VEHICLE HIGHWAY AUTOMATION

by

DINESH KUMAR CHALLA

B.Tech., Jawaharlal Technological University, 2003

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2009

Approved by:

Major Professor
Dr. Gurdip Singh

Abstract

Vehicle Highway Automation has been studied for several years but a practical system has not been possible because of technology limitations. New advances in sensing and communication technology have now brought a realistic system within reach. This paper proposes a Co-Operative Vehicle Highway Automation System for automating traffic information gathering and decision making in a vehicle on a highway which is cost effective and near to real life implementation. Co-Operative Vehicle-Highway Automation System is the system which is implemented by technology on-board a vehicle and also on the intelligent infrastructure technology along a highway. Vehicle Automation, Collision Prevention and Avoidance, Route Guidance, Highway Information System, Vehicle tracking, and Traffic surveillance are some applications which can be implemented in the Co-Operative Vehicle-Highway Automation System. Implementing Vehicle Highway Automation System will provide an ameliorated level of road transportation. The possible benefits to society and individuals are many in terms of time, safety, comfort and overall travel quality.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to start with thanking Dr. Gurdip Singh for his unparalleled support and encouragement towards my Masters Degree Program at Kansas State University. His courses laid the foundation of my knowledge and have been greatest motivation factors towards my research. I am grateful to him for giving me an opportunity to work with him. Research meetings with him are always a good learning experience and exciting. He has always given me time for discussions from which I have learned the approach to analyze, understand and solve complicated problems. I am thankful to him for his excellent mentorship in terms of research guidance, academic advising and moral support.

I would like to thank my committee members Dr. Daniel Andresen and Dr. Masaaki Mizuno for their timely support and invaluable help. I would also like to thank them for reviewing my thesis and offering important feedback and suggestions.

I would like to thank my research group members Sumeet Gujrati, Sandeep Pulluri and Samuel Benny for sharing their knowledge and for their continuous support towards my research work. I would also like to thank Pavani Ayyagari and all my friends for their unflinching moral support.

I would like to express special thanks to my parents for their blessings and good wishes, and also to my brothers for their undying love and support all the way through.

I would like to thank everyone who helped me accomplish my tasks and achieve my goals towards completing my graduate degree program.

Dedication

I am indebted to my parents, Mr. Ramesh Challa and Mrs. Mohini Challa, for everything they have given me. They taught me the value of knowledge, the joy of love and the importance of family. Without the eternal support of my parents, I would not have come this far in my education and life. To them I dedicate this thesis.

I also dedicate it to my brothers, Mr. Deepak Challa and Mr. Dilip Challa, have always supported me in both professional and personal aspects of my life. This thesis would not be complete without dedicating it to them as an acknowledgement for their faith in me.

CHAPTER 1 - Introduction

This paper aims to develop Vehicle Highway Automation System (VHAS) to automate information gathering and decision making on a vehicle on a highway, which will ameliorate the present road transportation system. There are many benefits to general public in terms of time, safety, comfort and overall travel quality.

Vehicle-Highway Automation system will reduce the overall travel time. It will also increase the throughput of highway by accommodating more vehicles on the highway. This will in turn reduce the necessity of increasing the size of the highway.

VHAS will also increase the safety of road transportation. The usage of advanced sensing and communication technology will provide a system which is reliable and consistent. When this system is implemented carefully, this eventually will decrease the number of accidents on a highway.

1.1 Types of Vehicle Highway Automation Systems

Vehicle – highway automation systems can be implemented in two different ways. They are Autonomous Vehicle-Highway Automation System and Co-Operative Vehicle-Highway Automation System.

1.1.1 Autonomous Vehicle Highway Automation Systems

Autonomous Vehicle Highway automation system can be implemented by technology that is entirely contained within the vehicle. In other words, this system does not use any assistance from technologies outside the vehicle. Implementing Vehicle – Highway Automation system in this way will lead to cost effectiveness but it has several critical drawbacks. Autonomous Vehicle Highway Automation system can provide vehicle automation, collision avoidance but cannot provide route guidance, collision prevention, highway information system, vehicle tracking, traffic surveillance, etc. Even though this system is cost effective to implement, it is not the ideal or safe way to implement because of its critical draw backs.

1.1.2 Co-Operative Vehicle Highway Automation System

Co-Operative Vehicle Highway Automation System is implemented by technology within the vehicle and is also implemented by the intelligent infrastructure technology outside the vehicle. Highways of the future will utilize intelligent road side infrastructure which interact with the vehicles. Co-Operative Vehicle Highway Automation System (CVHAS) will provide reduced and reliable travel times and increase safety of road transportation. CVHAS has the potential for significant safety and capacity improvements. Road Diversions and maintenance programs of highway can be implemented easily with this system. Above all, implementing Vehicle Highway Automation System co-operatively has several benefits in terms of safety, travel time, comfort and journey quality.

1.2 Implementing the Co-Operative Vehicle Highway Automation System

Vehicle Highway Automation has been studied for several years but a practical system has not been possible because of technology limitations. New advances in sensing and communication technology have now brought a realistic system within reach. This paper proposes a Co-operative Vehicle Highway Automation System that is more cost effective and near to real life implementation.

1.2.1 Using Wireless Sensor Network

A wireless sensor network (WSN) consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion, etc,. In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communication device, a microcontroller, and an energy source, usually a battery. A sensor node might vary in size from that of a shoebox down to the size of a grain of dust, although functioning “motes” of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from hundreds of dollars to a few pence, depending on the size of the sensor network and complexity required of individual sensor nodes.

This implementation of vehicle highway automation system uses wireless Sensor networks to create road side infrastructure to form Intelligent Highway system. Having Wireless Sensor network technology for creating Intelligent Highway system has several benefits. It is economical to implement them on real highways, easy and can be implemented on existing highways, reliable and does not take much space.

Sensor nodes of WSN are placed on the side of a highway for every certain distance. These Sensor nodes have sensors which can sense and store different kinds of things around them. These sensor nodes can communicate among themselves and also with other devices. By communicating they form a network by which they can exchange all kind of information directly or by multi hop. They can also connect and communicate with the Global Intelligent highway system servers through base stations. One base station is installed for every ‘n’ nodes. The value of n depends up on the amount of traffic and usage of the highway.

Even the vehicles using this intelligent highway system are installed with sensors and communicating devices. With the help of these sensors and communicating devices, vehicles can sense things around them and communicate with the WSN and also with other vehicles. As vehicles on the highway can interact with WSN, these vehicles can be automated to go to their required destination safely and efficiently by sensing and by exchanging the required necessary information between them. The information that is known to WSN is updated into the Global Intelligent highway servers through base stations which enable every sensor node to have the information of the global scenario.

1.3 Vehicle – Highway Automation System Characteristics

Implementing Vehicle Highway Automation system in this way will have many features/characteristics. Vehicle Automation, Collision Prevention and Avoidance, Route Guidance, Highway Information System, Vehicle tracking, Traffic surveillance are some of them.

1.3.1 Vehicle Automation

Fully Automation of vehicle is possible by using Intelligent Highway System. That is, a vehicle can drive automatically without any human driver interference to the destination with safety and with the minimum time possible. The vehicle uses several types of devices to sense its environment such as visual sensors, infrared sensors, laser sensors, accelerometers, magnetometers, etc. Each vehicle has a powerful computer to process sensory data. With the help of these sensors, the vehicle can track the road and drive on a particular lane, follow another vehicle, change the lane, avoid collision. The vehicle also contains communication devices by which it can communicate with the road side infrastructure (i.e. WSN). It can also process the information that comes from WSN. Planned routes to destination, needed speed fluctuations, information about unsafe vehicle operation, details of traffic ahead are some of the information that vehicle can get from the WSN. With this information and with the sensory data from vehicle, the automation of vehicle is done.

1.3.2 Collision Prevention and Avoidance

Collision prevention is one of the important features that is necessary to ensure safety. Collision prevention can be implemented with the help of roadside infrastructure (i.e. WSN). Every vehicle running on the Intelligent Highway communicates (sends and receives information) with the WSN (i.e. road side infrastructure). The vehicle sends information to the WSN such as the speed of the vehicle, lane of the vehicle, direction, etc. Because of this WSN has information of every vehicle on that highway. WSN can instruct the vehicle to decrease its speed or change its lane if there is a vehicle moving with lesser speed in front of it. If there is an intersection and if two vehicles are approaching to the intersection at the same time, then WSN may instruct one vehicle to stop or to change to other road. In this way WSN can inform vehicles before in time about the incoming traffic and hence can prevent collision.

One of the safety features of Intelligent Highway System is Collision avoidance. Collision avoidance can be implemented with the help of sensor devices inside the vehicle. A vehicle has several sensors such as visual sensor, infrared sensor, proximity sensor, ground sensor, magnetometer, accelerometer etc. With these sensors, the vehicle can detect any unsafe situation and act accordingly.

1.3.3 Route Guidance

Route guidance is an important feature in automating a vehicle to reach its destination safely and with a minimum time. Route guidance can be implemented with the help of WSN. The vehicle asks WSN for the route guidance to the destination. WSN gets the request and processes the request by first finding the shortest route with the least possible time i.e. avoiding the traffic congestions. After calculating the shortest route to destination, the sensor node of WSN sends a message to all the sensor nodes in that path to get the information of traffic and highway. With this information, the sensor node plans the route accordingly. By using this planned route the vehicle is guided to the destination by the sensor nodes of WSN. Thus route guidance helps coordinate the traffic flow efficiently, reduce speed fluctuations, maximize highway capacity and minimize avoidable traffic congestion.

1.3.4 Vehicle Tracking

Tracking of a vehicle is possible by implementing intelligent highway system. Every vehicle using intelligent highway system communicates with the sensor nodes of WSN in regular intervals. Whenever a sensor node gets information such as vehicle id, position, speed etc from a vehicle, it sends the information to the global intelligent highway system servers through the base station. The server updates the information of the vehicle. In this way location of the vehicle can be determined by getting the information from the server.

1.3.5 Traffic Surveillance

Traffic surveillance is one of the features of intelligent highway system that can be implemented. With the help of this feature we can monitor the traffic, information about the traffic congestion, information regarding road diversions, accidents, etc. Traffic surveillance is implemented with the help of wireless sensor network (WSN). Every sensor node of WSN sends information about the environment surrounding it to the global servers through base station. Thus traffic surveillance is implemented by getting data from these servers.

CHAPTER 2 - Implementation of Vehicle-Highway Automation System on a Test Bed

In this paper, the Vehicle-Highway Automation System (VHAS) is implemented on a test bed which was created to simulate a real life Co-operative VHAS in a smaller environment. The test bed consists of two main parts. The first part is to simulate highway scenario in a smaller environment. The second part is to create intelligent highway infrastructure system for the first part of the system. This chapter describes different kinds of components used in the test bed and also how these components are used.

2.1 Components

The test bed is a combination of several different components. E-puck, Telosb mote, IpaQ, USB hub, USB cable and Plexi Glass board are some of the components. The following data describes these components.

2.1.1 E-puck

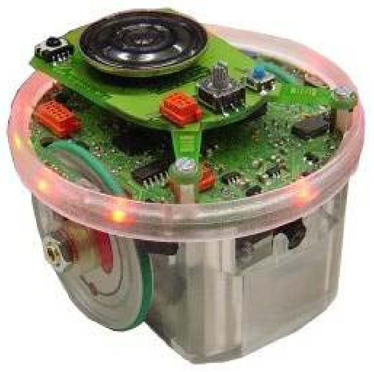

Figure 2-1 E-puck Mobile Robot

E-puck is a small differential wheeled mobile robot. It is powered by 16-bit dsPIC processor and features a large number of sensors. The e-puck hardware and software is fully open source providing low level access to every electronic device and offering unlimited extension possibilities.

E-puck robots are used to simulate a car that can be automated on a test bed. It has several sensors such as proximity sensors, ambient light sensors, 3D accelerometer, 3 Omni directional microphones (i.e. sound sensors), and color camera. With the help of these sensors it can sense the environment around it and get sensory data. E-puck also has different types of communication mechanisms through which it can communicate with other e-pucks and also with the intelligent wireless sensor network.

The sensors of an e-puck robot were used in different ways in the development of VHAS. An e-puck robot has eight proximity sensors surrounding it. These eight proximity sensors are used to detect any object near the e-puck and hence can detect and avoid collision. Ground sensor can be added as an extension in front of the e-puck, which comprises of three proximity sensors directly pointed to the ground. The ground sensor can be used to detect and follow the road on a highway and can also be used for fall-detection. It has a color camera which can also be used to detect, identify objects and also follow the road.

