ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

MULTI-AGENT BASED LARGE SCALE TRAFFIC FLOW SIMULATION OF INTELLIGENT TRANSPORTATION SYSTEMS

M.Sc. THESIS

Oğuz Ali Ekinci

Department of Mechanical Engineering

Mechatronics Engineering Programme

JUNE 2013

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

MULTI-AGENT BASED LARGE SCALE TRAFFIC FLOW SIMULATION OF INTELLIGENT TRANSPORTATION SYSTEMS

M.Sc. THESIS

Oğuz Ali EKİNCİ
(518091056)

Department of Mechanical Engineering

Mechatronics Engineering Programme

Thesis Advisor: Asst. Prof. Pınar BOYRAZ

JUNE 2013

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AKILLI TAŞIT SİSTEMLERİNDE TRAFİK AKIŞININ ÇOKLU AJAN YAKLAŞIMIYLA BÜYÜK ÖLÇEKTE BENZETİMİ

YÜKSEK LİSANS TEZİ

Oğuz Ali EKİNCİ
(518091056)

Makina Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

Tez Danışmanı: Yrd. Doç. Dr. Pınar BOYRAZ

HAZİRAN 2013

Oğuz Ali Ekinci, a M.Sc. student of ITU Graduate School of Science Engineering and Technology student ID 518091056, successfully defended the thesis/dissertation entitled "MULTI-AGENT BASED LARGE SCALE TRAFFIC FLOW SIMULATION OF INTELLIGENT TRANSPORTATION SYSTEMS", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Dr. Pınar BOYRAZ** İstanbul Technical University	.....
Jury Members :	**Prof. Dr. Şeniz ERTUĞRUL** İstanbul Technical University	.....
	Asst. Prof. Dr. Mustafa F. SERİNCAN İstanbul Bilgi University	.....

Date of Submission : 3 May 2013
Date of Defense : 5 June 2013

To my parents,

FOREWORD

Firstly, I would like to thank all people who have helped and inspired me during my M.Sc. study. I especially want to thank my supervisor Asst. Prof. Pınar Boyraz for her guidance and endless support during this thesis.
I am thankful and dedicate my thesis to my family, who are the references of my accomplishments. Their support has encouraged me to work hard and reach the best in my life.

May 2013 Oğuz Ali EKİNCİ

TABLE OF CONTENTS

Page

ABBREVIATIONS

ABLE	**:** Agent Building and Learning Environment
ACL	**:** Agent Communication Language
AID	**:** Agent Identifier
AMS	**:** Agent Management System
DARPA	**:** Defense Advanced Research Projects Agency
DF	**:** Directory Facilitator
ETC	**:** Electronic Tool Collection
FIBA	**:** The Foundation for Intelligent Physical Agents
IA	**:** Intelligent Agent
IDE	**:** Integrated Development Environment
IEEE	**:** The Institute of Electrical and Electronics Engineers
IP	**:** Internet Protocol
ITS	**:** Intelligent Transportation Systems
JADE	**:** Java Agent Development Environment
JAVA SE	**:** Java Standart Edition
JDK	**:** Java Development Kit
JVM	**:** Java Virtual Machine
MAS	**:** Multi-Agent System
MATLAB	**:** Matrix Labrotary
SQL	**:** Structured Query Language
V2I	**:** Vehicle to Infrastructure
V2V	**:** Vehicle to Vehicle

LIST OF TABLES

Page

LIST OF FIGURES

Page

MULTI-AGENT BASED LARGE SCALE TRAFFIC FLOW SIMULATION OF INTELLIGENT TRANSPORTATION SYSTEMS

SUMMARY

In modern urban life, automobile traffic and collisions lead to endless frustration as well as significant loss of life, time, and productivity. Recent advances in artificial intelligence suggest that autonomous vehicles may soon be a reality. There are many studies on intelligent transportation systems including autonomous vehicles, intelligent routing, and intelligent road infrastructure. These approaches alleviate many traditional problems associated with human inattention, in terms of both safety and efficiency. However, all these systems rely on all vehicles being equipped with the same technology and all of them are autonomous vehicles. Implementing such systems in the real world is extremely difficult, there would be a transition period and this period may be a chaotic or well organized. To observe that intelligent transportation systems would help to solve traffic congestion or improve safely travelling, this transition period must be analyzed.

In this study, we dwell on a simulation to allow autonomous vehicles and manually driven vehicles in same traffic environment. The developed software platform is able to simulate urban traffic with different proportions of autonomous vehicles over manually driven vehicles. Also manually driven vehicles are considerad as three main groups, aggressive drivers, cautious drivers and delayed drivers. All vehicles are defined as software agents, to gain decision-making capabilities and unpredictable behaviors in traffic. All vehicles calculate their own routes, make their own chooses and follow their own paths with the help of multi agent approach. An agent has specific properties according to limitations like its average velocity and driver skill level.

The work is fully implemented and tested in developed custom simulator, and we present detailed experimental results attesting to its effectiveness. Furthermore, we show that as the number of autonomous vehicles on the road increases, they travel in traffic more comfortable, faster, safer and delays caused by traffic jam decrease monotonically.

AKILLI TAŞIT SİSTEMLERİNDE TRAFİK AKIŞININ ÇOKLU AJAN YAKLAŞIMIYLA BÜYÜK ÖLÇEKTE BENZETİMİ

ÖZET

Kentsel yaşamın günümüzde en temel problemlerinden biri yoğun trafik, trafikte meydana gelen kazalar ve trafikte yaşanan zaman, can ve mal kaybıdır. Şehir yaşantısında herkes yoğunluktan, ve bu yoğunluğa bağlı olarak ulaşımdaki zorluklardan şikayetçidir. Nüfustaki yoğunluktan ve özellikle mesai öncesi ve sonrası oluşan uzun kuyruklu trafik bir çoğumuzun zamanını çalmakta ve yaşam kalitemizi önemli ölçüde düşürmektedir. Zaman kaybının yanı sıra, trafikte geçirilen uzun bekleyişlerden sonra sürücülerin dikkatinin dağılmasından yada sabırsız davranışlarından kaynaklanan can ve mal kayıpları da olabilmektedir.

Elektronik, yazılım, mekanik ve kontrol alanlarındaki ilerlemeler sayesinde sürücüsüz araçlar üzerinde yapılan araştırmalar hız kazandı. İlk olarak sürücüye yardımcı olan sistemler gelişmeye başladı. Hız sabitleme sistemleri, erken uyarı sistemleri, sürüş kontrol sistemleri ve otomatik park sistemleri araçlara entegre olan ilk akıllı teknolojilerdi. Bu teknolojiler sayesinde yarı-otonom araçlar gündeme geldi. Özellikle karar verme konusunda yardımcı olabilecek tüm sistemler yarı-otonom araç teknolojisine hizmet etmektedir.

Karar destek mekanizmalarının yanında kendi kendine karar verebilen teknolojiler de gelişmeye başlamıştır. Böylece tam otonom araçlar gündeme gelmiştir. Tam otonom araçlar yüksek teknolojiyle donatılmış, kendi kendine seyredebilen araçlardır. Üzerinde konum belirleyiciler, ivme ölçerler, akustik algılayılar, mesafe sensörleri gibi birçok sensörü barındıran ve bu bilgileri kullanarak araç kontrolünü sağlayan yapay zeka ile donatılmışlardır.

Otonom araçların elektromekanik araştırmalarının yanında, otonom araçların trafikle olan iletişimi üzerine de birçok çalışma yapılmıştır. Dinamik olarak değişen şehir trafiğinde anlık bilgiye ulaşmak ve bunu yorumlamak çok önemlidir. Özellikle akıllı taşıt sistemlerinin ve ulaştırma mühendislerinin üzerinde durduğu konular araçların trafikte optimum şekilde seyretmesidir. Otonom araçlar kadar trafikte araçların yönlendirilmesi de önemlidir.

Bahsedilen bütün teknolojiler, Akıllı Ulaşım Sistemleri adı altında toplanmaktadır, ve hepsi aynı amaca hizmet etmektedir; trafik kazalarını ve buna bağlı can ve mal kaybını önlemek, trafikteki seyir rahatlığını arttırmak, araç kullanım sayısını ve yakıt tüketimini azaltarak karbondioksit salınımını engellemektir.

Bu gelişmelerden de anlaşıldığı gibi yakın gelecekte otonom araçların şehir trafiğine karışacaklarını düşünmek hayal olmayacaktır. Google'ın üzerinde çalıştığı insansız araçlar (Toyota Prius) Amerika'nın Nevada eyaletinde ehliyetlerini almışlardır ve prototipler şehir trafiğinde aktif olarak seyir etmektedir. Projenin gidişatına bakılırsa otonom araçların amaca yönelik olarak insanlara trafikte büyük katkıda bulundukları görülmektedir.

Gün geçtikçe otonom araçların sayısının artacağı düşünülüyor, ancak otonom araçlar trafiği insan sürücülerle paylaşmak zorundalar ve bu beklenmedik sonuçlar doğurabilir. Karmaşaya benzeyen trafik ortamına daha da fazla karışıklık getirebilirler. O nedenle otonom araçlarının sayılarının artmasıyla birlikte, trafikteki etkileri analiz edilmelidir. Insanlar öyle bir noktaya geleceklerdir ki, otonom araç almanın ne zaman mantıklı olacağını düşüneceklerdir. Bu durumu yaşamadan önce benzetim programlarıyla bu durum analiz edilmelidir.

Trafiği simule etmek için birçok yazılım geliştirilmiştir. Ancak bu yazılımların çoğu tüm araçların aynı karakterde, aynı teknolojiyle donatılmış ve hepsinin otonom olduğu varsayılan benzetim ortamlarıdır. Aynı davranışı sergileyen, aynı şekilde seyir eden araçların paylaştığı bir trafk ortamı simule edilmeketedir. Ancak böyle bir trafikte araçları otnom olanlar ve otonom olmayanlar diye ikiye ayırmak yeterli olmayacaktır.Ayrıca otonom araçların sayısı trafik ortamında dün geçtikçe artan bir orana sahip olacaktır.

Geçiş dönemini simule etmek için trafik ortamına insan sürücüleri de dahil etmek gerekir. Çünkü trafikte beklenmeyen davranışları insan sürücüler sergilemektedir. Bazıları trafik kurallarına harfiyen uyar, bazıları uymak istemelerine ragmen sürücülük yetenekleri zayıf olduğu için uyamazlar ve bazıları da uymak istemeyebilir. Bu nedenle insan davranışlarının modellenmesi ve benzetim ortamına alınması büyük önem arzetmektedir.

Bu çalışmada, otonom ve manual araçların aynı trafikte olduğu bir benzetim üzerine durulmuştur. Geliştirilen yazılım platformu farklı oranlarda manual ve otonom araçların aynı şehir trafiğinde benzetilmesini sağlamaktadır. Manual kullanılan araçlar, agresif sürücü, dikkatli sürücü ve gecikmeli sürücü olmak üzere üç farklı grupta ele alınmıştır.Tüm araçlar trafikte kendi karalarını verebilmesi ve tahmin edilemeyen davranışalara sahip olabilmeleri için birer yazılım vekili olarak tanımlanmıştır. Vekil yaklaşımıyla, tüm araçlar kendi rotalarını hesaplayabilir, kendi tercihlerini yapabilir ve rotalarını takip edebilirler. Her vekilin kendine özgü hız limiti, ortalama hızı ve sürücü kabiliyeti gibi özellikleri vardır.

Benzetim ortamı oluşturulmak için ağırlıklı olarak Java programlama dili kullanılmıştır. Eclipse yardımıyla Java kodu geliştirilip farklı ortamalarla beraber çalışması sağlanmıştır. Şehir haritası oluşturmak için MATLAB ortamı kullanılmış ve Java yazılımıyla control edilmiştir. Özellikle haritanın MATLAB'te oluşturulmasının sebebi araçların rotalanması ve bu rotaların optimize edilmesi konusunda MATLAB'in güçlü bir ortam olduğu içindir. MATLAB'ın farklı fonksiyonları kullanılarak farklı boyutlarda yapay şehir haritaları oluşturulmuş, karmaşıklığı ise simule edilmek istene duruma göre değiştirilmiştir. İlk testlerin yapılması için küçük haritalar kullanılmıştır. Haritanın üzerinde araçların tanımlayabilmek ve ilk davranışlarını test etmek için yaklaşık 25 aracın rahat hareket edebileceği test haritaları kullanılmış, araçlar programlandıktan sonra ise harita kademeli olarak büyütülmüştür.

Harita üzerinde otonom ve manuel olmak üzere iki temel araç tipi oluşturulmuştur. Her bir araç bir özerk vekil olarak tanımlanmış ve trafikteki normal birer sürücü gibi davranmaları için programlanmışlardır. Otonom araçların karakteristikleri birbirinin aynısıdır, ancak manuel sürücüler; agresif sürücüler, dikkatli sürücüler ve zayıf tepkili sürücüler olmak üzere üç temel gruba ayrılmıştır.

Akıllı vekiller (araçlar) JADE platformunda oluşturulmuş ve hepsine yol takip davranışları, hız profilleri ve tepki süreleri gibi sürücü davranışları kazandırılmıştır.

Bu sürücü davranışları belli toleranslar içinde değişkenlik gösterebilmektedir. Örneğin, agresif sürücüler dikkatli sürücülere nazaran daha yüksek hız profillerine sahiptir ve kendi içlerinde de bir miktar değişiklik gösterebilir. Bu değişiklikler tamamen rastlantısal olarak oluşturulmaktadır.

JADE ortamında programlanan tüm araçlar, verdikleri kararları, aldıkları yol durumlarını, seçtikleri hızları v bu gibi sürücü davranışlarını MySQL veri tabanına kaydetmişlerdir. Bu kayıtlar üzerinden sorgular oluşturularak trafikteki durum analiz edilmeye çalışılmıştır. Bulunan sonuçlar detaylı bir şekilde incelenmiş ve sonuç olarak, otonom araçların sayısı arttıkça trafikteki seyirleri daha rahat, hızlı ve güvenli hale gelmiştir ve trafik sıkışıklığından kaynaklan gecikmeler azalmıştır.

1. INTRODUCTION

Nowadays, traffic congestion is one of the main problems for all countries in the world, especially in developing countries because of the rapid increase in urban population and the number of cars. This brought about the increase in number of traffic accidents and has negative impact on society and human life directly because of time and energy loss in traffic. This might also point to an inefficient infrastructure and traffic control strategy.

Human behavior remains the main factor in determining the number of traffic accidents and transportation trends even in the modern times. To minimize traffic accidents there are established traffic rules and signals to guide drivers for organized traffic flow and these rules also limit human behaviors while guiding them. With these limitations, in the current system, assumed that all humans behave in same driver profile and drives safely. However, this is not necessarily true.

It is not so easy to limit people behaviors especially in a chaotic situation like traffic jam; also, characteristics of human behavior may vary both in normal and stressful conditions. This causes more complexity in traffic flow. Due to the variation amongst the driver behavior and human errors, the safety on roads still depends on the human behavior no matter how advanced are the safety systems in today's vehicles.

In order to reduce human related errors and misconducts in traffic, there are suggested improvements in vehicle technology such as driver assistant systems. The degree of the assistance may range from vehicles with high technology controller units to more sensitive warning systems. However, this is not adequate to prevent or eliminate the effect of human behavior on traffic flow.

Autonomous ground vehicles are one-step further when compared to driver assistant systems and autopilot drivers. . An autonomous ground vehicle is a vehicle that navigates and drives entirely on its own with no human driver intervention and no remote control. Using various sensors and positioning systems, the vehicle senses all

the characteristics of outer environment and its own dynamics such as acceleration, velocity and position.

At the extreme end of the spectrum for intelligent vehicles, the research area in autonomous vehicles is getting more interest. For instance, one of the latest developments of autonomous vehicles is Google's self-driving car [1], which is modification of a Toyota Prius. The car is a project of Google Inc., which has been working in secret but in plain view on vehicles that can drive themselves, using artificial-intelligence software that can sense anything near the car and mimic the decisions made by a human driver.

