '1' , ';' ...
            num2str(Outer_Network_No),' ', num2str(Network_ID), ';' ...
            num2str(Const+Network_ID-1),' ', '2]']);
    end

    % Add Scope, Connect Blocks
    add_block('built-in/Scope', [gcs,['/Scope' num2str(Gate_ID)]],...
        'Position', [(470) (360+900*j) (490) (380+900*j)], ...
        'LimitDataPoints','off');
    add_line('topology_A',['Gateway Computer' num2str(Network_ID) '/1'],
        ['Scope' num2str(Gate_ID) '/1']);

    add_block('built-in/Scope', [gcs,['/Scope' num2str(Const2+Gate_ID)]],
        'Position', [(470) (440+900*j) (490) (460+900*j)], ...
        'LimitDataPoints','off');
    add_line('topology_A',['Gateway Computer' num2str(Network_ID) '/2'], ..
        ['Scope' num2str(Const2+Gate_ID) '/1']);

    % Add Mux

    %Mux 1
    add_block('built-in/Mux',[gcs,'/Mux' num2str(Network_ID)],...
        'orientation','right','inputs',num2str(Num_of_Drivers),...
        'position',[1050 (150+80*((Num_of_Drivers/2)-1)+900*j) 1100
        (180+80*((Num_of_Drivers/2)-1)+900*j)]);
    add_block('built-in/Scope',[gcs,['/Scope' num2str(Const*Const+
        Network_ID)]],'Position',[1150 (150+80*((Num_of_Drivers/2)-1)
        +900*j) 1200 (180+80*((Num_of_Drivers/2)-1)+900*j)] ...
        ,'LimitDataPoints','off');
    add_line('topology_A',['Mux' num2str(Network_ID) '/1'],['Scope'
        num2str(Const*Const+Network_ID) '/1'  ]);

    %Mux 2
    add_block('built-in/Mux',[gcs,'/Mux' num2str(Const*Network_ID)],...
        'orientation','right',...
        'inputs',num2str(Num_of_Drivers),...
        'position',[1050 (300+80*((Num_of_Drivers/2)-1)+900*j) 1100
        (330+80*((Num_of_Drivers/2)-1)+900*j)]);
    add_block('built-in/Scope',[gcs, ...
        ['/Scope'num2str(Const2*Const+Network_ID)]] ,...
        'Position', [1150 (300+80*((Num_of_Drivers/2)-1)+900*j) 1200
        (330+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');
    add_line('topology_A',['Mux' num2str(Const*Network_ID) '/1'], ...
        ['Scope' num2str(Const2*Const+Network_ID) '/1'  ]);

    % Generate MOTOR DRIVERS (Set Block Parameters, Add Scope, Connect
Blocks)
    for i=1:1:(Num_of_Drivers)

        Driver_ID=j*11+i+1; %used for naming Motor Drivers

        add_block('truetime/TrueTime Kernel', ['topology_A/Motor Driver'...
            num2str(Network_ID*Const+i)],...
            'Position', [(570 ) (20+80*(i-1)+900*j) (590 ) ...
            (60+80*(i-1)+900*j)]);
        set_param(['topology_A/Motor Driver' num2str(Network_ID*Const+i)],
            'sfun',init_function,'args',['[' num2str(Network_ID*Const+i+1)
            ']'],'ninputsoutputs','[1 4]','nwnodenbr',['[', num2str(j+1),
            ' ', num2str(i+1) ']']);
        add_block('built-in/Scope', [gcs,['/Scope' num2str(Driver_ID)]],...
            'Position', [(650 ) (10+80*(i-1)+900*j) (680)
            (40+80*(i-1)+900*j)],'LimitDataPoints','off');

        add_line('topology_A',['Motor Driver' num2str(Network_ID*Const+i)
            '/1'],['Scope' num2str(Driver_ID) '/1']);
        add_block('built-in/Scope', [gcs,['/Scope'
            num2str(Const2+Driver_ID)]], 'Position',...
            [(650 ) (50+80*(i-1)+900*j) (680) (80+80*(i-1)+900*j)], ...
            'LimitDataPoints','off');

        add_line('topology_A',['Motor Driver' num2str(Network_ID*Const+i)
            '/2'] , ['Scope' num2str(Const2+Driver_ID) '/1']);
        add_line('topology_A','Sum/1',['Motor Driver' ...
            num2str(Network_ID*Const+i) '/1']);