Bluetooth and Wireless Radio communication are the two types of communication mechanisms available in an e-puck. E-puck uses Bluetooth to communicate or exchange information with the other e-pucks on the highway. Communication through Wireless Radio is possible by mounting Zigbee communication turret on top of the e-puck. Wireless Radio is used to communicate or exchange information with the sensor nodes of intelligent WSN.

3.1.2 TelosB Sensor Motes

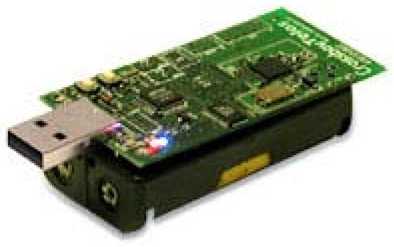

Figure 2-2 TelosB Mote

A sensor node, also known as a 'mote', is a node in a wireless sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network. TelosB sensor mote is one type of sensor node in a wireless sensor network. TelosB is an open-source experimental platform developed and published to the community by UC Berkeley. The IEEE 802.15.4 compliant device for wireless mesh networking features the TI MSP430 microcontroller. The platform provides an integrated processor radio solution including a USB interface, 2.4GHz radio, onboard antenna, sensor interfaces, and an optional pre-installed environmental sensor suite including light, temperature and humidity sensor.

A TelosB mote uses TinyOS operating system which is an open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks. nesC (network embedded systems C) is a component based, event driven programming language used to build applications for the TinyOS platform. nesC is built as an extension to the C programming language with components wired together to run applications on TinyOS.

3.1.3 IpaQ

Figure 2-3 IpaQ

IpaQ is a hand held pocket pc device which is used as a base station in Intelligent Highway System. One IpaQ which is acting as a base station is installed for every ‘n’ sensor nodes. These base stations can collect information or data from the sensor nodes and send the same data to the global server using internet. This global server gets and updates the data in the database. Global server can also communicate to sensor nodes using base station.

IpaQ is a powerful hand held device with a 520 MHz Intel PXA270 processor. IpaQ can connect to the internet through built in WiFi. This feature can be used to connect to the global server and send/receive information from it. IpaQ has Bluetooth wireless technology integrated into it. Additional features can be added to the ipaQ with the help of ports and slots provided in it.

3.1.4 Other Components

Figure 2-4 USB Hub

Figure 2-5 USB Cable

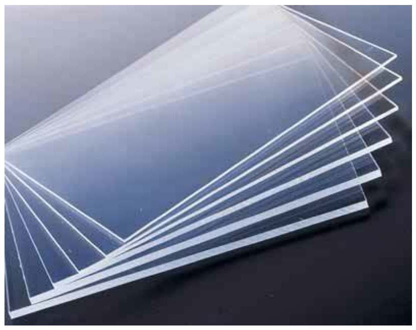

Figure 2-6 Plexi Glass Board

Figure 2-7 Velcro

USB Hub, USB Cable, Plexi Glass Board and Velcro are the various other hardware components used to create a Test Bed for the Vehicle-Highway Automation.

The Test Bed uses four Plexi glass boards for creating Intelligent Highway infrastructure and each board consists of eight TelosB Motes, which are thirty two TelosB motes in total. These TelosB motes are powered through USB. Hence each TelosB mote is connected to USB Hub through a USB cable. Every board has one USB Hub and all the motes on that board are connected to that USB Hub. Velcro is used to stick USB Hub, TelosB motes to the Plexi Glass Board.

2.2 Creation of the Test Bed

The creation of test bed consisted of two stages. The first stage was to create a highway road and the next stage was to create a road side intelligent highway infrastructure. These two actions were combined to form a complete Test Bed.

2.2.1 Creating a Highway for a Test Bed

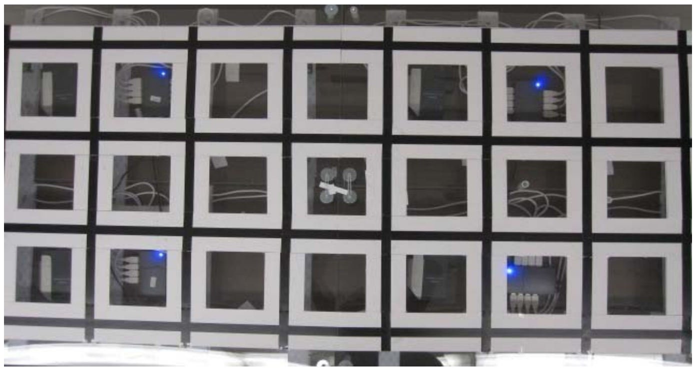

Figure 2-8 Highway on a Test Bed

Steps for creating a Highway for a Test Bed is described below.

1. Join four Plexi Glass Boards together.
2. Prepare a road representing a highway using white paper.
3. Stick the road prepared using the paper on the joined Plexi glass board.

2.2.2 Creating a Road Side Intelligent Highway Infrastructure for a Test Bed

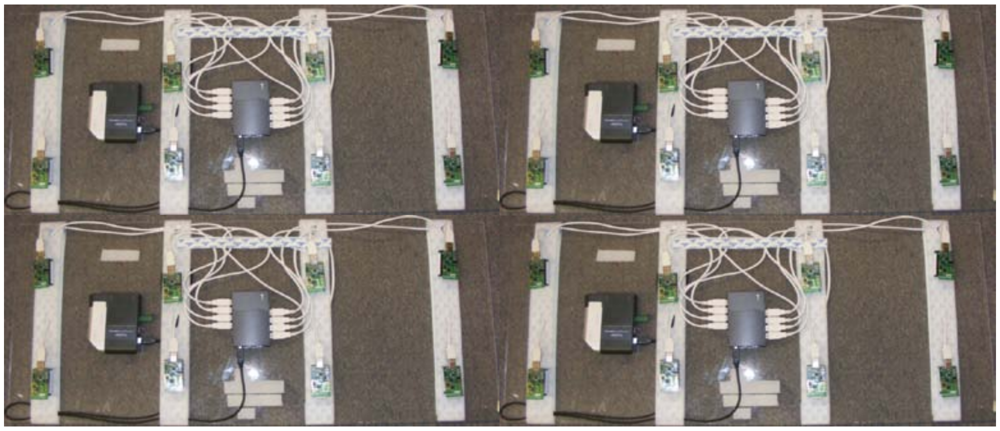

Figure 2-9 Road Side Infrastructure for a test bed

The following steps describe the creation of a road side intelligent highway infrastructure for a test bed.

1. On a Plexi Glass Board, strap the Velcro in four rows horizontally. Nine inches distance is to be maintained between rows.
2. For each row of Velcro, two TelosB motes are to be attached on it with a distance of nine inches between them. There are a total of 8 motes on each board.
3. A USB Hub is attached at the center of the board using Velcro.

4. All the motes are attached to the USB Hub with USB cable for powering and also for programming.
5. Each mote is attached with a PIR sensor which can detect movement.
6. The above procedure is done for all the four boards.
7. These four boards are joined together to form a road side intelligent highway infrastructure.

2.2.3 Vehicle-Highway Automation Test Bed

The creation of Test Bed is totally complete after combining the two parts which are created as above. The road side infrastructure created for test bed is placed below the highway created for the test bed. This way, the road side infrastructure can communicate with e-puck on the highway thus making the automation of highway possible.

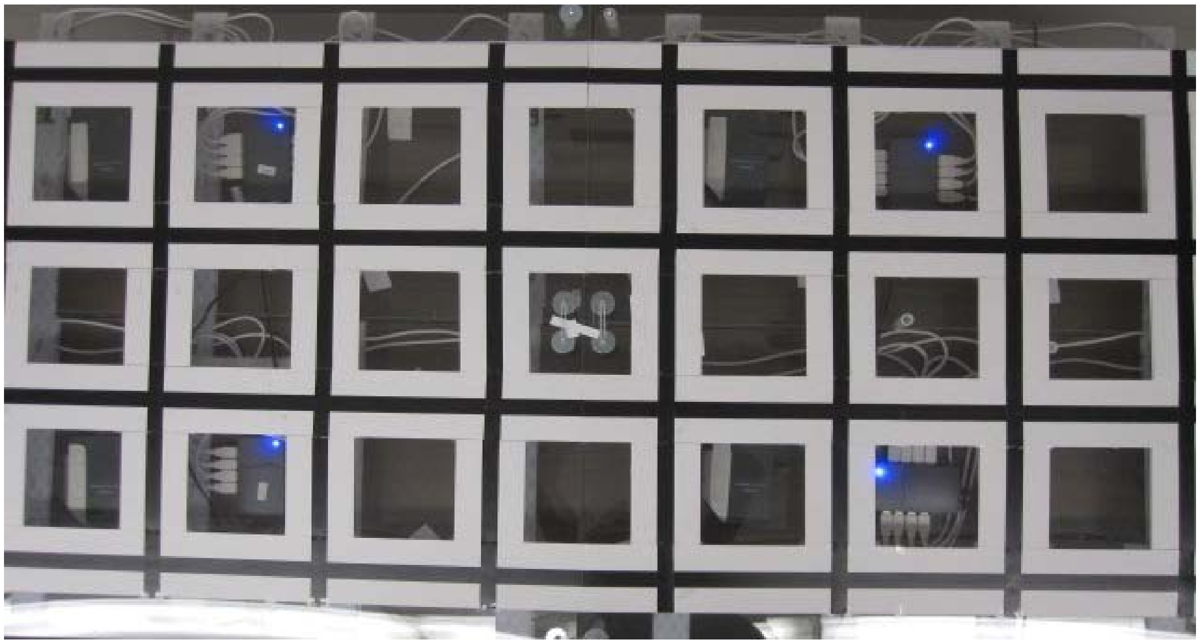

Figure 2-10 Vehicle-Highway Automation

CHAPTER 3 - Vehicle-Highway Automation Architecture

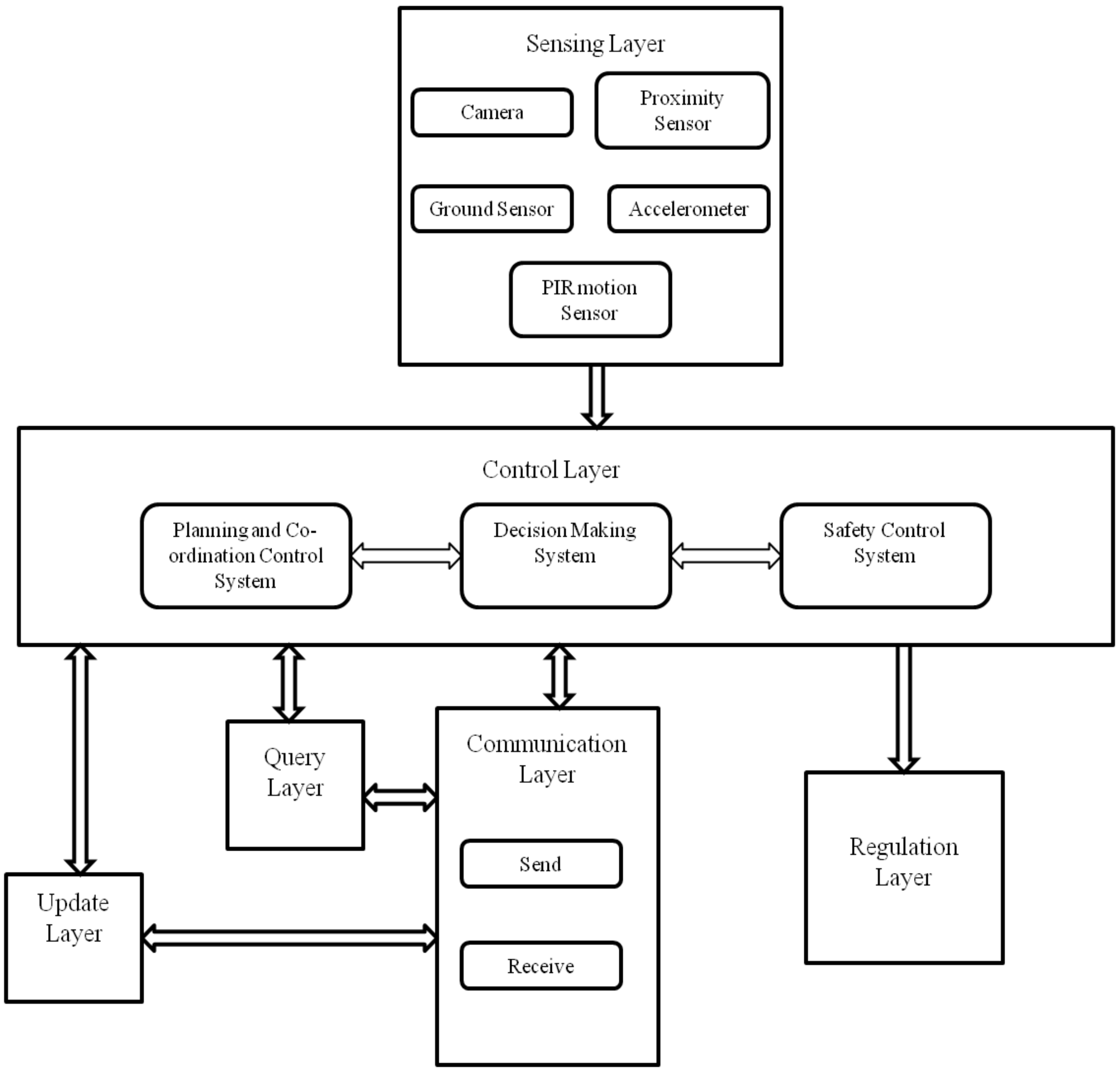

Figure 3-1 Vehicle-Highway Automation Architecture

The development of appropriate system architecture is a key to successful VHAS development. The VHAS architecture is shown in Figure 3-1 and consists of six major components which are as follows:

1. Sensing Layer
2. Control Layer
3. Regulation Layer
4. Communication Layer
5. Update Layer
6. Query Layer

The Sensing layer senses events of interest using the sensors and sends this data to the control layer. The Regulation Layer has interfaces to regulate the properties of the e-puck. The Communication layer is used by both the e-puck and the TelosB motes for communication with each other and also between themselves. The Update Layer is used to update or propagate the status of the node. The Query layer is used to query the information from motes and e-pucks. The Control layer is responsible for controlling the whole system. The control layer can send commands to the regulation layer, and can receive and send commands or data from the communication layer, update layer and query layer. The following concepts describe these layers in detail.

3.1 Sensing Layer

The Sensing layer is present in both the e-puck and in the TelosB mote placed on the road side infrastructure. The Sensing layer consists of several different interfaces for getting the data from different sensors and also processing them.

The Sensing layer of e-puck consists of proximity sensors, ground sensors, accelerometer and camera. The data from these sensors are collected and are sent to the control layer in regular intervals. The data obtained from the proximity sensors is used to detect collision by the control layer of the e-puck and hence can use it to avoid collisions. Values of ground sensors are used to follow the road or the particular lane on a highway by the control layer of the vehicle. The Control layer can also use camera of the sensing layer to follow the road and also to detect and avoid collision. Accelerometer can also be used to detect the inclination of the e-puck and adjust the speed accordingly.

The Sensing Layer of the TelosB mote present on the road side intelligent highway infrastructure system has several sensors such as PIR motion detection sensor, temperature sensor, light sensor and humidity sensor. Out of all these sensors, only PIR sensor is used in this project. The values from this sensor are sent to the control layer of the mote and hence the presence of e-puck or any object can be detected.

3.2 Regulation Layer

The Regulation layer is only present in the e-puck i.e. in the vehicle. This layer has several interfaces for regulating the properties of the vehicle. Speed, turn angle, stopping, starting, reversing are some of the properties of the vehicle that can be regulated

The Regulation layer gets the commands from the control layer of the e-puck and processes those commands to regulate the properties of the e-puck accordingly. The mote can also send commands to the regulation layer by sending a message to the e-puck. The control layer of the e-puck gets this message from the mote and in turn sends commands to the regulation layer for regulating the different properties of the e-puck.

There are several reasons for regulating an e-puck. Stopping and starting when needed, turning in different directions to reach destination, reversing, increasing and reducing speed when on steep road and obstacle avoidance are some of the reasons for regulating the e-puck.

3.3 Communication Layer

The Communication layer is a very important component as it provides communication between e-pucks, TelosB motes and also between each other. The Communication layer is present in both e-puck and TelosB mote.

Communication is done by radio transmission. The frequency of radio transmission is 2.4 GHz. Communication layer has different type of interfaces. Interfaces are used for sending and receiving data. The radio transmission power can also be set with the help of an interface.

Communication layer can send data and get data from control layer for e-puck and TelosB mote. Also for the mote, communication layer can send and receive data from update and query layer.

3.4 Update Layer

The Update layer is only present in the TelosB motes. The Update layer can receive commands from and send commands to the control layer and communication layer. The Update layer has several interfaces to send and receive update messages from different motes to update the status of the mote. It can send commands to neighboring nodes or to the far away nodes through multiple hops. When a mote receives an update command from some other node and if the command is for itself, it updates the status of that node else it forwards the command to the appropriate node.

3.5 Query Layer

The Query layer just as like the update layer, and can communicate with the control layer and the communication layer. The Query layer is used to query the status of the node from the neighboring nodes or from the far away nodes through multi hop communication, and has several interfaces to do so.

If a node wants to query the status of another node, then the control layer sends a command to the query layer, which in turn sends the query command to other motes using the communication layer. The Query command is received by the other node with the help of its communication layer, which is then processed though the query layer and a reply message giving the status of the node is sent back to the mote which was querying about it.

3.6 Control Layer

The Control layer is responsible for controlling the whole system, and communicates with all the layers in the system i.e. sensing layer, communication layer, regulation layer, update layer and query layer. The Control layer is present in both e-puck and the TelosB mote, and it controls the flow of data and commands.

The Control layer has three major components responsible for its correct and efficient functioning which are the Decision Making System, Planning and Co-ordination Control System and Safety Control System.

The Decision Making System is responsible for making major decision in the Control layer. All the data and commands in the Control layer are routed through the Decision Making System. When a message or a command is received by this system, the system analyzes the message/command and then a decision is made to route the flow of data or a command in an appropriate direction.

The Safety Control System controls all the safety related issues in the system. This system gets all the necessary sensor data from the Sensing layer and analyses it. If any safety warnings are detected from the analysis of the sensor data, then the Safety Control System takes the appropriate actions to ensure safety of the system. It also ensures safety if it receives any message regarding safety, from any of the other components of the system.

The Planning and Co-ordination Control System is necessary to plan the sequence of actions to be taken and also to co-ordinate between e-puck, road side infrastructure and also between each other. Planning includes getting the shortest path, shortest free path, less congestion traffic, obstacle prevention, avoiding deadlock, etc. Co-ordination control system's use is to co-ordinate the communication between e-pucks, motes and also between each other.

CHAPTER 4 - Vehicle-Highway Automation Interfaces

In this chapter, interfaces are defined and described which are used for the implementation of Vehicle-Highway Automation System. Every layer of Vehicle-Highway Automation System architecture provides several interfaces. These interfaces are defined in the following sections.

4.1 Sensing Layer Interfaces

The Sensing layer has several sensors such as camera, proximity sensors, ground sensors, PIR sensors. The Sensing layer provides interfaces for getting the data out of those sensors. These interfaces are defined as follows.

➔ Get_Camera_Capture_Data(Char Cam_Data[2*40*40])

This interface is provided by Sensing layer of e-puck. C programming language is used to program e-puck and hence this interface is written in C programming language. This interface is used to get the data values from the camera.

This function or interface takes one argument, i.e. the array of characters ‘Cam_Data []’. When this array of characters of size 2*40*40 is passed to this function, it gets the data from the camera and stores the same data in this ‘Cam_Data []’ array. The character array ‘Cam_Data[2*40*40]’ contains 40*40 RGB picture. Hence the data obtained can be used by the Control layer to follow a lane on a highway or for obstacle avoidance just by calling this function.

➔ *Get_Calibrated_Proximity_Sensor(int Prox_id)*

This interface or function is also provided by Sensing layer of e-puck. This interface is also written in C programming language. ‘Get_Calibrated_Proximity_Sensor( )’ function is used to get the sensor values from the proximity sensors. There are eight proximity sensor surrounding the e-puck.

This function takes one argument ‘Prox_id’. Prox_id is an integer variable and it defines the id of the proximity sensor. This function takes the id of the proximity sensor as an argument and returns sensor value of that particular proximity sensor in integer format. In this way all the sensor values of eight proximity sensors can be obtained by sending in an appropriate id. By invoking this interface the Control layer can perform Obstacle avoidance using the proximity sensor values.

➔ *Read_Floor_Sensor(int floor_id)*

Read_Floor_Sensor function is also provided by the Sensing layer of e-puck. This function is also written in C programming language. There are three Floor sensors at the bottom of the e-puck. This interface allows getting the sensor values from the Floor sensors.

'Floor_id,' an integer variable is taken as an argument to this interface. 'Floor_id' denotes the id of the floor sensor and when this id is passed to this function, the integer value of that particular Floor sensor is returned. When this interface is invoked by Control layer, three Floor sensor values are returned and hence with these values a lane on a highway can be tracked and followed.

➔ *Read_PIR_Sensor( )*

Sensing Layer of TelosB mote provides 'Read_PIR_Sensor ( )' interface. This interface is written in nesC programming language. The PIR sensor is attached to the top of the TelosB mote. Hence by this interface values from this sensor can be obtained.

This interface needs no arguments. When this interface is invoked by the Control layer of the TelosB mote an event is generated giving the value of the PIR sensor as an argument. This value obtained can be used to detect any object movement by the control layer of the TelosB mote.

4.2 Regulation Layer Interfaces

Regulation layer has several interfaces to control and regulate the movement of e-puck. These interfaces are written in C programming language. Definition and description of these interfaces is as below.

➔ *Move(float Speed, float Distance)*

This function or interface is available in the e-puck's Regulation layer. This function takes two floating point arguments 'Speed' and 'Distance'. These are the variables that can store floating point values.

'Move (Speed, Distance)' interface is used to move the e-puck straight for certain distance. Speed of the e-puck needed and distance to be travelled are given as inputs to this interface. Hence by this interface, Control layer of the e-puck can invoke this function from Regulation layer and make e-puck move straight up to the required distance.

➔ *Move_Time(float Speed, float Time)*

Regulation layer of e-puck has 'Move_Time (Speed, Time)' interface. This interface of Regulation layer takes two variables as argument 'Speed' and 'Time'. These two arguments are floating point variables.

'Move_Time (Speed, Time)' function when invoked will move the e-puck straight for certain amount of time. The interface takes two variables as inputs which are speed of the e-puck needed and time for which the e-puck is supposed to move straight. Therefore Control layer of e-puck can use this interface to make the e-puck move for a given amount of time.

➔ *SetSpeed(float Speed)*

This function is available in Regulation layer of e-puck. This takes one variable 'Speed' as an argument which is of floating point type. This function when invoked by the Control layer of e-puck, sets the speed of e-puck and hence enables the Control layer to control the speed of the e-puck.

➔ *Left_Turn (float Speed, int Stop)*

This function is available in Regulation layer of e-puck. This takes two variables as arguments - 'Speed' and 'Stop'. 'Speed' is a floating point variable and 'Stop' is an integer variable and it can take only two values which are 0 and 1.

'Left_Turn (Speed, Stop)' function when invoked will take a 90 degree left turn and stop if 'Stop' variable is 1. If its value is 0, it will continue to move straight after the left turn. The e-puck will turn and move with the speed that is given as input in the argument. Control layer of the e-puck can use this function of Regulation layer to take a left turn with the required speed.

➔ *Right_Turn (float Speed, int Stop)*

This function is available in Regulation layer of e-puck. This takes two variables as argument ‘Speed’ and ‘Stop’. ‘Speed’ is a floating point variable and ‘Stop’ is an integer variable and it can take only two values which are 0 and 1.

‘Right_Turn (Speed, Stop)’ function when invoked will take a 90 degree right turn and stop if ‘Stop’ variable is 1. Else, if it is 0, it will continue to move straight after the right turn. The e-puck will turn and move with the speed given as input in the argument. Control layer of the e-puck can use this function of Regulation layer to take a right turn with the required speed.

➔ *Turn (float Speed, float Angle, float Distance)*

This function is available in Regulation layer of e-puck. This takes three variables as floating point arguments - Speed, Angle and Distance.

Turn (Speed, Angle, and Distance) function when invoked will take a turn with the given angle and will not stop until the given distance is completed with the speed given in the argument. Speed, angle by which the e-puck is supposed to turn and the required distance are given as inputs to this function. Hence Control layer can use this function from Regulation layer to turn the e-puck at the desired angle, for the required distance.