In this project, seven test cars have driven more than 1000 miles (about 1600 Kilometer) in autonomous mode without human intervention. In addition, there were experienced technicians in driver and passenger seats and ready for action if necessary. Test cars have driven in highways and sometimes in crowded city traffic. The only accident while testing, engineers said, was when one Google car was rear-ended while stopped at a traffic light by an elderly driver.

Autopilots react faster than human drivers do; they have 360-degree perception capabilities and do not get distracted, sleepy or intoxicated when compared with human drivers. These technologies could double the capacity of roads by allowing cars to drive more safely while closer together like queuing and travelling together. Because the robot cars would eventually be less likely to crash, they could be built lighter, reducing fuel consumption. However, to be truly safer, the cars must be far more reliable than first prototypes.

The car may be programmed for different driving personalities from cautious, in which it is more likely to yield to another car, to aggressive, where it is more likely to go first.

In urban traffic tests of Google's car, Carnegie Mellon University robotics scientist, Christopher Urmson was in the driver seat but not driving it. To gain control of the travelling car, driver has to do one of three things: hit a red button near the right hand, touch the brake or turn the steering wheel. He had this situation two times, once when a bicyclist ran a red light and again when a car in front stopped and began to back into a parking space. However, the car seemed likely to have prevented an accident itself.

The project is the innovation of Sebastian Thrun, the director of the Stanford Artificial Intelligence Laboratory, a Google engineer and the co-inventor of the Street View mapping service of Google Maps. He announced the project in 2010 and the newbie autonomous vehicle program recently passed its driver's license test in Nevada in May 2012, the first license of its kind in the United States.

This is just a beginning, there are more researches about autonomous vehicles, and there was a prize competition for driverless cars named DARPA Urban Challenge in 2007 [2].

The DARPA Urban Challenge was an autonomous vehicle research and development program. This program's goal was to spur the development of technologies needed to create the first fully autonomous ground vehicles capable of completing a substantial off-road course within a limited time. The Urban Challenge features autonomous ground vehicles maneuvering in a mock city environment, avoiding obstacles, executing simulated military supply missions while merging into real traffic.

The program is managed as a series of qualification steps leading to a competitive final event. DARPA is offering monetary reward of $2M for the fastest qualifying vehicle, and $1M and $500,000 for second and third place.

All Urban Challenge teams come from across the United States and around the world, and have a passion for the advancement of machine intelligence and robotic technology. This multi-disciplinary group includes teams from different research areas like academia, the robotics, automotive, and defense industries. Each team works to develop an autonomous vehicle to complete the 60-miles (approximately 100 Kilometers) urban route in less than six hours.

As mentioned before, this technology interests the robotics as well, and some researchers has been working on it. Daniela Rus, Professor of Electrical Engineering and Computer Science and Director of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT publish one of the latest papers related with this topic.

Control theory is another approach for traffic congestion and distributed control approach is applicable on autonomous vehicles and their optimal routing. A study proposes a method for multi-agent path planning on a road network in the presence of congestion and a distributed method to find paths [3] for multiple agents is

suggested. This distributed method uses probabilistic path choice to achieve global goals like social optimal. Proposed approach shows that the global goals can be achieved by local processing using only local information, can be parallelized and speed up using parallel processing. This study implemented on a group of robots and it was tested in laboratory environment to coordinate their movements. It is a good approach, but a group of robots would not enough to simulate real, crowded urban traffic. In macro scale, it is a very efficient work.

There are many studies on intelligent traffic routing to reduce traffic jam and it is one the most important technology. This technology may be based upon different route traffic data. One of the work is about using loop detectors and a fleet of taxis to gather real traffic data to develop a traffic congestion model. Two different goals are considered, one of them is to optimize individual travel times and the other is to achieve the social optimum regarding to travel time over all the drivers in the system. Using this approach, the study [4] claims to improve the total travel time by 15%. However, the method could be applicable to only limited situations where there is a distributed sensor network infrastructure. Gathering traffic data from specific group of vehicles my not reflect the real traffic congestion, this system must be implemented all routes and all vehicles in traffic. Moreover, it is difficult to implement in real life.

It is hard to develop a solution to complex traffic environment, but testing this solution or model is an important step to develop insight on the feasibility of the proposed solution. It would not be safe directly implementing solutions; first, the system should be simulated to see the pitfalls. That is why we need a specific simulation environment for modelling real traffic environment. In this work [5], it is analyzed that which simulators is suitable for implementing autonomous vehicles.

Primarily, robotics simulators provide a suitable test environment for autonomous vehicles to experiment new methodologies such as long-term navigation algorithms, intelligent routing etc. However, when it concerns the deployment and validation of such vehicles in a larger scale urban traffic scenario, robotics simulators do not seem to provide the required functionality for road traffic analysis, or inter-vehicular communication infrastructure, as they seem present in today's traffic simulators. These features have key importance in simulation environment to assess the real urban traffic when an intelligent system is introduced. José Luis Ferrás Pereira

implemented autonomous vehicles in different simulation environments and critically reviewed the feasibility of the integrations.

There are many projects in the area autonomous vehicles and their implementations in simulation environments, here only a few of them are mentioned. As technologists said, autonomous ground vehicles are inevitable in the future and they may participate in real city traffic in next 20-30 years. In addition, there will be a transition period for switching from manual vehicles to autonomous vehicles, and this period should be analyzed in large scale in terms of feasibility. This period will be chaotic, or steadier than expected depending on how the transition is planned. To the best of knowledge of the author, there is no study on mixed traffic flow including different autonomy levels of vehicles and ITS in simulation or real traffic environment. Google's Toyota Prius is travelling in real traffic but if the numbers of autonomous vehicles are increased to form a certain percentage of the existing fleet, it is not clear that, what will be the portrait of the traffic flow or if they are as effective as they are claimed to be in reducing the number of accidents.

In addition, there must be a clue for customers who consider having an autonomous car. When will be this purchase is safer, or with rise of autonomous cars, will traffic jam reduce.

This thesis aims to analyze the mixed traffic when autonomous vehicles meet manually driven vehicles in urban traffic. This must be accomplished using a simulation environment, where both manually driven vehicles and autonomous vehicles can be defined at the same time. Ultimate goal of the project is developing a successful simulation environment to simulate population of different vehicle groups and analyze the mixed urban traffic. The selected approach envisions the integration of different software environments for simulation, visualization and logging.

1.1 Purpose of Thesis

This thesis tries to predict or develop insight on answering the question if the ITS technology will help solving congestion/traffic jam and accident problems in near future as expected. We propose an ITS structure to see if such a technology would be beneficial in transitory period where some certain percentages of vehicles are fully autonomous while the rest of the fleet is semi-autonomous or manually controlled.

2. BACKGROUND

2.1 Intelligent Transportation Systems (ITS)

Intelligent Transportation System (ITS) [6] applies advanced technologies of electronics, communications, artificial intelligence, sensing and control in all fields of transportation in order to improve driver and pedestrian safety, driving comfort and real-time information sharing. Intelligent Transportation System is architecture for both commercial users and the public.

ITS Technologies are applied to roadways and vehicles to perform navigation, sensing, monitoring, communications, data processing, traffic control, surveillance and various other functionalities. These functionalities are just in their beginning stages. After autonomous cars rise, there will be new and more effective functions like intelligent routing, accident warning systems, traffic flow optimization etc. Moreover, ITS technology will be more valuable while world population grows.

Several ITS components that are actively used in developed countries today, although they are not actively connected to each other to form a super-system of ITS yet:

- Electronic toll collections (ETC) [7]
- Freeway and incident Management [8]
- Vehicle to Vehicle (V2V) [9] [10] or Vehicle to Infrastructure (V2I) [11]
- Intelligent Routing [12] [13]
- Traffic density monitoring etc. [14] [15] [16]

In the next section, the route finding algorithms are reviewed since they are the backbones of the intelligent transportation systems providing an optimization for routing of the vehicles in a complex net of roads with possible bottlenecks and incomplete information on the dynamic situation of the network.

2.2 Route Finding Algorithms Review

A path finding method searches a graph by starting at one vertex and exploring adjacent nodes until the destination node is reached, generally with the intent of finding the shortest route. These algorithms are used in many fields such as mobile robotics, game programming, logistics and similar areas.

These algorithms can also be used in Intelligent Transportation Systems. There are many different graph search and path planning algorithms in computer science literature to be adopted for use in ITS development. Some of them [17] [18] [19] are aiming to find the shortest path between two points in the least possible computation, some other are aiming to find an acceptable path by pruning the related search tree to decrease computation cost in exchange for least possible distance.

In addition, there are more algorithms specific to topological characteristic of road architectures and connections. Likewise, there are heuristic search methods to increase computational efficiency of shortest path algorithms like [21] [22] [23].

It is not easy to use heuristic search methods in vehicle navigation, different heuristic strategies such as limiting the search area, decomposing search problem, limiting the searched links in graph and their combinations must be examined.

In any type of path finding method, the algorithm searches a graph by starting a specific vertex and exploring adjacent nodes until the destination node is reached. Usually this search ends up with possible shortest path. Thought graph searching methods such as a breadth-first search would find a route if given enough computation time, other searching methods, that explore the graph, would head the destination sooner. An analogy would be an individual walking cross the street; instead of examining every possible route in advance, the individual would generally walk in the direction of the destination and only diverge from the path to avoid an obstacle.

There are two primary problems in path finding, one is finding the path between two nodes in a graph and the other one is finding the shortest path. Fundamental algorithms such as breadth-first and depth-first search address the first problem by exhausting all possibilities; starting from the given node, they iterate over all

potential paths until they reach the destination node. On the other hand finding the optimal path is the most complicated problem. The exhaustive approach in this case is known as the Bellman-Ford Algorithm [20], but it is not necessary to examine all possible paths to find the optimal one. Algorithms such as A-Star [21] and Dijkstra [17] strategically eliminate paths, either through heuristics or through dynamic programming.

All path-planning algorithms have same goal, reaching destination point from start point. Moreover, path-planning algorithms generally have similarities on their flow charts; calculations are made recursively until destination point is reached. Figure 2.1 shows the general flow chart of Dijkstra's algorithm.

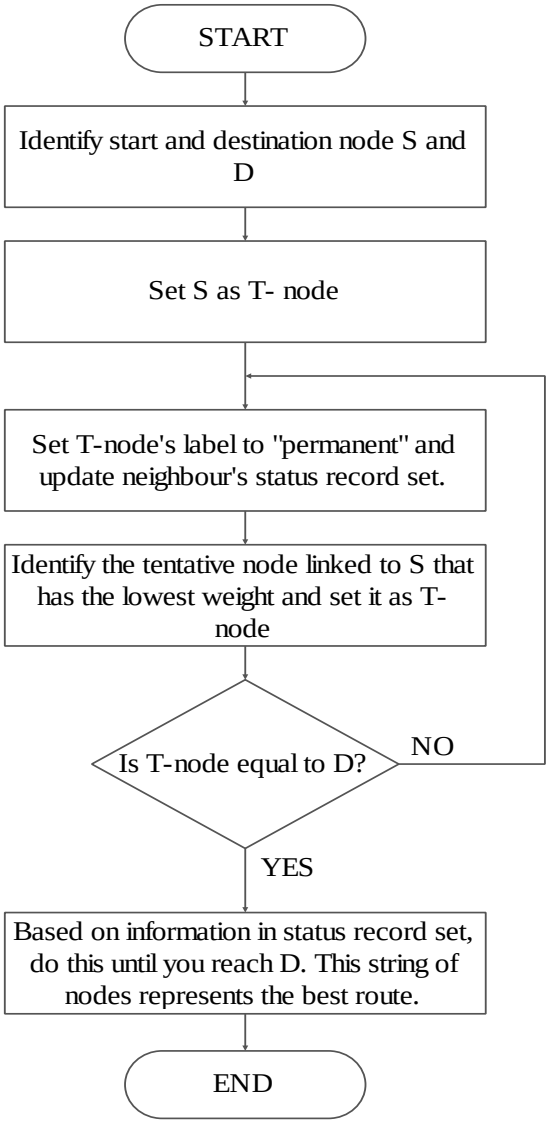

Figure 2.1 : Basic flow chart of Dijkstra's algorithm

2.2.1 Dijkstra's algorithm

A widespread example of graph-based path finding algorithm is Dijkstra's algorithm **[**17**]**. This commonly used algorithm begins with a start node and an "open list" of adjacent candidate nodes. Every step, the node in the open set which is the closest to instant node is examined. The examined node is marked as "closed", and all nodes adjacent to it are added to the open list if they have not already been visited. This cycle continues until a path to the destination has been found. Since the lowest distance nodes are visited first, the first time the destination is found, the path to it will be the shortest path.

Dijkstra's Shortest Path Algorithm is an algorithm to find shortest paths in a graph with weighted edges. It was developed by Dutch computer scientist Edsger Wybe Dijkstra in 1959 and is still widely used today, e.g. in routing and robotics.

Basically it computes the shortest paths from a source node to all reachable nodes. It only requires all edge weights to be positive, and works for undirected as well as directed graphs. If the graph is not connected, i.e. there are unreachable nodes, their distances are set to infinite initially and are marked as not reachable. Algorithm never fails, until weights are negative. Below, the algorithm for a graph **G= (nodes, edges)** is shown in pseudo code[26].

```
1       for each node n in nodes
2               distance[n]:= infinite
3       distance[s]:= 0
4       visited: = {}
5
6       while exist nodes not in visited
7               n: = node with smallest distance[n] which is not in visited
8               if distance[n] = infinite
9                       break
10              visited: = visited union {n}
11
12              for each node m with (n, m) in edges which is not in visited
13                      d: = distance[n] + weight (n, m)
14
15                      if d < distance[m]
16                              distance[m]:= d
```

2.2.2 A-star algorithm

A-Star is one of the general search algorithm which is easy to understand and simple to implement and also competitive and as powerful as other search algorithms [17] [18] [19]. As mentioned, search algorithms have a wide variety of usage ranging from artificial intelligence planning problems to sentence parsing in any language. Therefore, an effective search algorithm provides us to solve a large number of problems easily.

The problems that A-star is best used for are those that can be represented as a state space. In a suitable problem, you must represent the initial conditions of appropriate initial state and the goal conditions as the goal state. For each action, generate successor states to represent the effects of the action. If you continue doing this and at some point, one of the generated successor states is the goal state, and then the path from initial to goal state is the solution of related problem.

A-star is a variant of Dijkstra's algorithm commonly used in game development. A-star designates a weight to each open node equal to the weight of the edge to that node plus the approximate distance between that node and the destination node. This can be classified as a heuristic distance that is the minimum possible distance between that node and the destination node. This allows the algorithm to eliminate longer paths once an initial path is found. If there is a path of length L between the start node and destination node, and the minimum distance between a node and the destination is greater than L, that node need not be visited.

A* uses this heuristic to improve on the behavior according to Dijkstra's algorithm. When the heuristic equals to zero, A* generates same solution as Dijkstra's algorithm. As the heuristic approximate to the real distance, A* still finds optimal path, but computes faster. While the value of the heuristic is exactly the real distance, A* visits the fewest nodes. While the value of the heuristic increases, A* visits fewer nodes but no longer guarantees an optimal path. In application, especially in game development and simulations, it is acceptable to keep algorithm less processor hungry.

Briefly, A* algorithm searches all possible routes from a starting node until it finds the shortest path or cheapest cost to a destination node. Shortest path and cheapest

cost terms refer to a general notion. It could be called any alternative term depending on the problem. For graph search or map problems the cost refer to the term distance. This may decrease the obligation to search all the possible paths in a search space, and improve computation cost. A* evaluates nodes by considering g (n) and h (n). General formulation for A* is

$$f(n) = g(n) + h(n)$$

The aim of this equation is to obtain the lowest f score in a given path-planning problem. n being node number crossed until the final node, f(n) is the total search cost, g(n) is actual lowest cost(shortest distance traveled) of the path from initial start point to the node n, h(n) is the estimated of cost of cheapest(distance) from the node n to a goal node. This part of the equation is also called heuristic function/estimation.