        %mux1
        add_line('topology_A',['Motor Driver' num2str(Network_ID*Const+i)
            '/1'],['Mux' num2str(Network_ID) '/' num2str(i) ]);

        %mux2
        add_line('topology_A',['Motor Driver' num2str(Network_ID*Const+i)
            '/2'],['Mux' num2str(Const*Network_ID) '/' num2str(i) ]);
    end
 end

% Set a couple of model parameters to eliminate warning messages
set_param(gcs,'StartTime','0.0');
set_param(gcs,'StopTime','2.0');
set_param(gcs,'Solver','ode45');

% Save the model
save_system(fname);

% Open the model
uiopen('C:\Program Files\MATLAB\R2012a\truetime-2.0-beta7\ examples\ kubra\
TTcreate_block_set_param\120513\7ekim-AO(timestamp)(19kasim)\
topology_A.mdl',1)
```

Bibliography

[1] W.D. Jones. How to build a mile-high skyscraper. *Spectrum, IEEE,* 44(6):52-53, June 2007.

[2] T. Ishii. Elevators for skyscrapers. *Spectrum, IEEE,* 31(9):42-46, Sep 1994

[3] Kita H. Markon, S. and H. Kise. *Control of Traffic Systems in Buildings:Applications of Modern Supervisory and Optimal Control (Advances in Industrial Control).* Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006

[4] L. Farkas, J. Hnat. Simulation of Networked Control Systems Using TrueTime. In *Mezinarodni Conference Technical Computing,* Prague, 2009.

[5] G. Nicolescu, P. J. Mosterman. TrueTime: Simulation Tool for Performance Analysis Real-Time Embedded Systems. In *Model-Based Design for Embedded Systems*, CRC Press, November 2009.

[6] D. Henriksson, A. Cervin, K-E Årzén. TrueTime: Real-time Control System Simulation with MATLAB/Simulink In *Proceedings of the Nordic MATLAB Conference*, Copenhagen, Denmark, October 2003.

[7] Kita H. Suzuki H. Sudo T. Takahashi, S. ans S. Markon. Simulation-based optimization of a controller for multi-car elevators using a genetic algorithm for noisy fitness function. *The 2003 Congress on Evolutionary Computation, CEC '03*, 3:1582-1587 Vol.3, Dec. 2003

[8] Koseki T. Miyatake, M. and S. Sone. Design and traffic control of multiple cars for an elevator system driven by linear synchronous motors.In *Proc. Symposium on Linear Drives for Industry Applications*, number AP-22, pages 94-97, 1998.

[9] F. Hanssen , P. G. Jansen. Real-Time Communication Protocols: An Overview (2003) [online]. Available at http://eprints.eemcs.utwente.nl/862/01/000000df.pdf

[10] Koseki T. Miyatake, M. and S. Sone. Scheduling for high density transport in ropeless lift systems using multiple cars with transferability between shafts. *Computers in Railways*, pages 375-384, Aug. 1996

[11] Koseki T. Miyatake, M. and S. Sone. A proposal of a ropeless lift system and evaluation of its feasibility,. *IEEJ Trans. IA*, 119(11):1353-1360, 1999

[12] Osawa Watanable T. Yamaguchi, H. and H. Yamada. Brake control characteristics of a linear synchronous motor for roplesess elevator. *IEEE Trans. On Magnetics*, 37(5):3732-3736, 2001.

[13] Lorenz R.D. Jansen P.L. Transducerless position and velocity estimation in induction and salient ac machines. *IEEE INDUSTRY APPLICATION*, 31(2):240-247, 1995.

[14] Testa A. Consili A., Scarcella G. Sensorless control of ac motors at zero speed. In *IEEE ISIE*, 1999.

[15] M. Mihalachi and P. Mutschler. Position acquisition for long primary linear drives with passive vehicles. In *Industry Applications Society Annual Meeting, 2008. IAS '08*. IEEE, pages 1-8, 2008.