4.3 Communication Layer

Communication layer has two interfaces each in both e-puck and TelosB mote for providing communication between e-pucks, TelosB motes and also between each other through radio transmission. Radio transmission power can also be adjusted using Communication layer. Interfaces in Communication layer of e-puck are written in C programming language whereas interfaces in Communication layer of TelosB mote are written in nesC programming language. These interfaces are described as follows.

➔ *Send(int dest_id, int Size_Of_Message, TOS_MSG_PTR MSG)*

This function is provided by the Communication layer of TelosB mote. This interface takes three fields as an argument ‘dest_id’, ‘Size_Of_Message’ and ‘TOS_MSG_PTR’. ‘dest_id’ and ‘Size_Of_Message’ are integer variables. ‘MSG’ is a pointer to a message of type ‘TOS_MSG’ structure.

This function is used to send a message packet of type TOS_MSG structure to the destination id given. Argument ‘dest_id’ can be the id of e-puck or TelosB mote to which the mote wants to send a message. ‘MSG’ is a pointer to the message that needs to be sent to the e-puck or another mote of type TOS_MSG struct. ‘Size_Of_Message’ argument is the size of the message that is being sent. By invoking this function Control layer and other layers can send the message they want to send to any required e-puck or any other required TelosB mote. When this message is sent, an event ‘Send_Done’ is generated.

➔ *Receive (TOS_MSG MSG)*

This is an event generated when the Communication layer of TelosB mote receives a message. It has one argument ‘MSG’. ‘MSG’ is the received message of type TOS_MSG. This message can be sent by e-puck or by TelosB. In this way Communication layer receives the message and hence this received message can be used by different layers of TelosB mote.

➔ *SendPacket (char destinationgroup, int destinationaddress, char* packet, int packetsize)*

‘SendPacket’ function is provided by the Communication layer of e-puck. This function has three fields ‘destinationgroup’, ‘destinationaddress’, ‘packet’ and ‘packetsize’. ‘Destinationgroup’ is a character variable in which the group address of the destination is stored. Variable ‘destinationaddress’ is an integer variable in which the destination addresses of the destination can be stored. The destination can be a mote or an e-puck. ‘Packet’ is a character pointer to the message packet which is of character type. Variable ‘packetsize’ is an integer variable which stores the size of the message packet that is being sent.

This function is written in C programming language and the message packet that is being sent is a character array. When this function is invoked by any layer of e-puck, the message that is being sent is converted and type casted into TOS_MSG struct type and is sent through radio transmission. This function returns 1, if the message is sent. Therefore in this way the message packet can be sent to either e-puck or mote.

➔ *IsPacketReady (char* packet, int* packetSize)*

‘IsPacketReady’ is the function provided by the communication layer of e-puck. This function has two fields ‘packet’ and ‘packetSize’. ‘packet’ is a character pointer of type character, to a message packet that is being received. ‘packetSize’ is an integer pointer to the integer variable which stores the size of receiving message packet.

When this function is invoked, if message is received it returns 1 and stores the message packet in the character array to which the first argument character pointer ‘packet’ is pointing to. Else if the message is not received, then it returns 0.

Message packet is in TOS_MSG struct format when received from radio. This message is then converted or type casted into character message and then stores the character message into character array to which character pointer ‘packet’ is pointing. This way any layer of the e-puck can get the message packet by invoking this function.

4.4 Update Layer Interfaces

Update layer has an interface to send the update message to the other motes to update the status of its node. This layer also has an interface to receive the update message from the other nodes to update their status. These interfaces are defined and described as follows.

➔ *Send_Update(TOS_MSG Msg)*

‘Send_Update’ function is available in the Update layer of TelosB mote. This interface contains one argument ‘Msg’ of type TOS_MSG struct. When invoked, it sends an update message containing its status to the other motes.

When Control layer invokes this function from update layer of mote, it sends an update message containing its status and destination address of the mote, to the same destination mote. If the destination mote is not the neighboring mote, then it sends the message in the shortest multi hop route.

➔ *Receive_Update(TOS_MSG Msg)*

‘Receive_Update’ is an event generated when an update message is received from some node. This contains one argument ‘Msg’ of type TOS_MSG struct.

When the update message is received, this event is generated. When this event is generated it checks whether this message is for itself by verifying the destination address in the message. If the message is for itself, then it updates the status of that node locally. Else, if the update message is not for itself it forwards the packet to the destination mote in the shortest path.

4.5 Query Layer Interfaces

Query layer has several interfaces to query the status of the nodes. These interfaces of the Query layer present in the TelosB mote are defined and described as follows.

➔ *Query_Next_Node (int next_node_address)*

‘Query_Next_Node’ interface or a function is present in Query layer of TelosB mote. This interface is used to query the status of the next node to itself. This interface takes one integer variable ‘next_node_address’ as argument.

When Control layer of TelosB mote wants to query or get the status of the node next to itself, address of that node is given as an input to this function. This interface then sends a query message to that node by obtaining next node address from the input. When that next node receives the query message through Communication layer, it calls the ‘Send_Update’ interface of the Update layer which will send its status to the source node. When the update message reaches the source node, the status of the node is obtained.

➔ *Query_Node (int dest_node_address)*

‘Query_Node’ interface or a function is present in Query layer of TelosB mote. This interface is used to query the status of any node in the WSN. This interface takes one integer variable ‘dest_node_address’ as argument.

When Control layer of TelosB mote wants to query or get the status of any one node in WSN, the address of that destination node is given as an input to it. This interface takes the destination address from the input and then calculates the shortest path to that input. The ‘Query_Node’ message containing the destination address is then sent to the same destination node in the shortest multi hop route.

When the destination node receives this ‘Query_Node’ message, the Control layer of this node invokes ‘Send_Update’ to send the status of the node to the source node. When this update message is received by the source node, ‘Receive_Update’ event is generated giving the source node the status of the destination node for which the node is queried for.

➔ *Query_Path_toNode (int dest_node_address)*

‘Query_Path_toNode’ interface or a function is present in the Query layer of TelosB mote. This interface is used to query the status of all the nodes that are present in the path obtained to reach the destination node. This interface takes one integer variable ‘dest_node_address’ as argument.

When the Control layer of TelosB mote wants to query or get the status of all the nodes that are present in the path obtained to reach the destination node, address of that destination node is given as an input to it. This interface takes the destination address from the input and then calculates the shortest path to that input. Then ‘Query_Path_toNode’ message containing the destination address is sent to the same destination node through this shortest multi hop path.

When the destination node receives this ‘Query_Node’ message, its Control layer sends the ‘Update’ message including status of its node to the source mote. This message is sent in the same shortest path route. Every node in the route back adds its status to this ‘Update’ message. Hence by this, the source mote can get the status of all the motes in the path to destination.

4.6 Control Layer Interfaces

Control layer is responsible for controlling the whole system. It involves decision making, controlling safety issues and also for planning and coordinating the whole system. Control layer provides several interfaces for controlling all these.

➔ *Obstacle_Avoidance( )*

‘Obstacle_Avoidance ( )’ interface is provided by Control layer of e-puck. This interface has no arguments. It ensures safety of the system by detecting and avoiding obstacles.

This interface first detects for any obstacle ahead by getting all the eight Proximity sensor values from the Sensing layer of e-puck by invoking ‘Get_Calibrated_Proximity_Sensor’ interface. From the sensor data obtained, it checks for any obstacle and if any obstacle is detected, then it calls the interfaces of Regulation layer to stop or to change the direction of e-puck. Hence by this interface of Control layer, obstacle avoidance is achieved.

➔ *Lane_Following( )*

‘Lane_Following ( )’ interface is provided by Control layer of e-puck. This interface has no arguments. This interface is used to follow a particular lane or a road on a highway.

Following a lane or a road on a highway is implemented by using the Floor or Ground sensors which are present at the bottom of the e-puck. This interface gets the sensor values of the three Floor sensors by invoking ‘Read_Floor_Sensor’ interface of Sensing layer. It uses this sensor data obtained from these sensors for detecting a lane or a road. Hence it calls appropriate interfaces of Regulation layer to follow the lane or a road.

➔ *Detect_epuck ( )*

‘Detect_epuck’ interface is provided by Control layer of TelosB mote. This interface has no arguments. It is used to detect the e-puck on a road and update the information of it, in the mote.

Detecting an e-puck on a road is implemented by using the PIR sensor which is present on top of the TelosB mote. This interface gets the sensor value from this PIR sensor by invoking ‘Read_PIR_Sensor’ interface of Sensing layer. It uses this sensor data obtained from PIR sensor to detect the e-puck on a road. When an e-puck is detected, it calls interfaces of Update layer to update this information in the mote.

➔ *Get_Shortest_Path (int src_address, int dest_address)*

‘Get_Shortest_Path’ interface is also provided by Control layer of TelosB mote. This interface has two arguments ‘src_address’ and ‘dest_address’ which are of integer data type.

Source node address and destination address are given as inputs to this interface. When invoked, this interface gives the shortest path to the given destination node from the given source node. This interface is used by several interfaces of different layers.

➔ *Reserve_node (int node_address)*

‘Reserve_node’ interface is provided by Control layer of TelosB mote. This interface takes one argument ‘node_address’ which is of integer data type. This interface is used to reserve a node (i.e. the node provided as an input to this interface) if that node is not reserved.

Address of the node which has to be reserved is given as an input to this interface. The interface takes this address and sends a ‘reserve’ message to the given node address for reserving it. If that node is not already reserved by any other node it reserves the node.

➔ Reserve_NextNode (int dest_node_address)

‘Reserve_NextNode’ interface is provided by Control layer of TelosB mote. This interface takes one argument ‘dest_node_address’ which is of integer data type. This interface is used to reserve the next node in the shortest path to the destination node.

Address of the destination node is given as input to this interface. The interface calculates the shortest path to this destination address and reserves the next available node in that shortest path.

➔ *Reserve_Path (int dest_address)*

‘Reserve_Path’ interface is also provided by Control layer of TelosB mote. This interface takes one argument ‘dest_address’ which is of integer data type. This interface is used to reserve all the nodes in a path to the destination node from itself.

Address of the destination node till where the nodes in a path have to be reserved is given as input to this interface. When invoked, this interface takes the destination node address and finds a shortest route to the destination. In that shortest path it sends a ‘reserve_path’ message to the destination node. If any node in that path is not available for reservation, then all the node reservations in that path are cancelled and a ‘cancel’ message is sent back to the source mote.

When the source mote receives the cancel message, it again calculates the shortest path to the destination node excluding the node which was not available for reservation. Again the above steps are repeated. These steps are repeated until it can reserve the path to destination.

➔ *Reserve_Available_Path (int dest_address)*

‘Reserve_Available_Path’ interface is also provided by control layer of TelosB mote. This interface takes one argument ‘dest_address’ which is of integer data type. This interface is used to reserve all the available nodes in the path to the destination node starting from itself.

Address of the destination node till where the nodes in a path have to be reserved is given as input to this interface. When invoked this interface takes the destination node address and finds the shortest route to the destination. In that shortest path, it sends a message to the destination. If any node in that path cannot be reserved, then the node before that node in that path finds an alternative shortest path from there to reach the destination and reserves it. The destination node then sends the reserved path to the source mote. If there is no route to the destination then the node at the end of the reserved nodes sends the path up to this available reserved node to the source node.

➔ *goto_dest_NextNode (int dest_address)*

‘goto_dest_NextNode’ interface is provided by the control layer of the e-puck. It takes only one argument which is of integer data type. This interface is used by e-puck to reach to the destination by querying and reserving one node at a time.

Address of the destination node to which e-puck wants to travel is given as input to this interface. This interface allows e-puck to move to the destination node which is given as an argument by coordinating with every mote it reaches and reserving the next available mote in the shortest path to the destination node.

➔ *goto_dest_Path(int dest_address)*

‘goto_dest_Path’ interface is provided by the control layer of e-puck. It takes only one argument which is of integer data type. This interface is used by e-puck to reach the destination by querying and reserving the whole path at a time.

Address of the destination node to which e-puck wants to travel is given as input to this interface. This interface allows e-puck to move to the destination node which is given as argument by coordinating with only source mote and reserving the whole path to the destination at once by that mote. If there is no path where all the nodes can be reserved, then e-puck waits until one of the paths is reserved.

➔ *goto_dest_AvailablePath(int dest_address)*

‘goto_dest_AvailablePath’ interface is provided by the control layer of e-puck. It takes only one argument which is of integer data type. This interface is used by e-puck to reach the destination by reserving all the nodes in the path that are available for reservation at a time.