At each node, the lowest f value is chosen to be the next step to expand until the goal node is chosen and reached for expansion. Whenever the heuristic function satisfies certain conditions, A* search is both complete and optimal. Pseudo code for A* algorithm [38] is shown below

```
1 Create open list of nodes, at the beginning open list contains only our starting node
2 Create the closed list of nodes, initially it is empty
3 WHILE (we have not reached our goal) {
4   Consider the best node in the open list (the node with the lowest f value)
5   IF (this node is the goal) {
6     Then we are done
7   }
8   ELSE {
9     Move the current node to the closed list and consider all of its neighbors
10      FOR (each neighbor) {
11        IF (this neighbor is in the closed list and our current g value is lower) {
12          Update the neighbor with the new, lower, g value
13          Change the neighbor's parent to our current node
14        }
15        ELSE IF (this neighbor is in the open list and our current g value is lower) {
16          Update the neighbor with the new, lower, g value
17          Change the neighbor's parent to our current node
18        }
19        ELSE this neighbor is not in either the open or the closed list {
20          Add the neighbor to the open list and set its g value
21        } ENDIF
22      } ENDFOR
23    } ENDIF
24} ENDWHILE
```

2.3 Agent Development Environments

There are a few Java-based agent development environments such as ABLE [27], AgentBuilder [28], Aglets [29], FIPA-OS [30], JADE [31], JATLite [32]. In this thesis, JADE is chosen as the agent development environment due to its Open-Source license and well prepared documentation. Powerful and user friendly plug-in for Eclipse [33]. It is very difficult to debug agents while they are running and not possible with standard IDEs, there is a powerful Eclipse plug-in named EJADE for JADE platforms. The brief descriptions about JADE and other agent development environments are mentioned in the following sections.

2.3.1 Able (agent building and learning environment)

ABLE [27] is a Java library, framework and toolkit for developing intelligent agents with power of reasoning and machine learning. The ABLE framework was developed by IBM Watson Research Center. Able framework provides JavaBean components called AbleBeans to increase reusability, flexible interconnection methods to create working software agents. In addition provides a graphical user interface for interactive development environment.

ABLE has following major components:

- A Java framework for intelligent agents with messaging, event queuing and operation capabilities
- Machine learning agents for forecasting, prediction and classification
- A reasoning component which includes rule engine
- An agent platform that provides agent management and communication across a network

2.3.2 Agent builder

AgentBuilder [28] is an integrated software toolkit for software developers for quickly and easily building intelligent software agents and multi agent based applications. It is developed and supported by Reticular Systems Inc. There are two version of AgentBuilder toolkit as AgentBuilder Lite and AgentBuilder Professional. The Lite version is suitable for building single-agent standalone applications and small agencies and the other one has all features of Lite plus an advanced suite of

tools for debugging, testing and building multi-agent systems. This toolkit uses agent-oriented programming language and provides graphical tools for configuring agents and their behaviors. AgentBuilder aim at developers who have no artificial intelligent background to build agent based intelligent applications. In addition to agent-level development and debugging tools, it provides a set of project management and domain analysis tools.

2.3.3 Aglets

Aglets [29] is a Java mobile agent library and platform which provides development of agent based intelligent applications. Aglets was originally developed at the IBM Tokyo Research Laboratory. Now it is open source and is distributed under IBM Public License.

Aglets is completely made in Java, and ensures high portability of both agents and the agent platform.

Aglets includes:

- Java mobile agent platform
- Tahiti stand-alone server
- Library that allows developer to build mobile agents

2.3.4 Fipa-os

FIPA [34] is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.

FIPA-OS [30] is a FIPA compliant agent development environment which is fully implemented in Java. FIPA-OS supports the majority of the FIPA Experimental specifications and growing under Open Source Community project.

Two alternative distributions are available, Standard FIPA-OS and Micro FIPA-OS. First one contains code developed directly from FIPA-OS codebase without modifications and the second one is an extension to JDK version of FIPA-OS and was developed by the University of Helsinki. Both editions are free and distributed with source codes.

2.3.5 Jade (java agent development framework)

JADE [31] is an open source software framework that is distributed by Telecom Italia. Agent Development framework is fully implemented in Java programming language. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that supports the debugging and deployment phases. Distributed platform architecture is a big plus for sharing process load to different machines without operating system constraint. Agent platforms can be controlled remotely, configurations can be even changed at run-time and it is possible to move or copy one agent from one machine to another. JADE is completely implemented in Java language and the minimal system requirement is the version 1.4 of JAVA (the run time environment or the JDK).

In details, a platform is a layer where multi-agent system launches on JADE. A platform is composed of agent containers which can be distributed on different computers over any network. A container is JAVA environment which provides the JADE run-time engine and services needed by agents to live. It is possible to create more than one container in same platform for different types of process or agents. Many agents can be placed in a single container or distributed to different containers. Agents can communicate over different platforms or different containers easily according to FIPA specifications. There are two options for creating a container. If an IP address is given, the container joins to an existing platform which is hosted on the computer having that given IP address. Otherwise, the container becomes a main container, which is the bootstrap point of a new platform. To create a new agent platform, firstly a main container should be launched and then the other containers should register to main container.

Two agents are automatically started when main container comes alive, which are Agent Management System (AMS) and Directory Facilitator (DF). The AMS provides white-page and life-cycle services, maintaining a directory of agent identifiers (AID). DF is the agent that provides the default yellow page services in the platform [35]. Agents can find the Agent Identifiers of other agents by requesting information from Directory Facilitator. Architecture of container and platforms is depicted in Figure 2.2

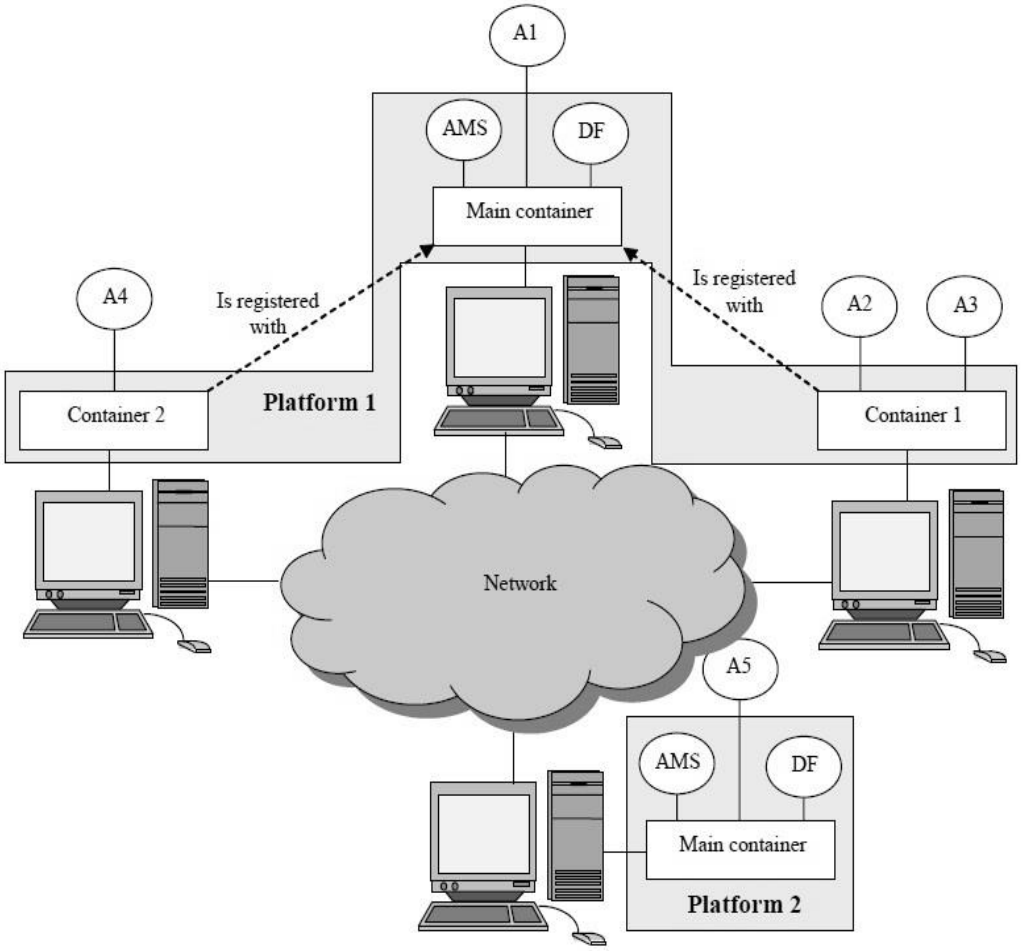

Figure 2.2 : Containers and platforms

Custom or user defined agents can be implemented by extending class "jade.core.Agent". An instance of any class derived from "jade.core.Agent" class is able to join any agent container and live there. An agent can be created while container is starting or after container started. There is a fundamental method called "setup()" in "jade.core.Agent" class, which should be overridden after extending. This method acts like constructor of an agent and initial definitions filled or specified while creating instances of an agent. Every agent has a local name in joined container, and a global name in the main container. Local name is given by the programmers; global name is automatically given by the main container.

An agent is able to execute several behaviours simultaneously. However, it is important to note that the scheduling of behaviours in an agent is cooperative. When a behaviour is scheduled for execution its "action()" method is called and runs until it returns. Therefore, the programmer defines when an agent switches from the

execution of one behaviour to the execution of another. This approach often creates difficulties for inexperienced JADE developers and must always be kept in mind when writing JADE agents. The path of execution of the agent thread is depicted in Figure 2.3.

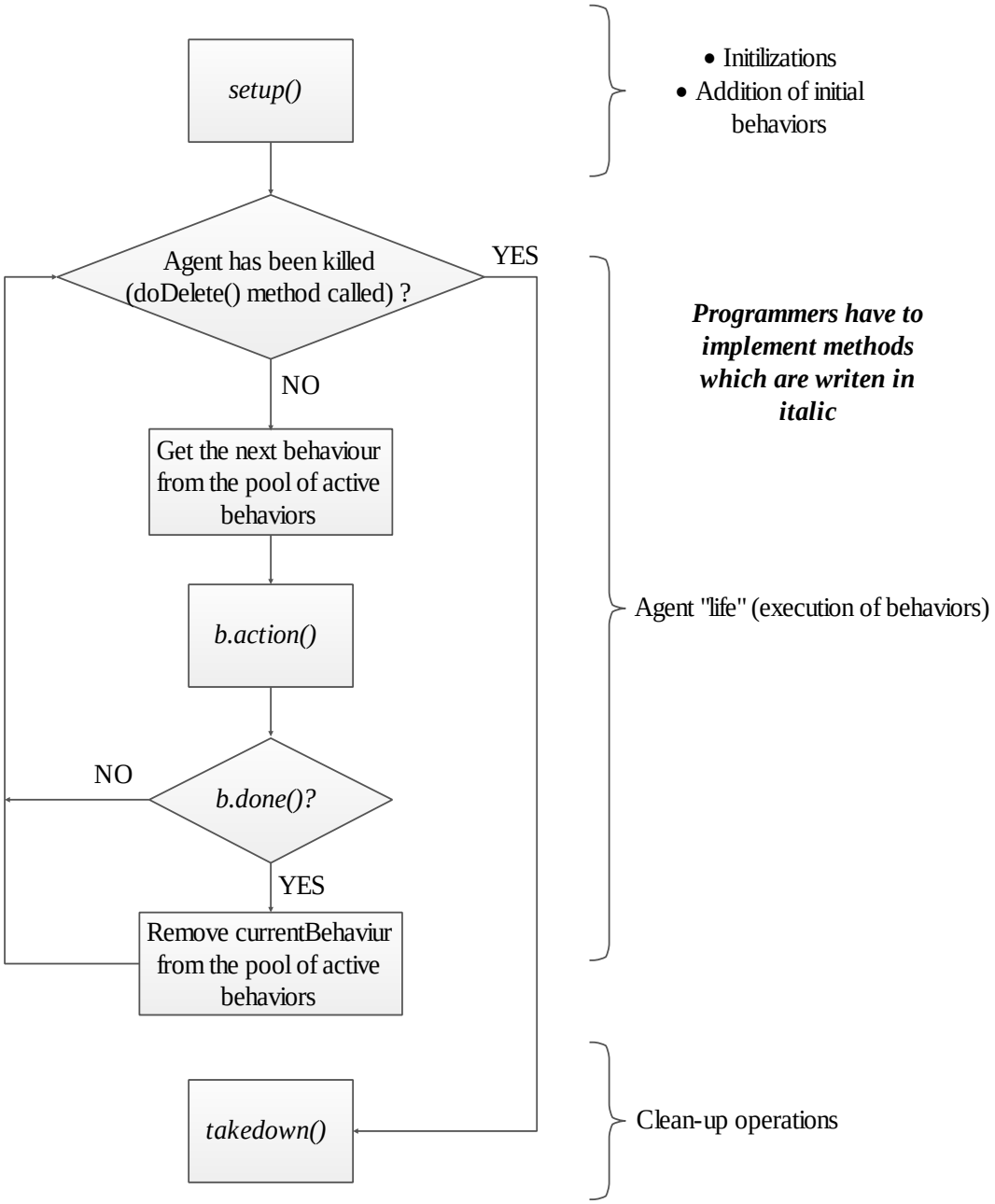

Figure 2.3 : Agent thread path of execution [36]

JADE provides an easy to use messaging mechanism, which enables the programmer to establish communication between the agents over network or internet. Programmers do not have to know about any socket programming to send messages between the agents. Messaging objects are created as an instance of “ACLMessage”

class in JADE that contains two types of content in it. First one is a pure string and second one is a serializable (the conversion of an object to a series of bytes) java object. Every ACLMessage has a performative, which can be manually set by the programmer. INFORM, AGREE, CONFIRM, PROPOSE, REQUEST, REFUSE are basic examples to most common performatives. These performatives have formal semantics defined by the FIPA specification. Briefly, these performatives are used to distinguish the message types, and then messages are processed according to its semantic by the receivers. Once the ACLMessage instance is created, message content and performative is set, Agent Identifiers of the receivers are specified and the message is sent by the methods provided by agent interface.

Every agent has some set of behaviors, which are programmed to perform independent or dependent actions. Generally, these behaviors need to run simultaneously such that, when a behavior is performing a long task, another behavior should perform a different task. Some tasks are performed just for once, some are repeated periodically, and some are repeated continuously. Also, all related tasks should be able to run independently. That's why agents should be multi-threaded and behaviors should be defined as JAVA threads. Agent life cycle states and synchronizations of these threads should be treated carefully.

Standard JADE package comes with a set of abstract behavior classes, which can be used to adapt commonly used task types. Programmer choose an appropriate abstract behavior class and extend it by implementing the abstract methods "action()" and "done()" which are derived from the abstract class. After a behavior class is extended from related abstract class, it is wrapped by a threaded behavior factory and becomes a threaded behavior that is independent than other behaviors. All synchronization, blocking and state transition operations of behavior threads are handled by JADE.

In order to add a behavior to an agent, an instance of a behavior class can be created in agent's constructor method which is called "setup()" and after that this behavior added to this agent's behavior pool. All added behaviors execute immediately after the setup method reaches the end. Some abstract behavior classes which are commonly used in the implementation phase of this thesis are explained below.

OneShotBehaviour is an abstract behavior class that can be extended from “jade.core.behaviours.OneShotBehaviour” class by the programmer to implement operations which will be executed just for once.

CyclicBehaviour is another abstract behavior class which can be extended from “jade.core.behaviours.CyclicBehaviour” and used to implement tasks which will never be completed. This behavior type is useful for reactive tasks such as message listening and replying. Because of it is execution type (lasts forever), this behavior should be handled carefully. Cyclic behaviors may cause other behaviors no to do their jobs when they enter infinite loops.