[16] M. Mihalachi and P. Mutschler. Position acquisition for linear drives a comparison of optical and capacitive sensors. In *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, pages 2998-3005, 2008.

[17] Phong C. Khong, Roberto Leidhold, and Peter Mutschler. Magnetic guiding and capacitive sensing for a passive vehicle of a long-primary linear motor. In *Power Electronics and Motion Control Conference (EPE/PEMC), 2010 14th International*, pages S3-1-S3-8, 2010.

[18] M. Mihalachi and P. Mutschler. Capacitive sensors for position acquisition of linear drives with passive vehicles. In *Optimization of Electrical and Electronic Equipment (OPTIM), 2010 12th International Conference on*, pages 673-680, May 2010.

[19] S. Markon, A. Onat, E. Kazan, and C. Gurbuz. Linear motor coils as position sensors. In *Linear Drives for Industry Applications, Proc. LDIA*, 2009.

[20] C. Gurbuz, E. Kazan, A. Onat and S. Markon, Lineer motorlu asansörler için güvenilir sürüş yöntemleri, In *Automatic Control International Congress 2009 (TOK'09*), Istanbul, Turkey, October 2009.

[21] Broadcast Communication Networks. In *National Programme on Technology Enhanced Learning (NPTEL)* [online]. June 2013. Available at http://nptel.iitk.ac.in/ courses/ Webcourse-contents IIT/ %20Kharagpur/Computer%20networks/pdf/

[22] Carrier Sense Multiple Access [online]. December 2013. Available at http://en. wikipedia. org /wiki/Carrier_sense_multiple_access>

[23] C. M. Krishna, K. G. Shin. *Real Time Systems,* McGraw-Hill Higher Education, New edition, May 1997.

[24] Javvin Company. In *Protocol Dictionary* [online]. Available at http://www.javvin.com/ protocolToken.html

[25] *Renesas Electronics Corporation* [online]. April 2010. Available at http:// documentation.renesas. com/doc/products/mpumcu/apn/rej05b0804_m16cap.pdf

[26] D. Henriksson. TrueTime Simulation of Networked Computer Control Systems. In preprints of *2nd IFAC Conference on Analysis and Design of Hybrid Systems*. Alghero, Italy, 7-9 June 2006.

[27] M. Andersson. Simulation of Wireless Networked Control Systems. In *44th IEEE Conference on Decision and Control, and the European Control Conference 2005*. Seville, Spain, December 12-15, 2005,476-481.

[28] S. Cojocaru, C. Radoi, S. Stancescu. The Analysis of CAN and Ethernet in Distributed Real-Time Systems. In *U.P.B. Scientific Bulletin* Series C, Vol. 71, Iss. 4, 2009 ISSN 1454-234x

[29] A. Cervin, M. Ohlin, D. Henriksson. Simulation of Networked Control Systems Using TrueTime. In *3rd International Workshop on Networked Control Systems*. Nancy, France, 2007

[30] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K. Arzen. How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime. In *Control Systems*, *IEEE* 23(3):16-30.

[31] K. Etschberger. Comparing CAN and Ethernet Based Communication. In *IXXAT Automation GmbH* [online]. November 2009 Available at http://www.ixxat.com/download/artikel_ comparison_can_and_ethernet.pdf.

[32] D. Henriksson, O. Redell, J. El-Khoury, A. Cervin, M. Törngren, K. Arzen. Tools For Real Time Control Systems Co-Design. In H. Hansson (ed.), *ARTES- A Network For Real Time Research and Graduate Education in Sweden 1997-2006*, Department of Information Technology, Uppsala University. Sweden, 2006.

[33] A. Cervin, D. Henriksson, M. Ohlin. *TrueTime 2.0 Beta Reference Manual*, Department of Automatic Control, Lund University, June, 2010