Address of the destination node to which e-puck wants to travel is given as input to this interface. This interface allows e-puck to move to the destination node which is given as an argument by coordinating with the motes and reserving all the available nodes in the shortest path to the destination at once. If any node in the path is not available for reservation, then the source mote reserves all the nodes till that point. E-puck starts and reaches the point till the nodes are reserved and then repeats the above process once more.

CHAPTER 5 - Algorithms

We developed several algorithms to implement on the Vehicle-Highway Automation system. The algorithms developed were implemented in C and nesC languages. These algorithms are discussed in the following sections.

5.1 Obstacle Avoidance Algorithm

The algorithm for avoiding obstacle has been developed and implemented in C language. The ‘Obstacle_ Avoidance ( )’ interface of Control layer uses this algorithm. The pseudo code for the algorithm is given as follows.

Steps:

1. Read the sensor values from Proximity Sensors.
2. Analyze the sensor values.
3. Look for any obstacle near it.
4. If any obstacle is detected then stop the e-puck or move back.

Pseudo Code:

```
do
    Read 8 Proximity sensor values into prox [8];
    for i ← 1 to 8
        if (prox [i] > 1000)    //obstacle is detected
            SetSpeed (0)    // Stop the e-puck
        fi
    end of for
od
```

5.2 Line Following Algorithm

Line following algorithm has been developed and implemented in C language. The ‘Lane_Following ( )’ interface of Control layer uses this algorithm. The pseudo code for the algorithm is given as follows.

Steps:

1. Read the three sensor values from Floor Sensor.

2. Analyze the three sensor values. Center floor sensor is used to track the black line. Left and right floor sensors are used to check for deviation.
3. Detect the black line.
4. Follow the detected black line by controlling the movement of e-puck.

Pseudo Code:

```
do
    Read the 3 Floor sensor value into floor [3]
    if (floor [1]>200 i.e. the center sensor is on the black line)
        move the e-puck straight
    else if (floor [0]>200 i.e. the left sensor is on the black line)
        move the e-puck towards left side.
    else if (floor [2]>200 i.e. the right sensor is on the black line)
        move the e-puck towards right side.
    fi
od
```

5.3 E-puck Detection Algorithm

The E-puck detection algorithm is developed and implemented in nesC language. ‘Detect_epuck’ interface of Control layer uses this algorithm. Pseudo code of this algorithm is given as follows.

Steps:

1. Read sensor value from the PIR sensor.
2. Look for any e-puck movement by analyzing the sensor value.
3. If e-puck movement is detected then update the information.

Pseudo Code:

```
do
    Read sensor value into PIR
    if (PIR >50 i.e. e-puck is detected)
        store the information about the e-puck in the mote
    fi
od
```

5.4 Next Node Algorithm

The Next Node algorithm is used to allow e-puck reach the desired destination by reserving one node at a time. The part of the algorithm running on the e-puck uses the '*goto_dest_NextNode*' interface. The part of the algorithm running on the TelosB uses the '*Reserve_NextNode*' interface. This algorithm is explained as follows.

Steps:

1. E-puck sends a message with the destination node address in it to the mote that is nearest to the current position of e-puck to reserve the next available node that is in one of the shortest path to the destination.
2. The mote which receives this message calculates the shortest path to the destination.
3. Then the mote sends a message to the next node in the shortest path that e-puck has to travel to reach the destination asking it to reserve if it is not already reserved.
4. If the next node is not already reserved then it reserves the node and sends a message back to the node confirming the reservation.
5. Else if the next node is already reserved, it sends a cancel message back to the node indicating no reservation.
6. If the node near the e-puck gets the confirmation of reservation, it then sends the address of the next node to the e-puck
7. Else if the node gets the cancel message from the e-puck, it once again calculates the shortest path to the destination excluding the next node from which it got a cancel message and repeats the process STEP 3.
8. When e-puck gets the node id from the mote, it moves to that next node whose node id is received from the mote. When e-puck leaves the current node, the reservation at the current node is removed.
9. E-puck again starts the same process from STEP 1. This process is repeated until the destination is reached.

Pseudo Code:

Code that is running on e-puck:

```
while (e-puck reaches the destination)
    Send 'reserve_next' message with the destination node address in it.
    while (1)
```

```
        if (message is received from the mote with next node address)
            Break from the while loop.
        fi
    End of while
    Move to the next node.
End of while
```

Code that is running on TelosB motes:

```
While (1)
    if (receives a ‘reserve_next’ message from e-puck)
        Calculate the shortest path to the destination node.
        Send a ‘reserve’ message to the next node in that shortest path.
    fi
    if (receives a ‘reserve’ message from the mote)
        if (not already reserved)
            reserve the node
            send ‘reserve_confirm’ message back to that mote.
        else if (already reserved)
            send ‘reserve_cancel’ message back to that mote.
        fi
    fi
    if (receives a ‘reserve_confirm’ message from the next node)
        send a message back to the e-puck with the next node address
    fi
    if (receives a ‘reserve_cancel’ message from the next node)
        Calculate the shortest path to the destination node excluding the next node
        that it got message from.
        Send a ‘reserve’ message to the next node in that shortest path.
    fi
End of While.
```

5.5 Whole Path Algorithm

Whole path algorithm is used to allow e-puck reach the desired destination by reserving whole path at a time. E-puck part of this algorithm is used by '*goto_dest_Path'* interface. The TelosB part of this algorithm is used by '*Reserve_Path*' interface. This algorithm is explained as follows.

Steps:

1. E-puck sends a message to the mote with the destination node address in it that is nearest to the current position of e-puck to reserve the whole path to the destination node.
2. The mote which receives this message calculates the shortest path to the destination.
3. Then the mote sends a 'reserve_path' message to the next node in that shortest path to the destination.
4. The node which receives this 'reserve_path' message checks whether it can reserve the node or not.
5. If it can reserve the node and if it is not the destination node then it reserves the node and repeats the STEP 2 and STEP3. If it is the destination node then it reserves the node and sends the 'reserve_confirm' message to the source mote in the same shortest path with its node address included in the message.
6. Else if it cannot reserve the node, then it sends 'reserve_cancel' message back to the mote with its node id embedded into the message.
7. If a node which is not source node receives the 'reserve_cancel' message then it removes its reservation and sends 'reserve_cancel' message back in the same path. If it is the source node then it calculates the shortest path excluding the node which was not able to reserve and repeats the STEP 3.
8. If the node receives the 'reserve_confirm' message and if the node is the source node, then it sends the reserved path to the e-puck. If the node is not the source node, then it sends the same 'reserve_confirm' message back to the mote with its address embedded into the message.
9. E-puck when receives the reserved path from the node near to it, moves to the destination node in that path.

Pseudo Code:

Code that is running on e-puck:

```
Send 'reserve_whole' message with the destination node address in it.
while (1)
    if (message is received from the mote with all the reserved path nodes)
        Break from the while loop.
    fi
End of while
Move to the destination in the reserved nodes path
```

Code that is running on TelosB mote:

```
While (1)
    if (receives a 'reserve_whole' message from e-puck)
        Calculate the shortest path to the destination node.
        Send a 'reserve_path' message to the next node in that shortest path.
    fi
    if (receives a 'reserve_path' message from the mote)
        if (not already reserved)
            Reserve the node.
            if (this node is not the destination node)
                Calculate the shortest path to the destination node.
                Send 'reserve_path' message to the next node in this path.
            else if (this node is the destination node)
                Send 'reserve_confirm' message back to that mote
                including its address in the message.
            fi
        else if (already reserved)
            Send 'reserve_cancel' message with its node address back to that
            mote.
        fi
    fi
    if (receives a 'reserve_confirm' message from the next node)
        if (the node is not the source node)
```

```
                    Send the same ‘reserve_confirm’ message back to that mote by
                    adding its node address.
               else if (the node is the source node)
                    Send the message back to the e-puck including all the reserve node
                    addresses in it.
               fi
          fi
          if (receives a ‘reserve_cancel’ message from the next node)
               if (the node is not the source node)
                    Remove reservation.
                    Send the same ‘reserve_cancel’ back to the mote.
               else if (the node is the source node)
                    Calculate the shortest path to the destination node excluding the
                    node that was not able to reserve.
                    Send a ‘reserve_path’ message to the next node in that shortest
                    path.
               fi
          fi
     End of while
```

5.6 Available Path Algorithm

Available path algorithm is used to allow e-puck reach the desired destination by reserving all the available nodes in a path at a time. E-puck part of this algorithm is used by ‘*goto_dest_AvailablePath’* interface. The TelosB part of this algorithm is used by ‘*Reserve_Available_Path*’ interface. This algorithm is explained as follows.

Steps:

1. E-puck sends a message to the mote with the destination node address in it that is nearest to the current position of e-puck to reserve the available path to the destination node.
2. The mote which receives this message calculates the shortest path to the destination.

3. Then the mote sends a ‘reserve_avail_path’ message to the next node in that shortest path to the destination.
4. The node which receives this ‘reserve_avail_path’ message checks whether it can reserve the node or not.
5. If it can reserve the node, then it reserves the node and also stores the time at which e-puck will be here. If it is not the destination node, it repeats the STEP 2 and STEP 3. Else if it is the destination node then it reserves the node and sends the ‘reserve_confirm’ message to the source mote in the same shortest path with its address embedded into the message.
6. Else if it cannot reserve the node, then it sends ‘reserve_cancel’ message back to the mote with its node id embedded into the message.
7. If a node receives the ‘reserve_cancel’ message from a node which is not the destination node, then it calculates the shortest path excluding the node which was not able to reserve and repeats the STEP 3. If a node receives the ‘reserve_cancel’ message from the destination node, then it sends ‘reserve_confirm’ message back to the mote with its address embedded into the message.
8. If the node receives the ‘reserve_confirm’ message and if the node is source node, then it sends the reserved path to the e-puck. If the node is not the source node, then it sends the same ‘reserve_confirm’ message back to the mote with its address embedded into the message.
9. E-puck when receives the reserved path from the node near to it, moves to the destination node or till the last reserved node in the path obtained from the message.
10. E-puck checks whether it reached the destination or not. If it did not reach the destination then it repeats the process from STEP 1.

Pseudo Code:

Code that is running on e-puck:

```
While (e-puck reached the destination)
    Send ‘reserve_availPath’ message with the destination node address in it.
    while (1)
        if (message is received from the mote with all the reserved nodes of path)
            Break from the while loop.
```

```
                fi
        End of while
    Move to the destination or till the last reserved node in the path obtained
    End of while
```

Code that is running on TelosB mote:

```
    While (1)
        if (receives a 'reserve_availPath' message from e-puck)
            Calculate the shortest path to the destination node.
            Send a 'reserve_avail' message to the next node in that shortest path.
        fi
        if (receives a 'reserve_avail' message from the mote)
            if (not already reserved at the time when e-puck will be here)
                Reserve the node.
                Store the time at which e-puck will be here.
                if (this node is not the destination node)
                    Calculate the shortest path to the destination node.
                    Send 'reserve_avail' message to the next node in this path.
                else if (this node is the destination node)
                    Send 'reserve_confirm' message back to that mote
                    including its address in the message.
                fi
            else if (already reserved at the time when e-puck will be here)
                Send 'reserve_cancel' message with its node address back to that
                mote.
            fi
        fi
        if (receives a 'reserve_confirm' message from the next node)
            if (the node is not the source node)
                Send the same 'reserve_confirm' message back to that mote by
                adding its node address.
            else if (the node is the source node)
```

```
                Send the message back to the e-puck including all the reserved
                node addresses in it.
            fi
        fi
        if (receives a 'reserve_cancel' message from the next node)
            if (node from which this message is obtained is not the destination node)
                Calculate the shortest path to the destination node excluding the
                node that was not able to reserve.
                Send a 'reserve_avail' message to the next node in that shortest
                path.
            else if (node from which this message is received is the destination node)
                Send the 'reserve_confirm' message back to that mote by including
                its node address.
            fi
        fi
    End of while
```

CHAPTER 6 - Performance Analysis

Vehicle-Highway Automation system is tested for performance and analyzed in several different ways. The system is tested for different scenarios with different amounts of area. As described in earlier chapters, three different algorithms were used to allow e-puck reach the desired destination on a highway. Performances of these algorithms are also analyzed in this chapter. Vehicle-Highway Automation system was also tested for safety related issues and for accuracy of the system. These are described as follows.