TickerBehaviour is also an abstract behavior class which is suitable to implement the operations that repetitively waiting a given period after each execution.

3. METHODS AND EXPERIMENTS

In this section, design and architecture of multi-agent based traffic simulation is explained. This simulation has been designed and developed to analyze the effect of manually driven vehicles and semi-autonomous vehicles when autonomous vehicles are introduced in the traffic flow.

3.1 Overview of the Architecture of the Simulation

In this simulation software, JAVA was chosen as the programming language. JAVA SE (JAVA Standard Edition) has many advantages; it is powerful, platform free and stable. There is also a free and ready-to-use agent development framework written in JAVA, named as JADE (Java Agent Development Environment) and JAVA has a flexible MATLAB control library. MATLAB has been used in this thesis for generating complex graph based random map and path planning. Moreover it has an easy-to-use plot function. In short periods, position of the manual, autonomous and semi-autonomous cars are plotted on main map graph which is called from MATLAB. Another reason for using JAVA is that there are some professional and free IDE's (Integrated Development Environment) such as Eclipse and Net Beans. In this thesis Eclipse is used during software development and implementation. Layer platform for combining JADE, MATLAB and MySQL is written in Eclipse. Last but not least, JAVA has powerful collections framework. Hash maps, vectors, array lists are frequently used in agent development and these objects are sent between the agents according to FIPA standards for communication and information sharing. Since these classes in JAVA are serializable, it is possible to send their instances to other agents over network or joined agent platforms.

Eclipse is the most powerful Java integrated development environment (Ide), this environment used for Java code development, controlling other software and software tools. General overview of simulation software and roles of Eclipse, JADE and Matlab are depicted in Figure 3.1.

Simulation starts in eclipse, and continues step by step until it ends. First, MATLAB proxy object is created, and becomes ready to use over Eclipse. City map is generated in MATLAB. To start a simulation over this map, initial parameters must be given. Number of agents in simulation, proportion of agents (autonomous / elderly / cautious / aggressive) and agent specific properties (Start Node, destination, average velocity, speed profile, detection radius etc.) are input parameter for simulation.

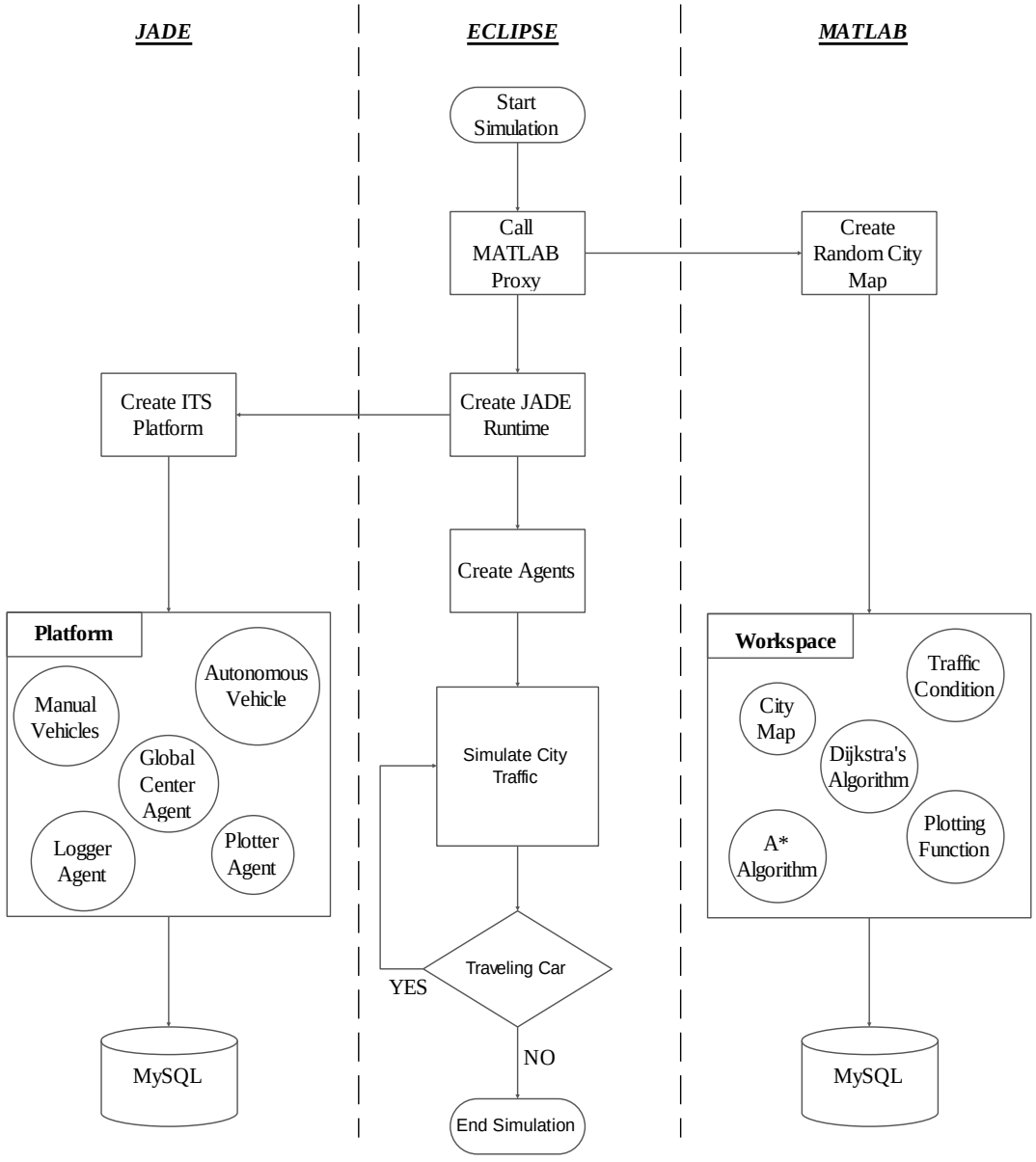

Figure 3.1 : Simulation architecture and platform communications

When agents are created as a software object, path planning algorithm runs in background (this is an agent behavior, agents call algorithm that is developed in MATLAB). In addition, than agent calculates two different paths; one of them is the

shortest path according to Dijkstra's algorithm and the other one is shortest path plus heuristic function based A* algorithm and it uses initial route density information. Some nodes traffic density already known from early simulations. A* calculates most efficient (route that has minimum cost) path for travelling in crowded traffic simulation instead of Dijkstra's shortest path. After route calculation driver knows how to reach destination point and estimated travel time. All these process done when agent objects are created on java runtime engine. Creating an agent object does not mean that agent is alive. When object is created, agent gains identification and driving behavior characteristics. Nevertheless, all agents live in a platform which is held by JADE (Java Agent Development Framework). Multi-Agent systems need communication and Jade supplies a communication protocol that is compatible FIPA (The Foundation for Intelligent physical Agents) specifications.

After Jade agent platform is created all agents born one by one and joins related Jade platform, in this case ITS platform.

Autonomous cars search for a server agent to communicate about route's instant traffic condition; manual cars do not have access to this information since it is assumed that this type of cars is not equipped with ITS devices. If autonomous car cannot find accepted server connection or losses the connection, driver type automatically switches to Semi-autonomous mode, else it is ready for travelling. When route's instant traffic condition differs from expected, route calculation runs again. All route calculation algorithms run on MATLAB.

According to driver type (Autonomous, aggressive, cautious, and delayed) and driver's behavior, agents start travelling. Mentioned behaviors generate different results or travelling profiles because of randomness in simulation. All aggressive or autonomous driven cars do not act like each other. In this thesis, behaviors of these different driver types are analyzed. So every step and every decision of all drivers are logged in to MySQL database. Start point, destination point, instant position, instant velocity etc. are logged to generate simulation output results.

The design has five main components:

- Road Map
- Agent Platform

- Agents
    - Agent Types
        - Manual Vehicle Agent
        - Autonomous Vehicle Agent
        - ITS Center Agents
    - Agent Behaviors
        - Communication Behavior
        - Message Handling Behavior
        - Route Planning Behavior
        - Route Following Behavior
        - Velocity Update Behavior
- Logging
- Visualization

3.2 Road Map

In order to develop the simulation software and test the multi-agent system in action, it is required to generate a road map. For this purpose, a road map storage format is constructed and a sample road map is generated using that format in MATLAB. A complex map was generated to cause some difficulties in simulation runtime, this is a non-constrained map with many connections between nodes.

Two main matrixes are defined, A and xy. A is n*n matrix, which stores adjacencies and xy is nx2 matrix, which stores coordinates of each points. Here, n is number of points in map. Delaunay [37] function in MATLAB is very useful for generating adjacencies; this function creates 2D Delaunay triangulation of the points. “Delaunay” function creates a Delaunay triangulation of a set of points in 2-D or 3-D space. A 2-D Delaunay triangulation ensures that the circumcircle associated with each triangle contains no other point in its interior. This definition extends naturally to higher dimension.

Adjacency matrix A and coordinates of points are listed in Table 3.1 and Table 3.2. Matrix A stores all neighbourhood, and adjacency between two different nodes are represented as 1.

Table 3.1 : Adjacency matrix A

Adjacency	Node 1	Node2	Node 3	Node 4	Node 5
Node 1	0	1	1	0	0
Node 2	0	0	1	1	0
Node 3	1	1	0	1	1
Node 4	0	0	1	0	1
Node 5	1	0	1	0	0

This table shows adjacencies between points, one means they are adjacent, zero means they are not adjacent. According to this table Node 1 and Node 3, Node 1 and Node 5 etc. are adjacent.

Table 3.2 : Coordinates of nodes matrix xy

Coordinates	x	y
Node 1	1,621823082	6,019819414
Node 2	7,942845407	2,629712845
Node 3	3,11215042	6,540790985
Node 4	5,285331355	6,892145031
Node 5	1,656487295	7,481515928

Another useful function for generating maps is gplot function in MATLAB. This functions expect two input parameters. One is adjacency matrix and the other is point coordinates. Generated map using gplot function in MATLAB with A and xy matrix inputs is plotterd in Figure 3.2

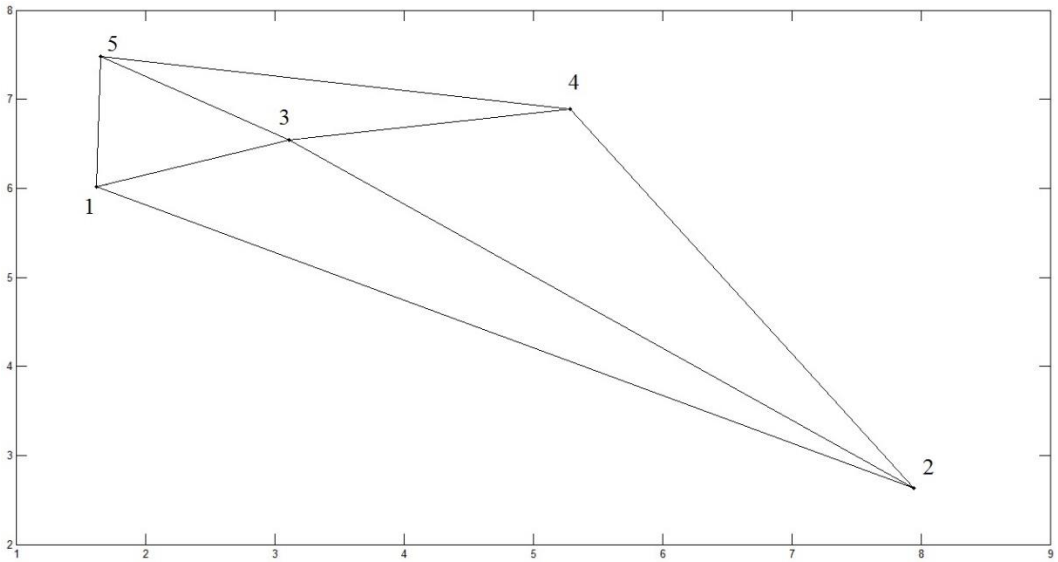

Figure 3.2 : Generated sample map

As it seen from the figure, gplot function generates a simple map. According to adjacency matrix A, Node 1 and Node 3, Node 1 and Node 5 are a few samples of adjacent nodes.

In the scope of this study, different road maps are developed for different purposes. Smaller maps are developed for testing performance and debugging agents which behaves like real drivers. However this generated maps are too small to simulate more than hundred cars simultaneously, thus a bigger map is generated which has more than hundred nodes. Briefly, a little complex city or village is generated randomly and Figure 3.3 shows plotted virtual city. In future works, this map can be generated from Google Maps or any other map providers automatically and easily.

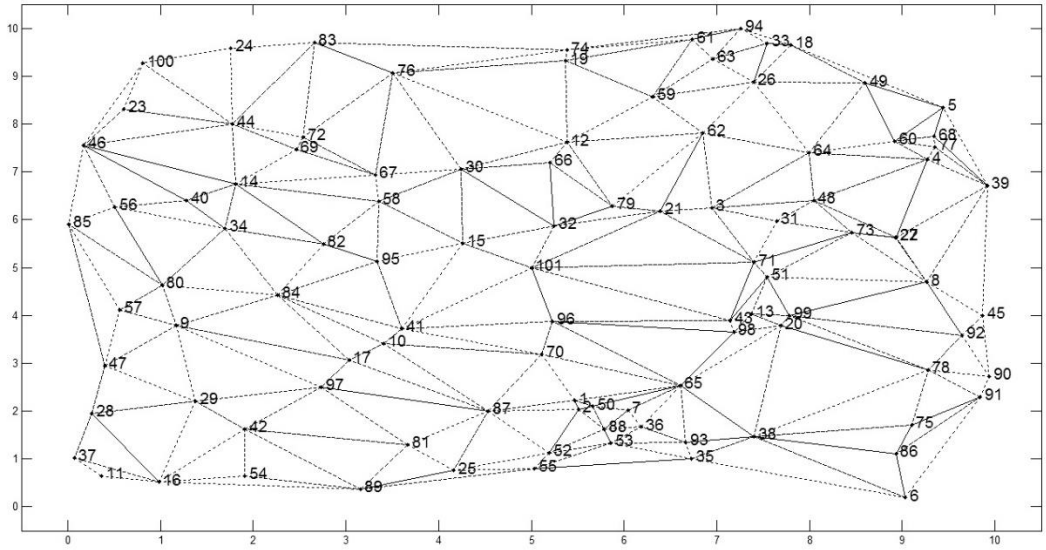

Figure 3.3 : Base map of ITS simulation road map

In simulation, all agents start from a start node and travels to destination node. All of these processes are random; an agent travels from new start node to new destination node in each simulation. This is designed for randomization in simulation. After start node and destination node are defined, path planning calculates the shortest path and returns detailed path.