6.1Test with four boards

Performance of vehicle highway automation system and the three algorithms is done with all the four boards put together. Two scenarios were considered for testing. Performance analysis of these scenarios is as follows.

6.1.1 First scenario

In this scenario all the e-pucks have different starting point and have a one common destination. The results from this testing are described and analyzed as follows.

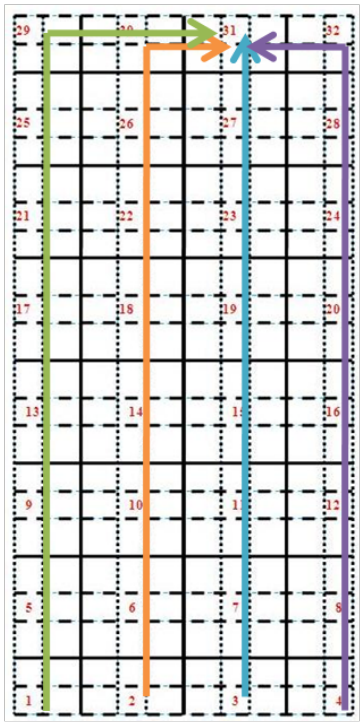

Figure 6-1 First Scenario

E-puck / no of epuck	1 – epuck	2 – epuck	3 – epuck	4 – epuck
Path: 3 to 31	Hops: 7 Time: 26 Msg: 28	Hops: 7 Time: 26 Msg: 28	Hops: 7 Time: 26 Msg: 28	Hops: 7 Time: 26 Msg: 28
Path: 2 to 31	Hops: 8 Time: 30 Msg: 32	Hops: 8 Time: 32 Msg: 34	Hops: 8 Time: 32 Msg: 34	Hops: 8 Time: 32 Msg: 34
Path: 1 to 31	Hops: 9 Time: 33 Msg: 36	Hops: 9 Time: 33 Msg: 36	Hops: 11 Time: 45 Msg: 46	Hops: 11 Time: 45 Msg: 46
Path: 4 to 31	Hops: 8 Time: 30 Msg: 32	Hops: 8 Time: 32 Msg: 34	Hops: 8 Time: 34 Msg: 36	Hops: 10 Time: 41 Msg: 42

Table 6-1 Next Node Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 3 to 31	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 16
Path: 2 to 31	Hops: 8 Time: 33 Msg: 18	Hops: 8 Time: 33 Msg: 22	Hops: 8 Time: 33 Msg: 22	Hops: 8 Time: 33 Msg: 22
Path: 1 to 31	Hops: 9 Time: 37 Msg: 20	Hops: 9 Time: 37 Msg: 24	Hops: 9 Time: 37 Msg: 24	Hops: 9 Time: 37 Msg: 24
Path: 4 to 31	Hops: 8 Time: 33 Msg: 18	Hops: 8 Time: 33 Msg: 22	Hops: 8 Time: 33 Msg: 22	Hops: 8 Time: 33 Msg: 22

Table 6-2 Available Path Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 3 to 31	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 1	Hops: 7 Time: 28 Msg: 1	Hops: 7 Time: 28 Msg: 1
Path: 2 to 31	Hops: 8 Time: 33 Msg: 1	Hops: 8 Time: 59 Msg: 1	Hops: 8 Time: 59 Msg: 1	Hops: 8 Time: 59 Msg: 1
Path: 1 to 31	Hops: 9 Time: 37 Msg: 1	Hops: 9 Time: 63 Msg: 1	Hops: 9 Time: 96 Msg: 1	Hops: 9 Time: 96 Msg: 1
Path: 4 to 31	Hops: 8 Time: 33 Msg: 1	Hops: 8 Time: 59 Msg: 1	Hops: 8 Time: 90 Msg: 1	Hops: 8 Time:124 Msg: 1

Table 6-3 Whole Path Algorithm

The figure 6-1 shows the scenario where all the e-pucks have a different starting point and a common destination i.e. node 31. The e-puck at node 3 starts first and then e-puck at node 2, node 1 and node 4 start respectively. This scenario is tested with all the three algorithms. Next Node Algorithm, Available Path Algorithm and Whole Path Algorithm are the three algorithms which allow e-puck to move to the destination. The results obtained when running these algorithms are available in Table 6-1, Table 6-2 and Table 6-3 for next node, available path and whole path algorithms respectively. The results are obtained separately for each case i.e. when there is one e-puck, two e-pucks, three e-pucks or four e-pucks running on the board at a time. These results in the table contain number of hops taken, amount of time taken and number of messages sent out for an e-puck to reach the final destination.

It can be observed from the tables that all the algorithms take almost the same number of hops to reach the destination except the next node algorithm, which takes more number of hops to reach the destination when all the four e-pucks are running at a time on board. It can also be observed that the time taken to reach the destination is almost the same for next node algorithm and available path algorithm except that the next node algorithm takes a little more time when there are all the four e-pucks running on the board at a time. But when whole path algorithm is used, it takes a lot of time when there is more than one e-puck running on the test bed at one point of time. If the number of messages sent is considered then it can be clearly observed that the number of messages sent out by available path algorithm is far less than the messages sent out by next node and whole path algorithm.

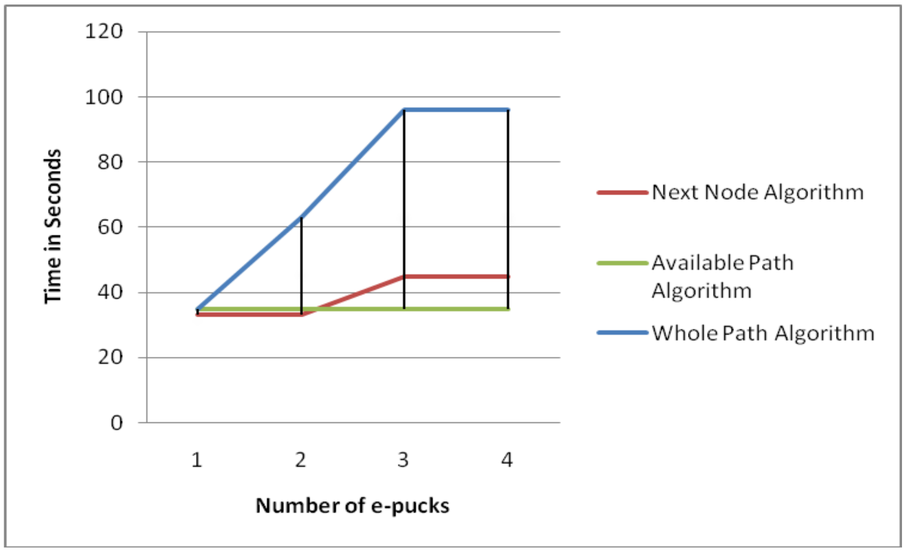

Figure 6-2 Time in seconds vs Number of e-pucks for Scenario 1

The figure 6-2 gives the Time vs Number of e-pucks graph of e-puck which starts at node-1 for all the three algorithms. Clearly from the graph it can be observed that the available path algorithm performs better when compared to the other algorithms for this scenario in terms of time taken. Next node algorithm takes a little more time than the available path algorithm but much less than the whole path algorithm. The time taken by the available path algorithm is almost the same even when the number of e-pucks running on the board increase, but the time taken by the next node algorithm and whole path algorithm increase.

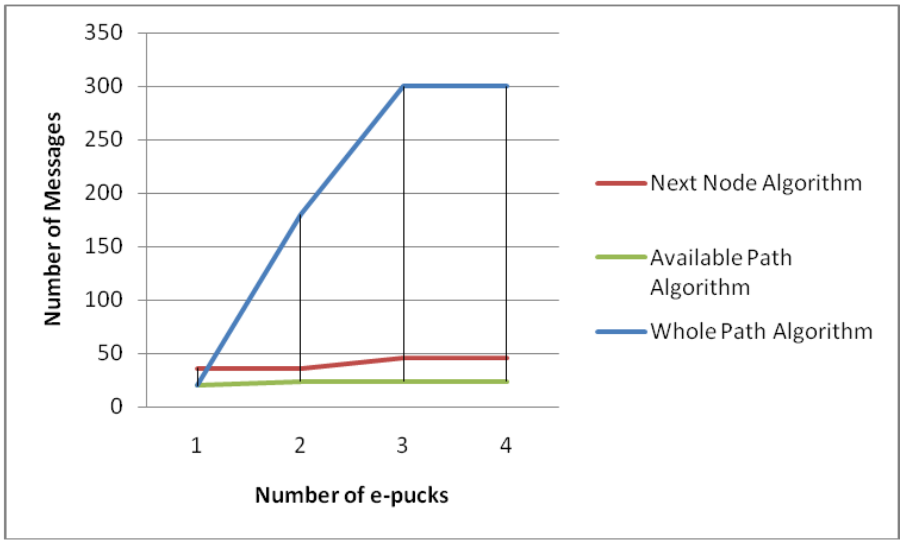

Figure 6-3 Number of messages vs Number of e-pucks for Scenario 1

The figure 6-3 gives the Number of messages vs Number of e-pucks graph of e-puck which starts at node-1 for all the three algorithms. Clearly from the graph, it can be observed that the available path algorithm performs better than the other algorithms for this scenario in terms of the number of messages sent out. Next node algorithm sends out more messages than the available path algorithm but much less than the whole path algorithm. The number of messages by the available path algorithm is almost the same when the number of e-pucks running on the board increase. But the number of messages sent by the next node algorithm and whole path algorithm increases when the number of e-pucks running on the board increase.

6.1.2 Second Scenario

In this scenario all the e-pucks have different starting points and have different destinations and move perpendicular to each other. The results from this testing are described and analyzed as follows.

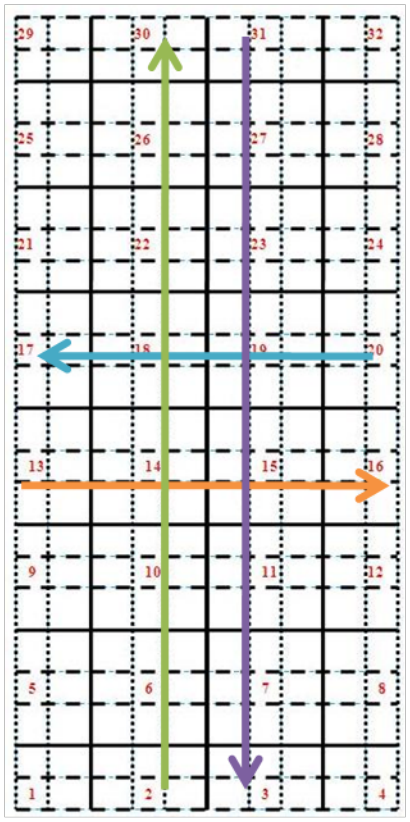

Figure 6-4 Second Scenario

E-puck / no of epuck	1 – epuck	2 – epuck	3 – epuck	4 – epuck
Path: 20 to 17	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12
Path: 13 to 16	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12
Path: 2 to 30	Hops: 7 Time: 23 Msg: 28	Hops: 7 Time: 23 Msg: 28	Hops: 7 Time: 23 Msg: 28	Hops: 7 Time: 23 Msg: 28
Path: 31 to 3	Hops: 7 Time: 23 Msg: 28	Hops: 7 Time: 23 Msg: 28	Hops: 7 Time: 23 Msg: 28	Hops: 7 Time: 23 Msg: 28

Table 6-4 Next Node Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 20 to 17	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 13 to 16	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 2 to 30	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 20	Hops: 7 Time: 28 Msg: 20
Path: 31 to 3	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 28 Msg: 20	Hops: 7 Time: 28 Msg: 20

Table 6-5 Available Path Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 20 to 17	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 13 to 16	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 30	Hops: 3 Time: 13 Msg: 30	Hops: 3 Time: 13 Msg: 30
Path: 2 to 30	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 32 Msg: 34	Hops: 7 Time: 38 Msg: 56	Hops: 7 Time: 38 Msg: 56
Path: 31 to 3	Hops: 7 Time: 28 Msg: 16	Hops: 7 Time: 32 Msg: 16	Hops: 7 Time: 38 Msg: 56	Hops: 7 Time: 38 Msg: 56

Table 6-6 Whole Path Algorithm

The figure 6-4 shows the scenario where all the e-pucks have a different starting point and different destinations and move in perpendicular directions to each other. The e-puck at node 20 starts first and then e-pucks at node 13, node 2 and node 30 start. This scenario is tested with all the three algorithms. The results obtained when running these algorithms are available in Table 6-4, Table 6-5 and Table 6-6 for next node, available path and whole path algorithms respectively. The results are obtained separately for each case i.e. when there is one e-puck, two e-pucks, three e-pucks and four e-pucks running on the board at one point of time. These results in the table contain number of hops taken, amount of time taken and number of messages sent out for an e-puck to reach the final destination.