3.3 Agents

In artificial intelligence, an intelligent agent (IA) is an autonomous entity, which observes through sensors, acts upon an environment using actuators, and directs its activity towards achieving goals. Intelligent agents may also learn or use knowledge

in order to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is an intelligent agent, as is a human being, as is a community of human beings working together towards a goal. Simple reflex agent [39] model is depicted in Figure 3.4

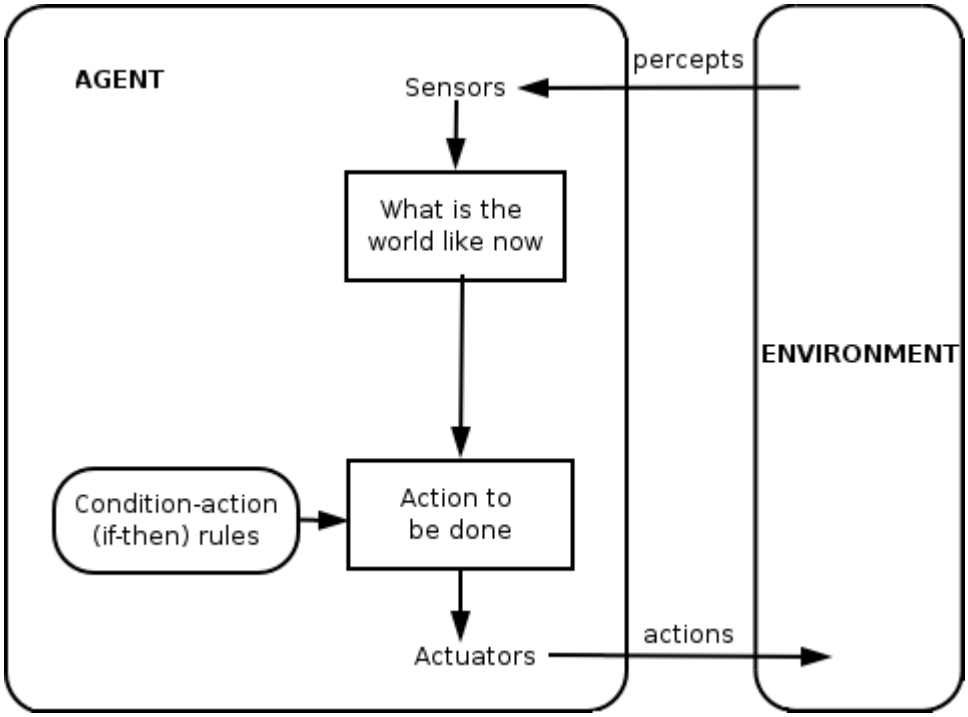

Figure 3.4 : Simple reflex agent

In this study, all vehicles are represented as agents, some are autonomous, some semi-autonomous and some are manually driven. Autonomous cars are represented by one type, and manual cars are represented by three main types; aggressive, cautious and elderly driver.

Two different object types were inherited from Jade agent class and have riched with other parameters that they generate a characteristic for a vehicle. These parameters are inputs for some equations like route following, choice of speed or vehicle following distance.

All agents were programmed with simple reflex agent pattern to mimic human behaviors and to simplfy driver skills. More complicated design pattern are available in literature, but they are difficult to implement in traffic simulation and some of them are far away from mimicking human driver's behaviors. In this case, simple reflex design pattern is enough for our simulation platform.

Simply, all of them are represented by agent class, programmed with simple reflex design and they have specific behaviors. Internal structure of vehicle agent is shown in Figure 3.5

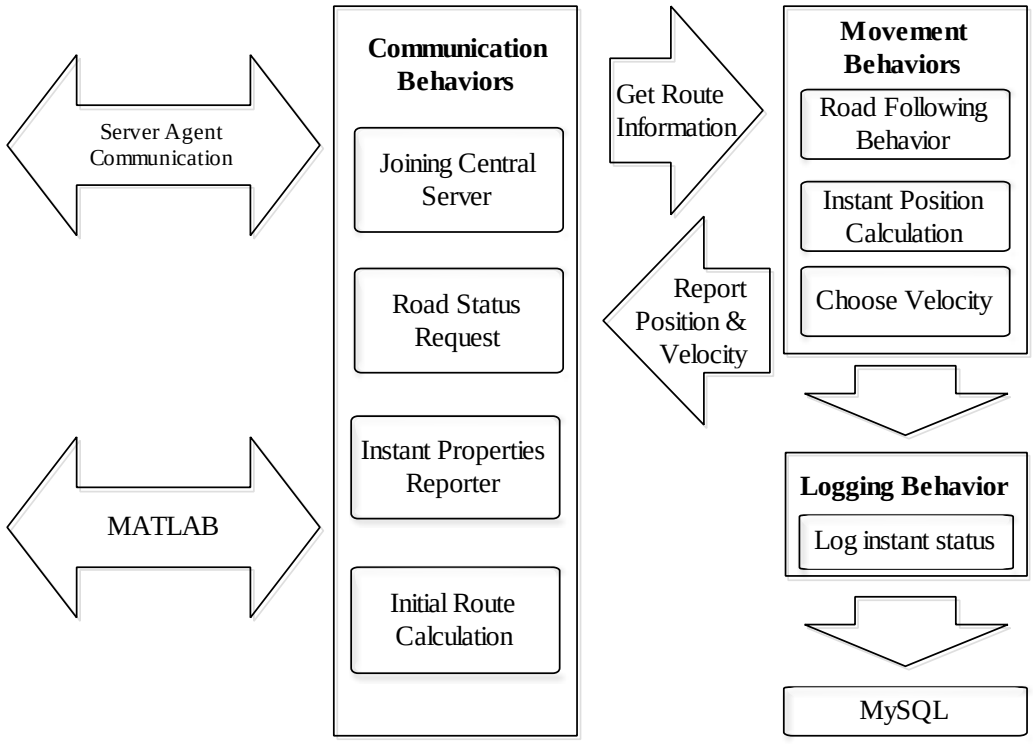

Figure 3.5 : Internal structure of vehicle agent

All vehicle agents and server agent are using the same abstract road map, actually vehicle agents are only interested in their routes and other vehicles agents on their routes. Road map is at the bottom level of vehicle agent architecture, when agent's "setup" method is called, random start node and random end node are chosen. After start point and destination point are defined, detailed route points are calculated by path calculator engine in MATLAB. This function uses Dijkstra or A* path finding algorithm to calculate node list to visit while travelling from start point to destination point. This path contains only node list to visit, after node list generated, this node list must be detailed into tiny pieces. User defined MATLAB path detailer engine serves for this purpose.

Every vehicle agent report their instant positions and velocity to central server and receive instant route information. Every step, vehicle agent calculates its next movement and stores instant position in a hashtable. This process never stop until vehicle agent reaches its destination node. Hashtable object which includes vehicle's instant properties like position, velocity, direction and path is sent periodically to

joined central servers. At the same time, vehicle agent waits for replying message which includes instant traffic information especially nearest vehicle agent's positions and velocities.

- **MatlabProxy** is a proxy object driven from MatlabControl library that is used for controlling Matlab run-time engine and also creating Matlab statements.
- **AgentName** is an argument which determines agent local name. Agents that share same container, reach other agent by their local names. This argument generated programmatically with a simple notation like "CarType_(generated Order)" for instance "AutonomousCar_(23)" is given for autonomous car agent which is generated as 24th agent in container (remember index in Java starts with 0).
- **AgentType** is an argument that determines the type of the car agent, this argument can take below values:
    - Autonomous Driver

      This driver type represents autonomous cars in traffic.
    - Semi-Autonomous Driver

      This driver type represents autonomous cars in traffic.
    - Aggressive Driver

      This driver type represents aggressive drivers in traffic, they have good driving skills but unpredictable behaviors.
    - Cautious Driver

      This driver type represents cautious drivers in traffic, they have standard driving skills and they are closest to autonomous drivers.
    - Drivers with weak reflex

      This driver type represents who do not have good reflex for driving skills so this cause unexpected actions in traffic flow. They have low speed profile.

  Autonomous drivers may switch to semi-autonomous mode when some emergency situations occurs. But when creation of the agent, it always takes "Autonomous Driver" as driver type parameter. While simulation starts, every autonomous vehicles are in auto-pilot mode, and they switch to semi-autonomous if necessary.

- **StartNodeId** is an argument to place the agent at a specific node when simulation starts. The value must be a valid node id in the map. This node is randomly chosen while agent starts as destination node. For manual agents, only start node id is given as parameter.
- **DestinationNodeId** argument is used to set a destination for the car agent. The value must be a valid node id in the map. Car agent travels from start node to destination node. For manually driven agents, only start node id is given as parameter, and then a neighbor node is chosen randomly as destination node, when manually driven agent reaches destination point, destination point becomes new start point and manually driven agent picks a new neighbor node as destination point. This cycle continues until all autonomous driven agents terminate and simulation ends.
- **MaxVelocity** argument is used to set a maximum speed limit for the car agent. This is also generated randomly around speed limit for urban city. For manual car agents there is floating in speed profile while autonomous cars have stable speed profile.
- **SimulationName** argument is necessary for logging mechanism, intelligent transportation center creates a logging record in MySQL database and after this step all logging records are inserted with that simulation name.
- **Connection** argument is necessary for logging mechanism, like MatlabProxy object MySQL connection object is initialized when simulation starts and this connection object is given as input parameter for all car agents.

When a vehicle agent is created, setup method firstly parses input arguments explained above, some of them internally declared as constructor. These paramaters are inputs for some calculations and some communication behaviors. For example MaxVelocity parameter is input for an equation that calculates speed for next step using traffic conditions.

Then it searches for the server agents by using the Directory Facilitator provided by the main container. When the agent identifications of the server agents are found, they are stored for later communication. Finally, the behaviors are defined, created and agent becomes ready to run.

3.3.1 Agent types

There are two types of vehicle agents available in this scenario such as autonomous vehicle agent and manual vehicle agent. These agents are classified according to driver types. One of them is autonomous agent that is equipped with all latest technology in ITS field and automotive industry and the other one is manual car, which is driven by only human skills and it is assumed to have no technology like server communication, route planning etc.

Classifying vehicles in two groups is not enough to analyze traffic flow behavior when mixed traffic because driving characteristics differ from driver to driver. Autonomous cars may be assumed as identical and in this thesis, it is assumed that all autonomous vehicles meet the minimum requirement of autonomous driving.

Defining manual vehicles in simulation is harder than defining autonomous cars. All drivers in city traffic has unique characteristic and behaviors may vary according to traffic situation. Generally, three type of manual vehicle defined; having aggressive driver, cautious driver and elderly driver. Moreover, driver type's proportion may vary from city to village or from city to another city. To simulate worst scenario, crowded city traffic profile is chosen as base city traffic simulation.

3.3.1.1 Autonomous vehicle agent

Autonomous vehicles are equipped with high technology such as driving control systems, communication systems, and warning systems. This type of vehicle (agent) has a long range communication, early warning systems for accident prevention, conjugate communication with local center servers and large detection radius while car following mode.

The autonomous car agent starts from a node and moves to a selected destination node. This agent firstly calculates the route between these two nodes and follows that route until it reaches to destination node. In this study, these types of agents are especially tracked, their behaviors, delays, actions in simulation environment are key points for answering our research quest. Using these logs, a report is generated at the end of the simulation.

As a software object, vehicle first takes its parameters and load them as its characteristics, calculates path to travel from start node to end node according to

suitable path planning algorithm A* or Dijkstra and gets ready to travel along calculated path. In constructor part of object, all heavy load processes are done and all necessary communications are established to increase simulation performance. Also joining agent platform, which is served by JADE, is done in constructor part. All vehicle (agent) objects get ready and wait for travelling.

After all construction processes ended and join process succeeded, agents become alive. Nevertheless, agent (vehicle) does not starts travelling; agents becoming alive method that is named "setup" must be called. After all agents are ready, eclipse environment calls all agents "setup" method.

Agents start travelling, behaviors that are necessary for route following run in cyclic manner and every next movement of autonomous car is calculated in constraint to route and traffic condition. In addition, car types in detection radius are important for making decisions especially for selecting the proper velocity profile.

If any dangerous situation occurs near autonomous car, it warns passenger and waits for an approval to switch to semi-autonomous driving mode. In this mode, car assists to driver but does not control cars movement behaviors as primary controller and acts more like a manual vehicle.

Every cycle, the agent (vehicle) controls its position, if it has reached its destination point, the agent reports this to the center server agent and terminates itself. The simulation does not stop until there is no remaining agent stays active/alive in platform.

3.3.1.2 Manual vehicle agent

In this thesis, the aim was analyzing that how proportion of manual cars affect the autonomous car's behaviors, the general traffic flow and travel profiles. In simulation environment, autonomous cars perceive manual cars as disturbance. Manually driven car's behaviors are difficult to predict and a human driver has not a stable speed profile or route following behavior. On the other hand, a human driver does not trust to an autonomous vehicle when following it although the leader is a high tech car. Therefore, there must be a huge gap between autonomous cars and manual cars while they are following each other.

Manual vehicle agents behave very differently according to their driver types and in differently than other vehicle agents in same group. For example, aggressive driver group behaves more independent to traffic than cautious driver group and an aggressive driver behaves different than another aggressive behavior because there is some randomly generated parameters in driver behaviors. One aggressive car may behave like cautious driver in some simulation steps as well, but fundamentally it is an aggressive driver and most of its behaviors are aggressive. This gives unpredictable movements to aggressive type vehicles.

Manual vehicle agents are not equipped with technological driver assistant systems and all route following behavior depends on driver characteristics. They may have weak reflexes, unexpected movements and low driving skills. They generally have limited detection capacities and do not have conjugate communication with local or global center agent. For simulation necessities, they report instant positions and velocity information to its center agent (server) and does not receive any route information about server. To calculate next movement in traffic, its server sends only ahead vehicle's last known position and velocity.

The agent starts to travel from selected start node and picks one of its neighbors randomly as a destination node. After reaching to the destination node, another random neighbor is selected as new destination node and instant node becomes a start node. Agent repeats this behavior as long as it is alive in the active platform. Agent platform will be deactivated when all autonomous agents reached their destination nodes.

As a software object, manual vehicle agent first takes its parameters and load them as its characteristics, travel from start node to end node according to random strolling behavior. Manual vehicle agents do not have a path planning algorithm in this case, route of a manually driven vehicle is not known even in a simulation. That is why they do not need a complex path planning algorithm, manual driver makes its own route planning and follows it. They may be stuck in a traffic jam, may choose the longest route or pass the same node more than once. As autonomous vehicle agents, all this processes run in constructor part too and must join a living platform which is served by JADE to participate simulation. After all of this processes ends, manual vehicle object (agent) become ready and wait for travelling.

After all of this processes ends, manual vehicle object (agent) becomes alive, waits for travelling. But manual vehicle agent does not start travelling, agents becoming alive method that is named “setup” must be called. After all agents are ready, eclipse environment calls all agents “setup” method as mentioned before.

Manual vehicle agents start travelling, behaviors that are necessary for route following run in cyclic manner and every next movement of manually driven car is calculated in constraint to route and traffic condition. Also the type of the car in detection radius is important for making decisions especially for velocity profile. There are some differences like cruise control, detection area and decision making capabilities between manually driven and autonomous vehicles.

It is assumed that manual vehicles are not equipped with any ITS technology system for worst case simulation scenario. If any dangerous situation, driver is on his own and has to sense the danger and deal with the problem. Autonomous cars can sense this type of dangerous situations before manual cars and take precaution like putting a larger gap between ahead vehicles and slowing.

Every cycle, agent (vehicle) controls its position, if it has reached its destination point, agent reports this to the center server agent and picks a new destination node to travel. This cycle runs until simulation ends. Simulation needs manual vehicles in city traffic to generate complexity in traffic conditions.

Aggressive driver

This driver type has high speed profile, wide range detection circle and usually unpredictable dangerous actions in urban traffic. Most drivers has this profile in crowded city traffic, even the cautious drivers. Drivers who have sportive driver skills also have similar capabilities. For example taxi drivers are in this group in Turkey, they spent a lot of time in traffic, they have enough experience but behave unexpected in city traffic. Drivers must be careful while a taxi is around.

Actually they have better driving skills than others but they are impatient while driving and want to go their destination as soon as possible. Therefore, this type of driver is dangerous every time, and the other vehicles must be cautious when they are in the vicinity of them even the other vehicles have an auto-pilot mode.

Like the other manual agent types, this type of manual vehicle agent takes starting arguments restricted to aggressive driver profile. This type of manually driven

vehicles is usually speeding, has rarely unstable speed profile, brakes suddenly and has wide range detection capabilities.

These group is dangerous for other drivers in city traffic, because of their impatient, they cause some difficulties in traffic like line changing unexpectedly.

Cautious driver

This driver type has average speed profile, middle range detection circle and usually predictable actions in urban traffic. Cautious drivers are the closest to autonomous car agents. Cautious driver has good driving skills too, different from aggressive or sportive drivers, they have middle range detection area and low speed profile. This type of drivers wants to reach their destination point safely, they generally travel with at least one family member or a friend. This type of vehicles acts like autonomous car friendly in urban traffic.

Like the other manual agent types, this type of manual vehicle agent takes starting arguments restricted to cautious driver profile. This type of manually driven vehicles do not exceed urban city speed limit, have stable speed profile, brake gradually and have middle range detection capabilities.

Family guy is a good example for this group, imagine a father traveling with his family in city traffic and avoid dangerous actions while driving. He will have stable speed profile and will obey all of traffic rules not to endanger his family.

Drivers with weak reflex

This driver type has floating speed profile, short range detection circle and usually unpredictable actions in urban traffic, however they are not as dangerous as aggressive drivers. Long for short elderly driver used instead of driver with weak reflex.

Inexperienced, elderly and intoxicated drivers are in this group and they do not have good driving skills. In dangerous or unexpected traffic situations, they respond early or lately. Autonomous cars must be cautious when any elderly or delayed driver around and must follow them with a big gap.

This type of manual vehicle agent takes starting arguments restricted to delayed driver profile. This type of manually driven vehicles do not exceed urban city speed limit, do not have stable speed profile, brake suddenly, unexpectedly and have short range detection capabilities.

This group is also dangerous for other vehicles but not as much as aggressive drivers. They unintentionally cause some unexpected behaviors in urban traffic and may leave other drivers in the lurch.

Summary of all vehicle types and their behavior comparison is shown in Table 3.3.

Table 3.3 : Agent types and properties

Agent Type	Autonomous Agent	Manual Aggressive	Manual Cautious	Manual Delayed
Average Speed Profile	medium	high	medium	Low
Cruise Control	high	low	medium	Low
Car Following Behavior	high	medium	medium	Low
Communication Capabilities	high	no comm.	no comm.	no comm.
Driving Skills	high	high	medium	Low
Coverage Area	high	medium	low	Low

3.3.1.3 ITS center agents (global center agent)

Another agent type is central ITS server, which collects traffic data and processes them. Every autonomous vehicle has to connect the center agent for route and positions information before they start journey. Otherwise, autonomous vehicle will not have enough knowledge about real city traffic data, it will just sense what is in detection area. All agent's data is processed and analyzed by center, obtained results are sent to each autonomous vehicle agents.

For this simulation environment its center agent has the key importance. All localization process is dependent on its center agent, this center knows every agents position and instant properties, and also autonomous agent's travelling route and broadcast instant route data. To improve accuracy, its center agent sends all autonomous and known manual vehicle's positions to autonomous vehicles.

This agent is necessary for simulation to localize all vehicles, based on this localization agent, all the vehicle fleet senses and knows the environment. This is

like obtaining all other agents exact position on map in defined detection radius. It was mentioned before, detection radius is a parameter for a driver and it changes from driver to driver according to their driving skills, ages and experiences. All agents connect this center to report their positions, otherwise it is not possible to define a vehicle in simulation environment. Also agent cannot travel in this map unless it connect to its center agent.

Its center collects all vehicles instant properties objects and stores them in a suitable object. When every message handling behavior runs, this object is created again to hold updated properties. Message handling behaviour has the same flow mechanism like other agents. It checks inbox for a message and if there is any, it processes receiver car (agent) identity and stores instant properties to share with related other agents.

Three main behavior of this agent are Behavior_JoinRequestHandler that handles the connection requests received from vehicles to participate in simulation, Behavior_RouteRequestHandler that handles the route request received from autonomous cars in autonomous driving mode and Behavior_PositionVelocityHandler that gathers all instant properties of all vehicles (agents) in simulation and sends back related vehicle's positions.

All localization and communication process is performed on its center agent. For simulation frameworks this is necessary, especially if of the cost of the localization algorithm is needed to be reduced.

Its center agent is also a local center agent and capable of doing all the job of local center agent as well.

3.3.1.4 Local centre agent

Local center agents are located on bottle neck points of the road infrastructure. These points are very important, imagine a corner in city, more than three roads converges on a route. They affect traffic flow in that area in a negative way. Local center agents know traffic density inside their coverage area and broadcast this information to all connected vehicles inside communication range.

They gather instant positions of vehicles that are inside detection range by querying MySQL. As mentioned before, all agents report their positions and record them into

the MySQL database. Instant positions of all agents recorded with simulation identity, vehicle identity, simulation step, exact position and also driver type. In this thesis, only number of vehicles in range is used for density calculations. For future work, specific driver types may be used for density and according to situation, autonomous vehicles can be guided to more appropriate routes. This can be considered some kind of of intelligent routing, however not developed to its full potential yet.

In this simulation framework, local densities are used for route calculation and information gathering. Autonomous car may change its route when any overload in local center points is sensed /estimated.

Message handling behavior flow for local centre agent is depicted in Figure 3.6.

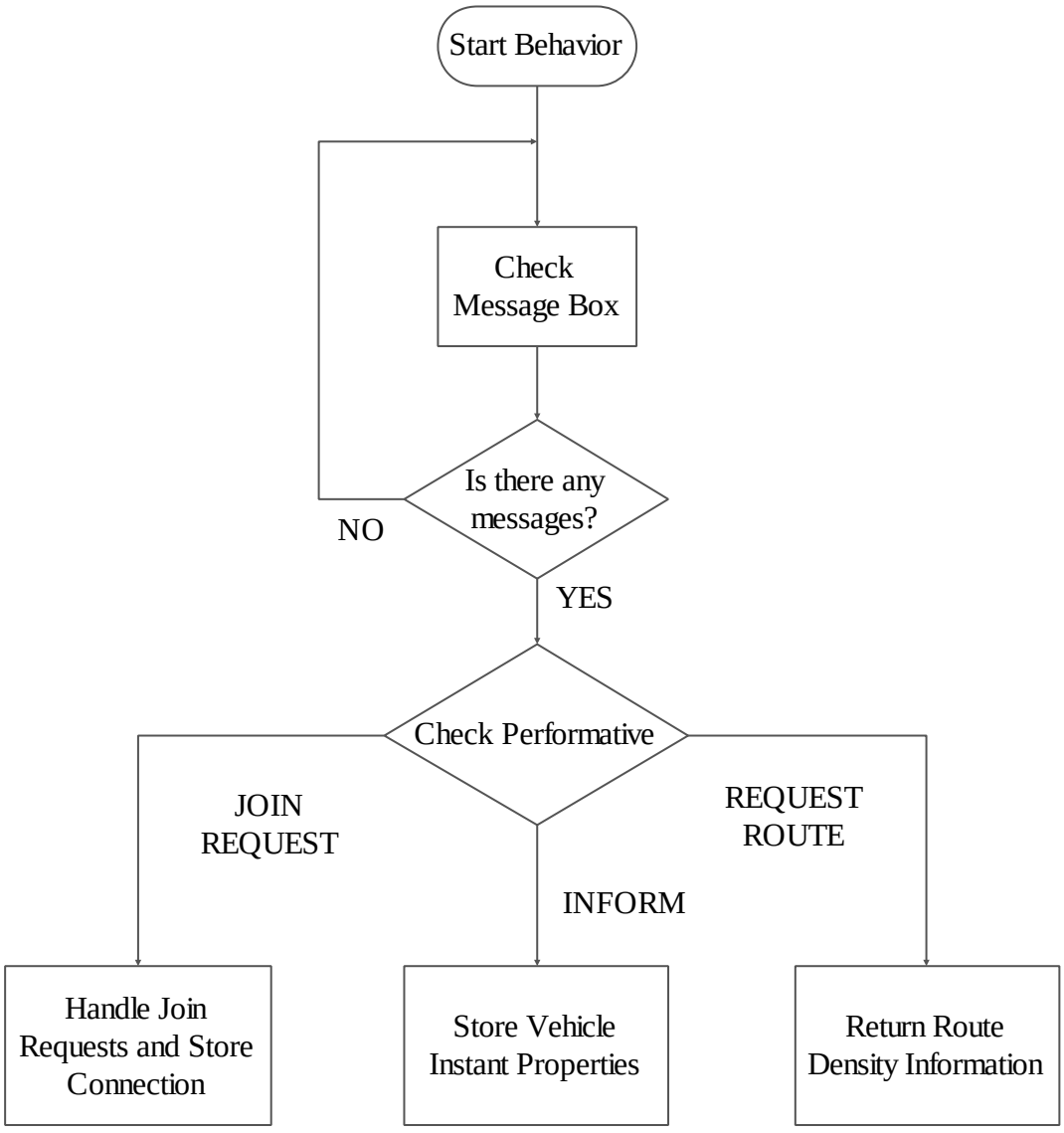

Figure 3.6 : Message handling behavior flow chart

When simulation starts, local center agents are created, they settle on desired locations and become ready for connection. Local center agents have two behaviors, one behavior is for communication establishment which is named as Behaviour_JoinRequestsHandler and the other one is for instant density calculation that is named as Behaviour_UpdateLocalDensity.

3.3.1.5 Plotter agent

This agent helps visualization of instant positions for all moving agents. It calls a MATLAB script which takes all instant positions from MySQL database and plot them on a MATLAB-based figure. This agent is a helper tool for simulation, it only plots the instant agent positions. Plotting feature developed with multi agent architecture and written according to Singleton software pattern [41] to improve performance. The singleton pattern involves only one class which is responsible to instantiate itself, to make sure it creates not more than one instance. In this case the same instance can be used from everywhere, being impossible to invoke directly the constructor each time.

For early simulations, every agent was creating its own logger and it takes more than five minutes to draw all agent's positions on map. After Singleton pattern applied for plotter agent, it took only seconds.

Agent plots autonomous vehicle positions as diamond shape in MATLAB and manually driven vehicles as circle shape. To improve visualization while simulation running, plotter agent colorize manual vehicle with red to take attention and colorize autonomous vehicles with blue.

Plotting cycle is changeable, if there is any concern about performance issues, one should use ten seconds as maximal interval. Vehicle agents all instant step by step positions are recorded in MySQL database, therefore, more detailed report may be prepared from querying related database. Plotting agent works only for graphical visualization.

3.3.2 Agent behaviors

Behavior types existing in JADE are explained previously, mentioned behavior types are valid only for software inheritance. New vehicle behaviors developed over this base behavior models and new features are added in this study. Behavior types are

only a template for coding and they all have their calling methods. When an agent is born, “setup” method is triggered from eclipse environment and necessary behaviors are added one by one according to precondition order. For example, route following behavior runs after path planning behavior finishes its work.

In a vehicle agent, there are four different behaviors implemented which are explained in the following chapters.

3.3.2.1 Communication behavior

The most useful feature of JADE platform is communication. For this study, every agent does not communicate with each other, but all agents inform server agents. Communication architecture is based on standards of FIPA [34] agent communication language specifications. When an agent is created, behaviors start simultaneously in order of declaration. Firstly, vehicle agent tries to find server agents. After server agents found in Directory Facilitator, vehicle agent requests to join these servers and waits for a reply.

Agents are also have a communication with MATLAB for generating route in city map. This part of the process is handling in generated MATLAB route finding engine. But this communiation is not developed as behavior, an agent is programmed for this purpose using singleton software pattern. With help of singleton, platform gain a huge performance improvement.

Same architecture was applied for logging mechanism too while writing logs in MySQL database.

3.3.2.2 Message handling behavior

This behavior may be the most important behavior, and also reflects the power of JADE environment or multi agent programming. Even simple reflex agent has to know about outside environment to make decisions. Messaging between agents provides knowledge sharing in same agent platform or other known agent platforms. All this communication occurs according to FIPA standards. Basic flow diagram of message handling behavior shown in Fig 3.7

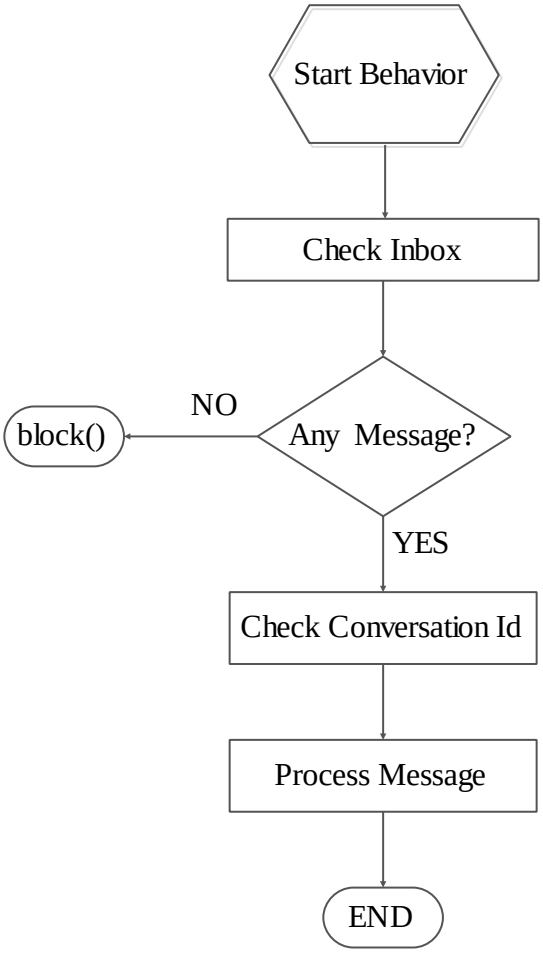

Figure 3.7 : Message handling behavior

All messages have a format in communication, also FIPA offers a language called ACL (Agent Communication Language). This a communication protocol, so it gives programming freedom in different languages. It is possible to develop agents in different programming languages like Java, C#, C++ and make these platforms communicate with each other.

Every message has a sender, receiver and content. Same template and different performatives like REQUEST, INFORM, CONFIRM are generally used. And java objects are sent as content. Every receiver checks it's performative and processes the message if it is necessary.Vehicle's positions are sent to global center agent that gathers all traffic information and all vehicles instant positions and velocities. Every autonomous vehicle has to send connection request to global center agent when they are born. Server can accept or reject communication request on the authority of server load or distance for healthy communication. If autonomous agent cannot make connection with server agents, it switches to semi-autonomous mode. Therefore,

communication is the key feature for autonomous cars in simulation environment Flow diagram of communication is shown in Figure 3.8.

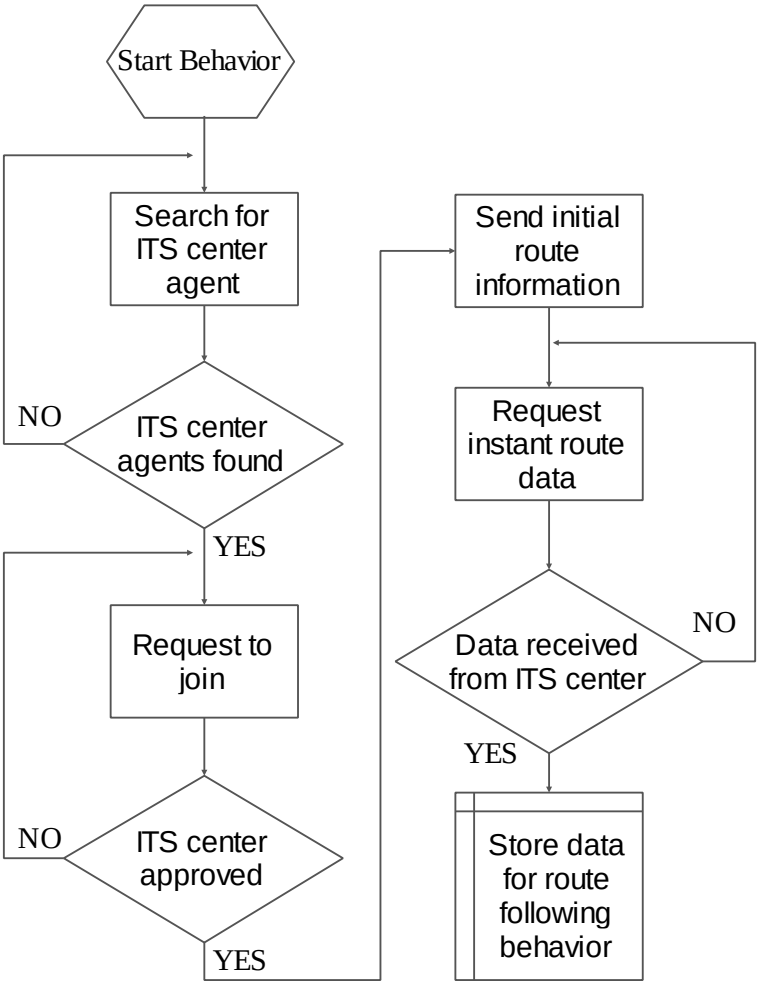

Figure 3.8 : Flow diagram of communication behavior for vehicle agents

3.3.2.3 Route planning behaviors

This behaviour calculates an initial route using two input parameters, one of them is start node ID and the other one is destination node ID. There is at least one route between two points in a map, results may vary according to the path planning algorithm.