It can be observed from the tables that all the algorithms take the same number of hops to reach the destination. The number of hops taken to reach the destination remain same even when the number of e-pucks running on the board increase. Therefore in terms of hops, all the three algorithms have the same performance. It can also be observed that the time taken to reach the destination is almost the same for next node algorithm and available path algorithm except that available path algorithm takes a little more time as it has to reserve the whole path. But when whole path algorithm is used, it takes a lot of time when there is more than one e-puck running on the test bed at one point of time. If the number of messages sent is considered, then it can be clearly observed that number of messages sent out by available path algorithm is far less than the messages sent out by next node and whole path algorithm.

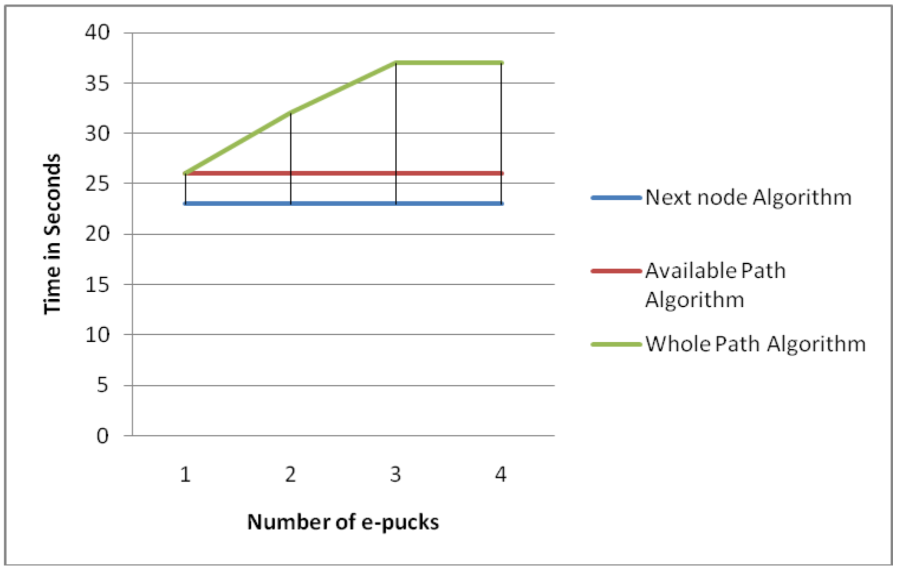

Figure 6-5 Time in Seconds vs Number of e-pucks fro Scenario 2

The figure 6-5 gives the Time vs Number of e-pucks graph of the e-puck which starts at node-2 for all the three algorithms. Clearly from the graph it can be observed that next node algorithm and available path algorithm perform better than the whole path algorithm for this scenario in terms of time taken. Available path algorithm takes a little more time than available path algorithm but much less than whole path algorithm. The time taken by the available algorithm and next node algorithm is almost same even when the number of e-pucks running on the board increase. But the time taken by the e-puck to reach the destination using whole path algorithm increases when the number of e-pucks running on the board increase.

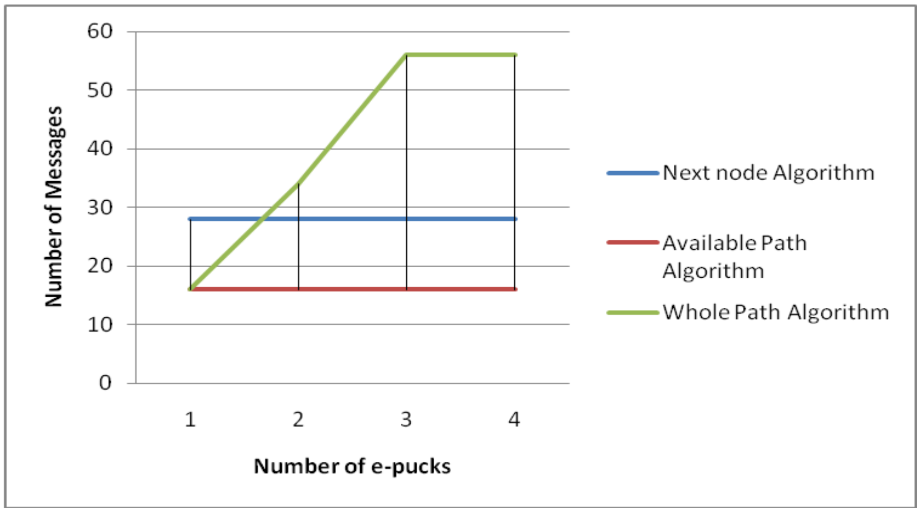

Figure 6-6 Number of Messages vs Number of e-pucks for Scenario 2

The figure 6-6 gives the Number of messages vs Number of e-pucks graph of e-puck which starts at node-2 for all the three algorithms. Clearly from the graph it can be observed that available path algorithm performs better than the other algorithms for this scenario in terms of number of messages sent out. Next node algorithm sends more messages out than available path algorithm but much less than whole path algorithm, when there is more than 1 e-puck running on the board. The number of messages by the available path algorithm and next node algorithm is almost same when the number of e-pucks running on the board increase. But the number of messages sent by the e-puck when using whole path algorithm increases when the number of e-pucks running on the board increase.

The above two scenarios were tested for safety related issues and also for accuracy. The system ensured safety and was accurate for all the scenarios and for different algorithms.

6.2Test with Two Boards

Performance of vehicle highway automation system and the three algorithms is done with two boards together. Two scenarios were considered for testing. Performance analysis of these scenarios is as follows.

6.2.1 First Scenario

In this scenario all the e-pucks have different starting point and have a one common destination. The results from this testing are described and analyzed as follows.

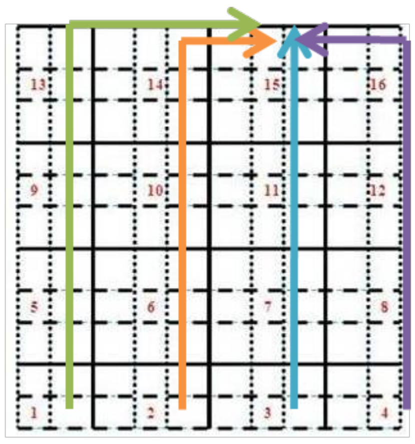

Figure 6-7 First Scenario

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 3 to 15	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12
Path: 2 to 15	Hops: 4 Time: 15 Msg: 16	Hops: 4 Time: 15 Msg: 16	Hops: 4 Time: 15 Msg: 16	Hops: 4 Time: 15 Msg: 16
Path: 1 to 15	Hops: 5 Time: 19 Msg: 20	Hops: 5 Time: 19 Msg: 20	Hops: 7 Time: 27 Msg: 29	Hops: 7 Time: 27 Msg: 29
Path: 4 to 15	Hops: 4 Time: 15 Msg: 16	Hops: 4 Time: 16 Msg: 16	Hops: 4 Time: 17 Msg: 16	Hops: 6 Time: 24 Msg: 28

Table 6-7 Next Node Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 3 to 15	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 2 to 15	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 17 Msg: 10
Path: 1 to 15	Hops: 5 Time: 20 Msg: 12	Hops: 5 Time: 20 Msg: 12	Hops: 5 Time: 20 Msg: 18	Hops: 5 Time: 23 Msg: 18
Path: 4 to 15	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 21 Msg: 16	Hops: 4 Time: 21 Msg: 16

Table 6-8 Available Path Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 3 to 15	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 2 to 15	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 30 Msg: 34	Hops: 4 Time: 30 Msg: 34	Hops: 4 Time: 30 Msg: 34
Path: 1 to 15	Hops: 5 Time: 20 Msg: 12	Hops: 5 Time: 34 Msg: 42	Hops: 5 Time: 51 Msg: 92	Hops: 5 Time: 64 Msg: 92
Path: 4 to 15	Hops: 4 Time: 17 Msg: 10	Hops: 4 Time: 30 Msg: 34	Hops: 4 Time: 51 Msg: 92	Hops: 4 Time: 68 Msg: 130

Table 6-9 Whole Path Algorithm

The figure 6-7 shows the scenario where all the e-pucks have a different starting point and a common destination i.e. node 15. The e-puck at node 3 starts first and then e-puck at node 2, node 1 and node 4 start. This scenario is tested with all the three algorithms. The results obtained when running these algorithms are available in Table 6-7, Table 6-8 and Table 6-9 for next node, available path and whole path algorithms respectively. The results are obtained separately for each case i.e. when there is one e-puck or two e-pucks or three e-pucks or four e-pucks running on the board at one point of time. These results in the table contains number of hops taken, amount of time taken and number of messages sent out for an e-puck to reach to the final destination.

It can be observed from the tables that all the algorithms take almost the same number of hops to reach the destination except the next node algorithm takes more number of hops to reach destination when all the four e-pucks are running at a time on board. It can also be observed that the time taken to reach the destination is almost the same for next node algorithm and available path algorithm except that next node algorithm takes little more time when there are all the four e-pucks running on the board at a time. But when whole path algorithm is used, it takes a lot of time when there are more than one e-puck running on the test bed at a time. If the number of messages sent is considered then it can be clearly observed that number of messages sent out by available path algorithm is far less than the messages sent out by next node and whole path algorithm.

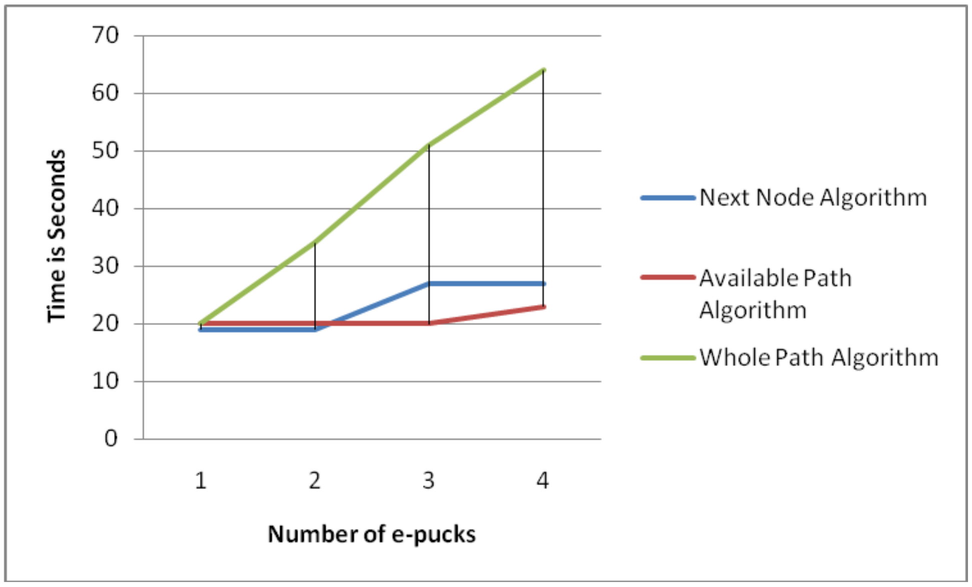

Figure 6-8 Time in seconds vs Number of e-pucks for Scenario 1

The figure 6-10 gives the Time vs Number of e-pucks graph of e-puck which starts at node-1 for all the three algorithms. Clearly from the graph it can be observed that available path algorithm performs better than the other algorithms for this scenario in terms of time taken. Next node algorithm takes a little more time than available path algorithm but much less than whole path algorithm. The time taken by the available algorithm is almost same as the number of e-pucks running on the board increases, but the time taken by the next node algorithm and whole path algorithm increases when e-pucks running on the board increases.