When agents born, they take start node and destination node parameters. Using these points, agent calculates its first route according to known node traffic densities. Route Planning behaviour calls path planning engine in MATLAB and waits for calculated route. After node list is calculated, behaviour calls MATLAB path planning engine and have it sliced into tiny pieces for simulation steps. While using this route planning engine, MATLAB is also communicates with MySQL database to

get initial node densities. This initial route densities are calculated from last simulations and gives an input for Dijkstra's algorithm.

In MATLAB, user defined path planner engine uses two different algorithms for calculation. A* and Dijkstra path planning algorithms are very similar, actually Dijkstra algorithm is the special case of A* algorithm. While Dijkstra algorithm always calculates the shortest path, A* algorithm calculates the most efficient path between given two points. A* uses heuristics, in Dijkstra algorithm heuristic is always zero. A* uses Euclidean distance plus heuristic value.

Inner structure of vehicle agent, path, cost and detailed path are stored in different objects, especially in hashtables. Detailed path is a kind of planned route, if everything goes as expected, vehicle agents will follow the detailed steps on abstract map.

After initial route is calculated, agent car sends this information to the ITS center, this information is very valuable input for intelligent routing applications and it is useful statistical data.For instance, calculated path for a travelling vehicle agent from node 37 to destination node 49 is shown below. This path was generated by Dijkstra algorithm. Dijkstra's solution from node 37 to node 49 is plotted in Figure 3.9.

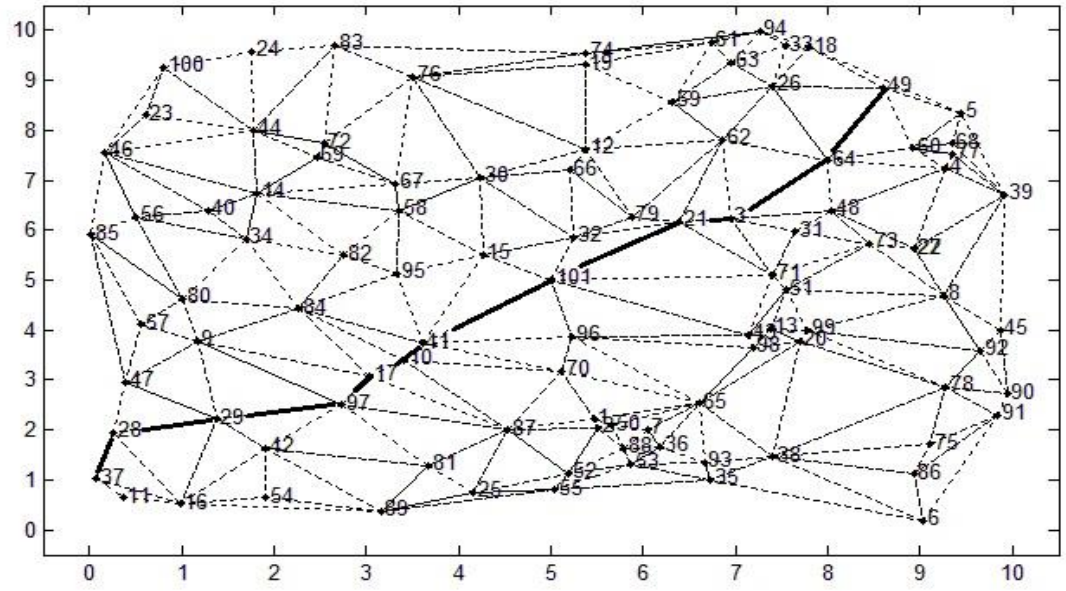

Figure 3.9 : Example path planning on ITS road map

Algorithm takes Node 37 as start node and searches shortest path to destination node in this case Node 49. All nodes which are related with this sample path finding process and distances between nodes are shown in Table 3.4.

Table 3.4 : Dijkstra's node list and calculations

Visit Order	Node Id	x	y	Distance (Unit)	Node Density (factor)	Heuristic Distance (Unit)
START	37	0,068461	1,020938			
1	28	0,258533	1,943551	0,941989	0,727665	1,627442
2	29	1,37563	2,214528	1,149493	0,941416	2,231644
3	97	2,733544	2,498528	1,387294	0,897555	2,632466
4	17	3,032216	3,074959	0,649214	0,832001	1,189360
5	10	3,408015	3,403937	0,499451	0,951344	0,974600
6	41	3,605828	3,724355	0,376559	0,968305	0,741183
7	101	5,000000	5,000000	1,889706	0,938958	3,664060
8	21	6,389436	6,176702	1,820758	0,713361	3,119615
9	3	6,945368	6,240993	0,559636	0,927324	1,078600
10	64	7,998314	7,391522	1,559619	0,701701	2,654005
END	49	8,597923	8,851172	1,578008	0,904163	3,004785
			Cost:	**12,41173**		**22,917761**

As mentioned before, Dijkstra's path finding algorithm finds shortest path, and if you feed this algorithm with heuristic feedback, it becomes A* path finding algorithm. Actually Dijkstra's path finding algorithm is the special case of A* algorithm with heuristic function is always equal to zero. To compare Dijkstra and A* algorithm, same two points are selected as start and destination node. Dijkstra finds the shortest path between two points and A* calculates another path because of known or if necessary, instant traffic density.

As it is shown in graph, path calculated by A* is longer than Dijkstra's solution. Total distance for Dijkstra's path is about 12.4 units where A* path is about 13.2 units. Dijkstra never calculates heuristic costs, to compare Dijkstra versus A* algorithm, instant node densities that are Dijkstra's solution are used to calculate heuristics manually. Dijkstra's calculated cost with heuristics is about 23 unit length and the path that A* offers is 19 unit length, detailed calculations are shown in Table 3.5 and Table 3.6. To compare two different paths, Figure 3.10 shows us both paths.

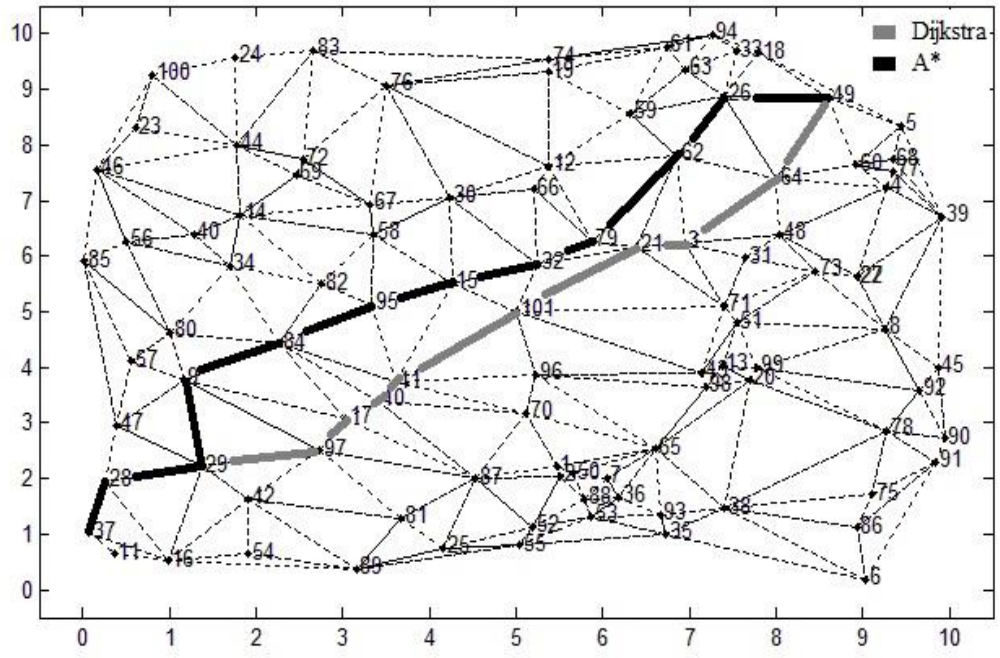

Figure 3.10 : Dijkstra vs A* algorithm for same destination

Two calculated paths are shown above. Node 101 is the center of the city as it seen, seven roads cross at the same point. This is where generally bottle neck occurs. City map below is simulated more than fifty times and initial densities updated after every simulation ended.

These shows that, city center (node number 101) has a higher initial node density because of earlier simulations. May be a bottleneck was generated in some of the simulatons at this node. Because that node has seven connections with other nodes. So traffic flow on this point may be different than others if all neighbours have traffic jam in their area.

Dijkstra's algorithm cannot sense this traffic density, because it searches shortest path without considering their traffic conditions. But A* algorithm uses node densities as input to calculate real shortest path in urban traffic.

Detailed calculation for A* route finding algorith is shown in Table 3.5. Same destination with Dijkstra's algorithm is selected to compare two algorithm's performance. They have same number of nodes to visit, but generates different routes.

Table 3.5 : A* node list and calculations

Visit Order	Node Id	x	y	Distance (Unit)	Node Density (factor)	Heuristic Distance (Unit)
START	37	0,0685	1,0209			
1	28	0,2585	1,9436	0,942059	0,511184	1,423624
2	29	1,3756	2,2145	1,149478	0,331676	1,530733
3	97	1,1695	3,784	1,582974	0,526868	2,416993
4	17	2,2654	4,4214	1,267784	0,444598	1,831438
5	10	3,3404	5,1352	1,290401	0,400369	1,807037
6	41	4,2625	5,5108	0,995662	0,421705	1,415537
7	101	5,2444	5,863	1,043155	0,452397	1,515075
8	21	5,877	6,2741	0,754444	0,545660	1,166114
9	3	6,8527	7,8154	1,82417	0,558704	2,843341
10	64	7,3979	8,8802	1,196262	0,463818	1,751110
END	49	8,5979	8,8512	1,20035	0,430764	1,717418
			Cost:	**13,24674**		**19,418420**

In this simulation environment, all chosen paths and instant traffic densities are logged. So every node in a city map has an initial traffic density which was obtained from early simulations. A vehicle may use this data for path planning when starting journey. This is a general human behavior while we haven't started journey yet and we are at the planning stage. If a driver wants to go from point A to point B, first him or her draws the path in his/her mind according to distance or traffic density and in this simulation environment this real driver behaviors are simulated as well.

In any situation in traffic, an autonomous vehicle is able to recalculate its route according to local center agent's advice. This is a useful capability in urban traffic environment like taxis in traffic, they always communicate with other taxi drivers and related taxi center.

3.3.2.4 Velocity update behavior

This is a common behavior for vehicles. None of the vehicles in traffic has stable limit profile and it is assumed that this speed profile has a normal distribution [42].

For simulating drivers choose of speed, mean of speed and variance parameters are chosen randomly in specific range. These ranges differ from driver to driver.

Every cycle, vehicle checks nearest cars or cars in range, and according to the result, vehicle updates its speeds, Figure 3.11 shows this process. If there is no vehicle around, it choses its speed regarding to related distribution, else it changes its speed to follow ahead vehicle or lower it to take a precaution.

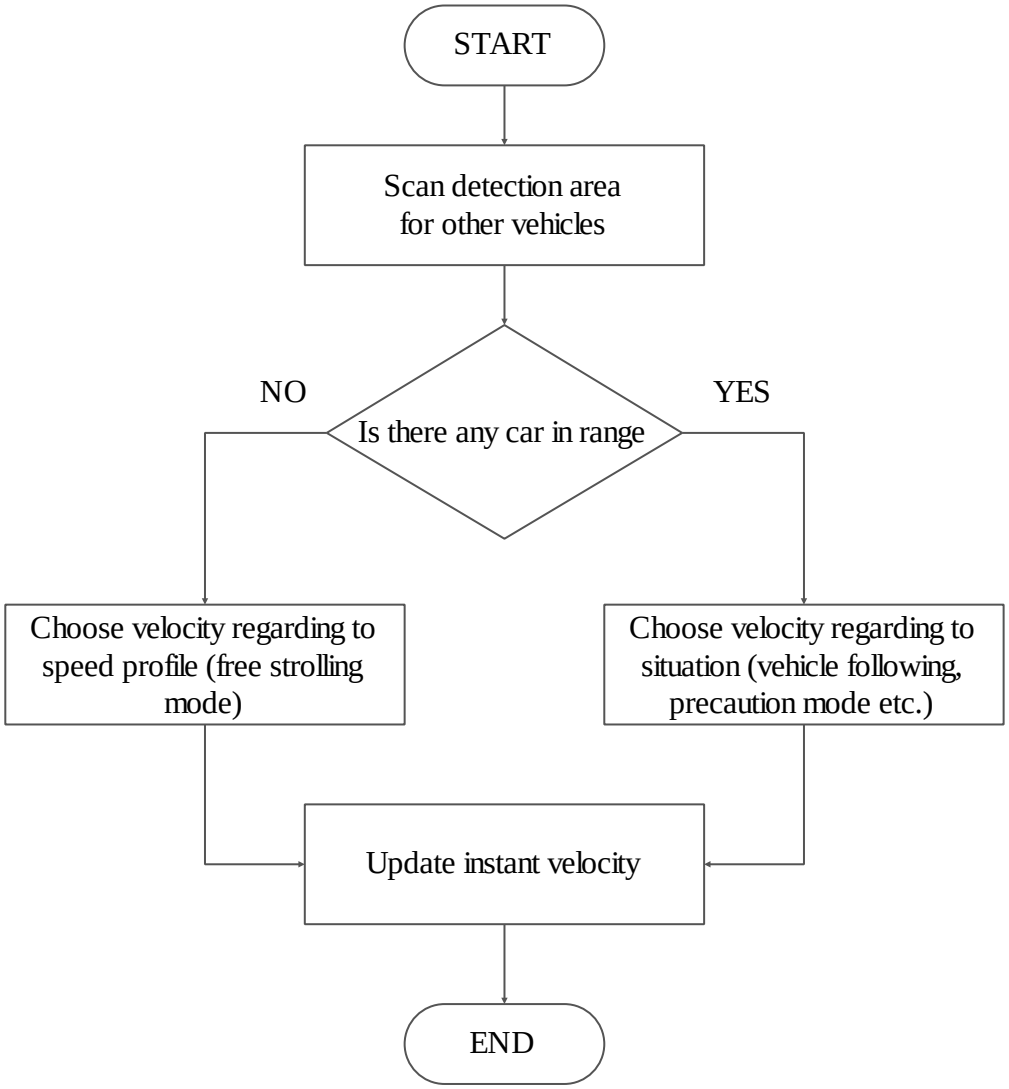

Figure 3.11 : Velocity update behavior flow chart

After velocity is updated, route following behavior will use this speed to calculate next position and store in agent specific object in every cycle. According to [40], driver speed chooses has a normal distribution characteristics, in this behavior, driver tries to choose it's velocity according to below calculations. If there is no dangerous situations around or road is empty, driver choose vehicles speed accordin to him/her

driver level plus a variance in speed. If it has good driving skills, change in speed will be very small in next calculation cycle. $v_{(i+1)}$ represents next speed choose, $v_{(i)}$ is instant speed when calculating and DriverSkillCoef is between 0 and 1.

$$v_{(i+1)} = v_{(i)} \pm \Delta v$$

$$\Delta v = (1 - DriverSkillCoef) \times deviationRange$$

$$v_{(i)} \cong initial\ velocity$$

When vehicle is following mode, that means some one is leading the vehicle and it have to follow it. In this case, vehicle chooses a speed between maximumSpeed and its defined average speed plus or minus deviation. Equation for this choose is detailed below.

$$averageSpeed \pm \Delta v \leq\ choose\ of\ \ v_{(i+1)} \leq min(aheadVehicleSpeed, v_{(i+1)})$$

3.3.2.5 Route following behaviors

Major difference between autonomous and manual vehicles is the route following behaviour. Manual vehicle agents generally have unpredictable behaviors in traffic whereas autonomous vehicle agents have smooth driving behaviors. If you imagine an urban traffic in low scale, this behavior variation may not affect overall traffic, but when scale is getting larger, nobody can predict its effects easily. In this study, simulation results will give us such a clue or first insights on a mixed traffic flow with multi-agent structure. Every microscopic movement may have effect on macroscopic scale, however deriving general rules on how this affect evolves is out of the scope of this thesis.