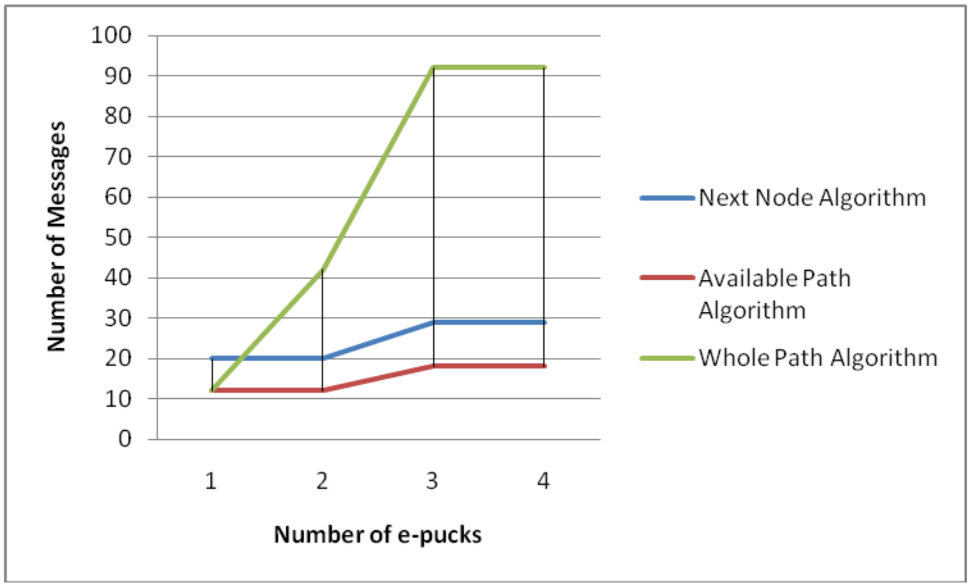

Figure 6-9 Number of messages vs Number of e-pucks for Scenario 1

The figure 6-11 gives the Number of messages vs Number of e-pucks graph of e-puck which starts at node-1 for all the three algorithms. Clearly from the graph it can be observed that available path algorithm performs better than the other algorithms in this scenario in terms of number of messages sent out. Next node algorithm sends out more messages than available path algorithm but much less than the whole path algorithm. The number of messages by the available path algorithm and for the next node algorithm is almost the same when the number of e-pucks running on the board increase. But the number of messages sent by the whole path algorithm increase when the number of e-pucks running on the board increase.

6.2.2 Second Scenario

In this scenario all the e-pucks have different starting point and have different destination and move perpendicular to each other. The results from this testing are described and analyzed as follows.

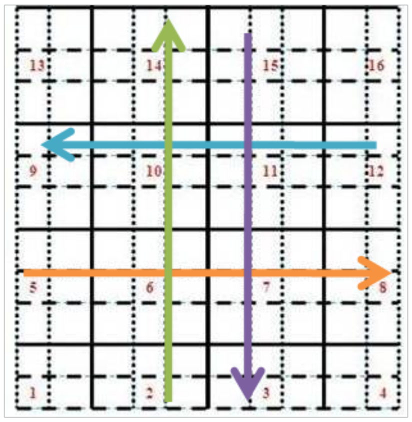

Figure 6-10 Second Scenario

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 12 to 9	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12
Path: 5 to 8	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12	Hops: 3 Time: 12 Msg: 12
Path: 2 to 14	Hops: 3 Time: 12 Msg: 12	Hops: 5 Time: 21 Msg: 22	Hops: 5 Time: 21 Msg: 24	Hops: 5 Time: 21 Msg: 24
Path: 15 to 3	Hops: 3 Time: 12 Msg: 12	Hops: 5 Time: 21 Msg: 22	Hops: 5 Time: 21 Msg: 24	Hops: 5 Time: 21 Msg: 24

Table 6-10 Next Node Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 12 to 9	**Hops: 3 Time: 13 Msg: 8**	**Hops: 3 Time: 13 Msg: 8**	**Hops: 3 Time: 13 Msg: 8**	**Hops: 3 Time: 13 Msg: 8**
Path: 5 to 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 2 to 14	Hops: 3 Time: 13 Msg: 8	Hops: 5 Time: 20 Msg: 14	Hops: 5 Time: 20 Msg: 16	Hops: 5 Time: 20 Msg: 16
Path: 15 to 3	Hops: 3 Time: 13 Msg: 8	Hops: 5 Time: 20 Msg: 14	Hops: 5 Time: 20 Msg: 16	Hops: 5 Time: 20 Msg: 16

Table 6-11Available Path Algorithm

E-puck / no of epuck	**1 – epuck**	**2 – epuck**	**3 – epuck**	**4 – epuck**
Path: 12 to 9	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 5 to 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8	Hops: 3 Time: 13 Msg: 8
Path: 2 to 14	Hops: 3 Time: 13 Msg: 8	Hops: 5 Time: 24 Msg: 20	Hops: 5 Time: 32 Msg: 32	Hops: 5 Time: 32 Msg: 32
Path: 15 to 3	Hops: 3 Time: 13 Msg: 8	Hops: 5 Time: 24 Msg: 20	Hops: 5 Time: 32 Msg: 32	Hops: 5 Time: 32 Msg: 32

Table 6-12 Whole Path Algorithm

The figure 6-8 shows the scenario where all the e-pucks have a different starting point and different destinations and move in perpendicular directions to each other. The e-puck at node 12 starts first and then e-puck at node 5, node 2 and node 15 starts. This scenario is tested with all the three algorithms. The results obtained when running these algorithms are available in Table 6-12, Table 6-13 and Table 6-14 for next node, available path and whole path algorithms respectively. The results are obtained separately for each case i.e. when there is one e-puck, two e-pucks, three e-pucks and four e-pucks running on the board at one point of time. These results in the table contain number of hops taken, amount of time taken and number of messages sent out for an e-puck to reach the final destination.

It can be observed from the tables that all the algorithms take the same number of hops to reach the destination. The number of hops taken to reach the destination increase when there is more than one e-puck running on the board. Therefore in terms of hops, all the three algorithms have the same performance. It can also be observed that the time taken to reach the destination is almost the same for next node algorithm and available path algorithm. But when whole path algorithm is used, it takes a lot of time when there are more than one e-pucks running on the test bed at one point of time. If the number of messages sent is considered then it can be clearly observed that number of messages sent out by available path algorithm is less than the messages sent out by next node and whole path algorithm.

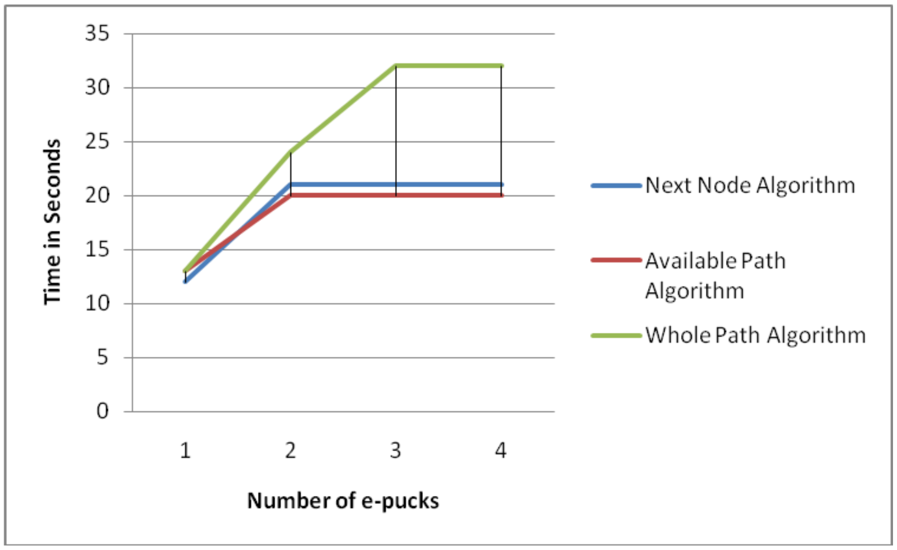

Figure 6-11 Time in seconds vs Number of e-pucks for Scenario 2

The figure 6-9 gives the Time vs Number of e-pucks graph of e-puck which starts at node-2 for all the three algorithms. Clearly from the graph it can be observed that next node algorithm and available path algorithm perform better than the whole path algorithm in this scenario in terms of time taken. Available path algorithm takes a little more time than available path algorithm but much less than the whole path algorithm. The time taken by the available algorithm and next node algorithm is almost same even when the number of e-pucks running on the board increase. But the time taken by the e-puck to reach the destination using whole path algorithm increases when the number of e-pucks running on the board increase.

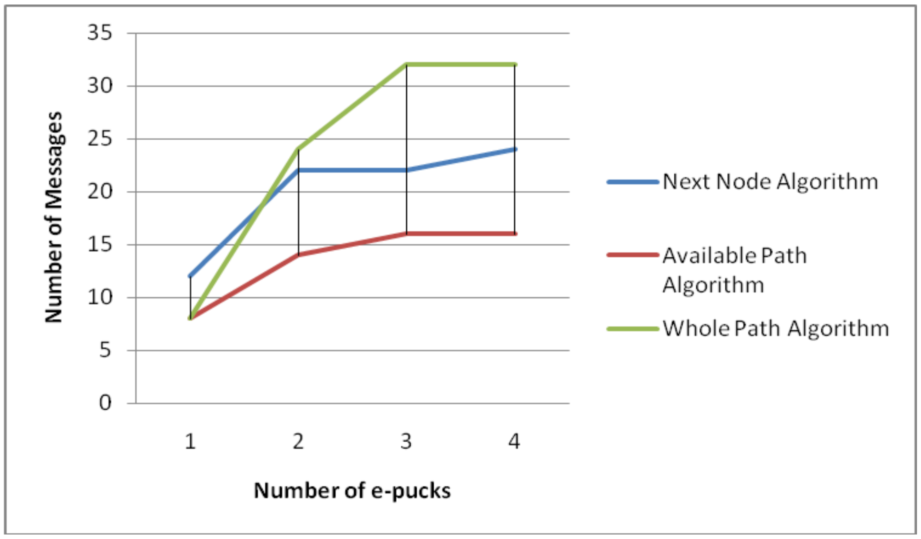

Figure 6-12 Number of messages vs Number of e-pucks for Scenario 2

The figure 6-10 gives the Number of messages vs Number of e-pucks graph of e-puck which starts at node-2 for all the three algorithms. Clearly from the graph it can be observed that available path algorithm performs better than the other algorithms in this scenario in terms of number of messages sent out. Next node algorithm sends out more messages than available path algorithm but much less than whole path algorithm when there is more than 1 e-puck running on the board. The number of messages by the available path algorithm is almost the same when the number of e-pucks running on the board increase. But the number of messages sent by the e-puck when using next node algorithm and whole path algorithm increase when the number of e-pucks running on the board increase.

The above two scenarios were tested for safety related issues and also for accuracy. The system ensured safety and was accurate for all the scenarios and for different algorithms.

CHAPTER 7 - Conclusion

This thesis work introduced a new Co-Operative Vehicle-Highway Automation system for fully automating a vehicle on a highway, which is cost effective and near to real life implementation. A test bed is created to simulate the real highway scenario and for implementing and testing this Vehicle-Highway Automation system.

The Co-Operative Vehicle Highway Automation system has many characteristics. These characteristics i.e. obstacle avoidance, vehicle automation, collision prevention and avoidance, route guidance, vehicle tracking, decision support system, etc., were developed and implemented successfully on a test bed.

In this thesis work three different algorithms i.e. next node algorithm, available path algorithm and whole path algorithm are developed to automate the vehicle to move to the desired destination effectively, accurately and safely. Performance analysis of these algorithms have been done on the test bed and was concluded that available path algorithm is more effective in terms of time taken, number of messages sent out and number of hops taken to reach the destination.

The results from performance analysis of Vehicle-Highway Automation system indicated that the system achieved fully automation of vehicle effectively and ensured safety for all the scenarios it was tested for.

CHAPTER 8 - Future Work

Co-Operative Vehicle-Highway Automation System has other characteristics that can be implemented in future. These are described as follows.

1. **Traffic Surveillance:** One of the characteristics of Vehicle-Highway automation system is traffic surveillance. This traffic surveillance characteristic is not implemented in this system and can be implemented in future.
2. **Priority reservation of path:** In this thesis work of Co-Operative Vehicle-Highway Automation system, paths are reserved without any priority. In future, reservations of the path can be made with priorities. This will help emergency vehicles to reach destination faster.
3. **Extending to Multi lane Highway:** In this thesis work the Co-Operative Vehicle-Highway Automation system is implemented for only one lane highway. This implementation can be extended to multi lane highway in the future.

References

[1] TinyOS
http://www.tinyos.net/

[2] TinyOS 1.x tutorial.
http://www.tinyos.net/tinyos-1.x/doc/tutorial/

[3] nesC
http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf

[4] E-puck
http://www.e-puck.org/

[5] E-puck wiki
http://www.e-puck.org/index.php?option=com_openwiki&Itemid=81

[6] Rodney Lay, Lyle Saxton, “Vehicle-Highway Automation Directions, Challenges, and Contributing Factors”, Transportation in New Millennium, 2000.