After initial route is calculated and server is informed, vehicle agent starts its movement. Every small step was calculated and agent starts to follow this detailed path. Vehicle agent moves "v" number of steps in every tick where "v" represents vehicle agent's velocity.

$$position_{(i+1)} = position_{(i)} + v_{(i)}$$

It occurs actually in normal circumstances which mean there is no traffic or other vehicle agents in detection circle. Unless any vehicles are around, vehicle agent

identifies their driver types and calculates next moves. Simplified route following behavior flow chart is shown Figure 3.12.

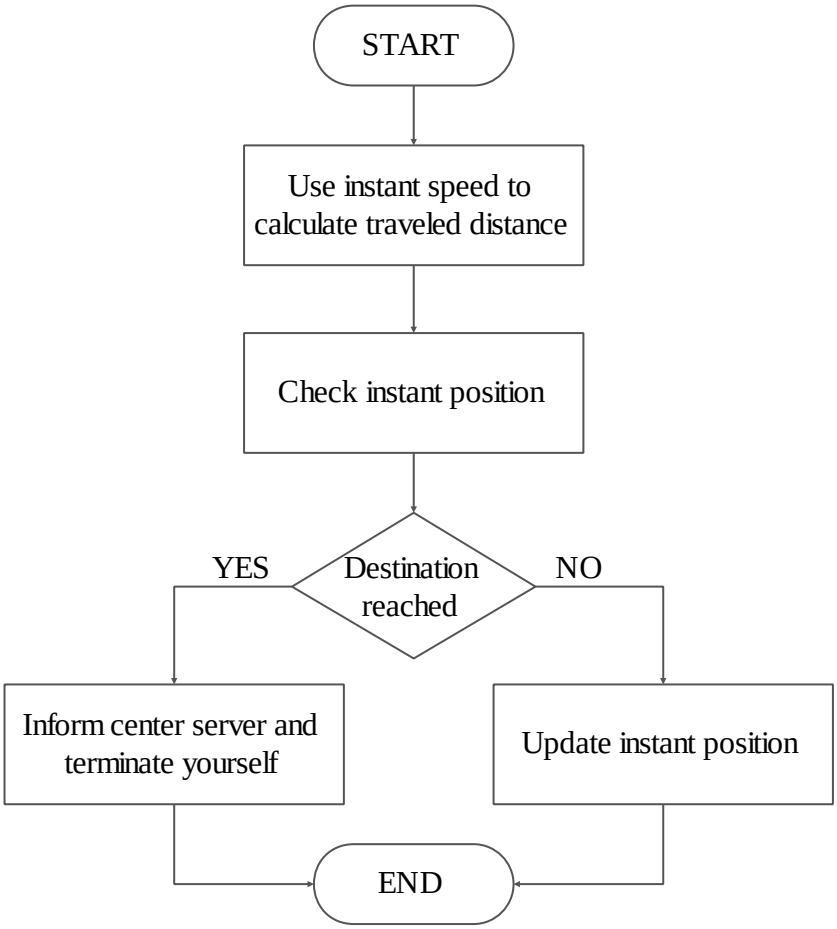

Figure 3.12 : Route following behavior of autonomous cars in simulation

It is not possible to integrate all traffic rules in simulation environment like priority at crossing edges when vehicles come face to face. Therefore, to define priority, a procedure is implemented in route following behavior. First, born agent takes priority at crossing edges and the other vehicle waits for prioritized vehicle move to calculate next step. Every decision or route following behavior result written to database and are stored in vehicle's instant properties object. To remind agent's instant properties object is send to center server agent in every reporting cycle.

Manually driven vehicles route following behavior is different than autonomous vehicles and represent human behavior variance in simulation and they are used to generate disturbances for our approach. Destination points of all manually driven vehicles are unknown and they have unpredictable behaviors.

For manually driven car, simulation picks a point to start and places manually driven car at that point but never knows where this journey will end. Manually driven vehicle picks a random point which is start points neighbor and picks another point when it reached destination. And then destination point becomes the new start point and the update cycle goes on like this until simulation ended. The random strolling behavior of a manually driven vehicle is shown in Figure 3.13 [43]

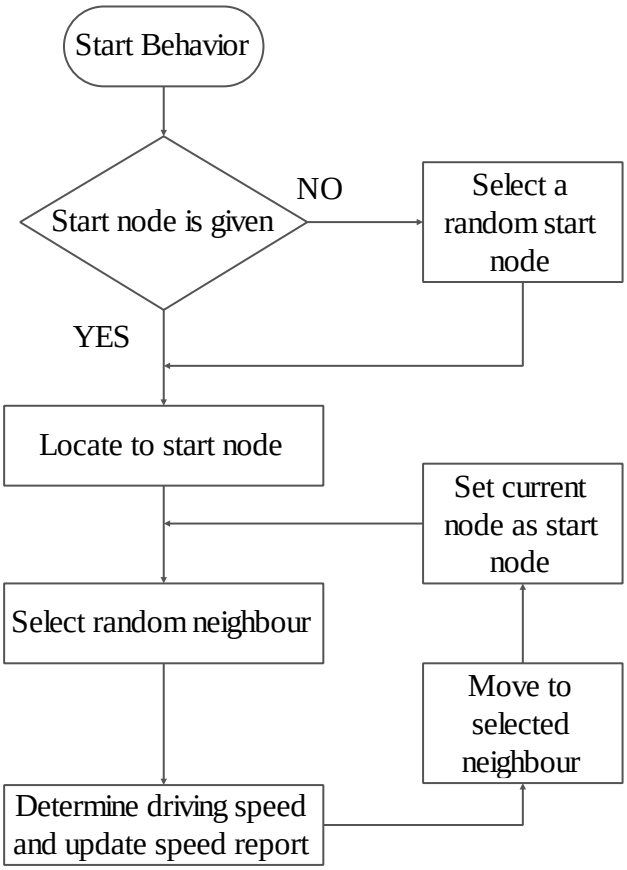

Figure 3.13 : Route following behavior of manual cars in simulation

3.3.2.6 Logging behavior

All events in simulation is logged by user defined logger class in simulation framework. MySQL is the world's most used open source relational database management system and is chosen for database instance in simulation framework. And MySQL Workbench is a next-generation visual database design application that can be used to efficiently design, manage and document database schemata. It is available as both, open source and commercial editions.

Events below are logged in database

- Vehicle Agent instant positions
- Accidents
- Any extraordinary situations in route, overloaded routes
- Heavy traffic on route
- Vehicle's planned routes, and any forced changes
- Vehicle's start point, destination point, desired travel time, delays on this path

All these events are inserted in related tables in database named ITS with simulation step and instant timestamp by every agent and some cases are logged by local and global center agents like high density traffic. Delays are logged in delays table, positions are logged in positions table.

Logger agent serves for this purpose and it is written using Singleton design pattern to increase performance. For handling log request at MySQL site, routines in MySQL database is created, these routines expect input parameters to insert or update related tables. First, a simulation is created in simulation table with a given simulation name and now simulation has unique identity, and all other events will be logged with this ID. This join may be named as foreign key.

After simulation ended, all relational records are grouped to generate conclusion results.

3.4 Visualization

Asdad In order to track vehicles behaviors in simulation environment, plotter agent is developed and implemented. As mentioned in agent behavior section, this agent serves for plotting process. Agent runs the MATLAB script periodically that gets all instant positions and vehicle types of all alive agents and plot them using MATLAB internal plot function. And only dynamic parameters while simulation starts are number of agents. So there is no need to an interactive user interface, showing instant positions is enough for tracking an agent's behaviors. In every five seconds, agents instant positions are plotted on city map with their unique properties like their identities. In this simulation, agent born orders are used to identify them on city map.

MATLAB built-in plotting function is used to generate sample screenshot of city map and agent positions are shown in Figure 3.14.

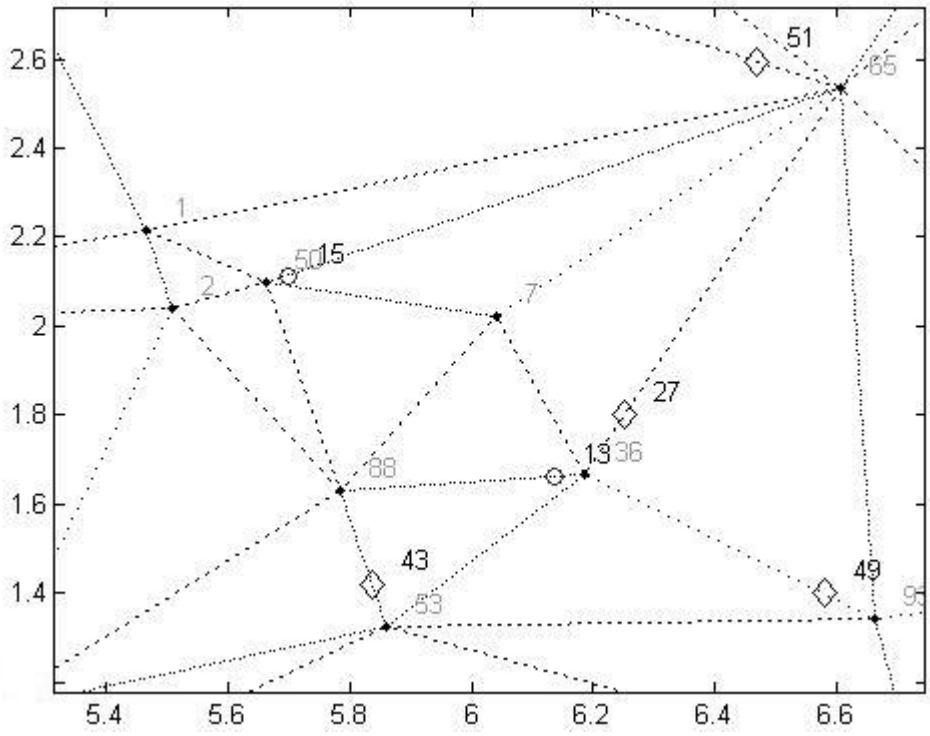

Figure 3.14 : Visualization of instant agent positions on urban map

Above image is a zoomed in MATLAB figure, to distinguish autonomous vehicles and manually driven vehicles, different shapes are used when plotting. Diamond shaped objects represent autonomous vehicles and tiny circles represent manually driven vehicles. Also to take attention, circles are plotted in red color and autonomous vehicles are plotted in blue. To track specific vehicle and not to scramble up with others, agent's identity number is also plotted near vehicle.

4. TEST RESULTS AND FINDINGS

In order to analyze the effect of manually driven cars proportion in mixed traffic flow with multiple agent types (i.e. autonomous, semi-autonomous, and manually driven) in an urban scenario, several tests are conducted on the simulation platform developed based on JADE. In this section, the testing environments, limitations, test cases, results and major findings are given.

4.1 Testing Environment

All simulation environments and components are developed on a laptop that gives a reasonable performance. Also debugging made on same computer with not more than 50 agents. If this limit exceeded, some performance problems occurred.

During the tests a high performance workstation is used. All simulation and simulation components (JAVA, JADE, MATLAB, MySQL, Agent Platforms, and Agents) are executed on a powerful workstation computer having the following configurations:

- Intel Xeon CPU E31240 3.30 GHz processor,
- 16 gigabytes of RAM, and
- Windows 7 Professional 64 bit operating system.

In all machines, Java Runtime Environment Version 7 is used.

4.2 Limitations

In the real-life scenario, each agent is assumed to run its own device as embedded systems. In this simulation, all agents including global center, vehicle, plotter agents, MATLAB run time engine, MySQL, JADE framework are executed on the same computer during the tests using single CPU. Because of this, there appeared some limitations in the number of agents in the simulation and size of city map used. In JADE framework, each behavior runs a separate JAVA thread and the agent itself is

another thread so that each vehicle agent creates multiple threads in the Java Virtual Machine (JVM). As the number of active threads in JVM increases, it is more likely to face thread synchronization problems and deadlocks appear in the JAVA runtime. Therefore, it is limited to 300 complex agents on the same container not to cause some threads of JADE to get blocked. When this happens, communication platform of JADE overloads and some delays and loses occurs.

4.3 Map Building

In the scope of the study, apart from the simulation of real city map, a generic map drawing engine has been developed in MATLAB. During the tests, an imaginary road map is generated as shown in the Figure 4.1. Topology of the generated map has complex and can be considered as a grid structure with Delaunay principles. There are about 100 nodes in total and more than 220 edges between these nodes. These edges are suitable for round trip, they all have two separated ways for going and returning. The reason of this limitation is to not to allow overtake, which makes the simulation environment more complex.

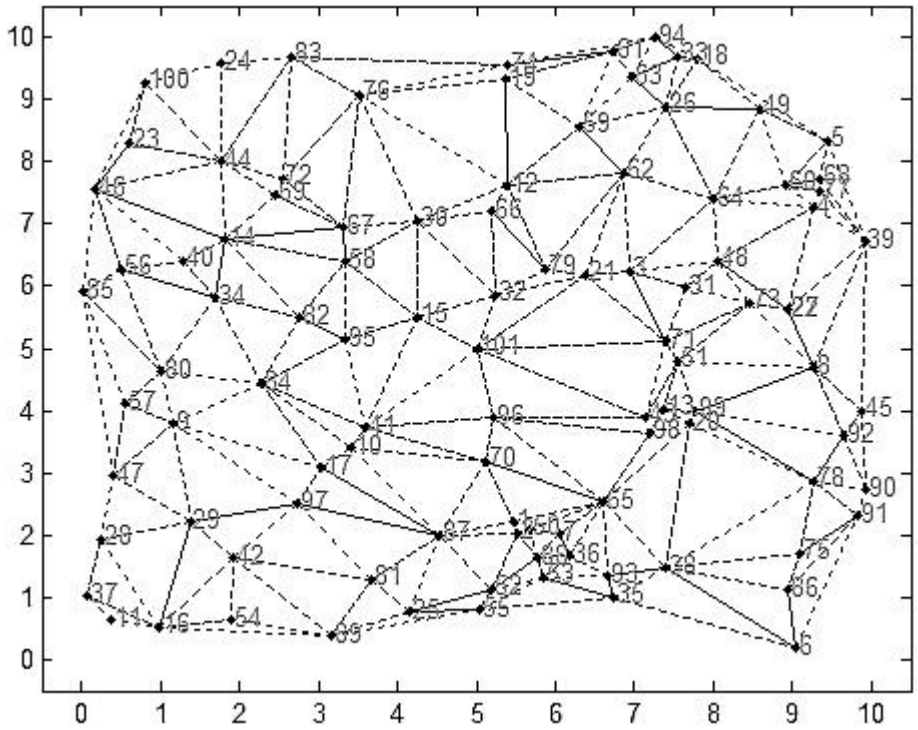

Figure 4.1 : Sample city map

Proportion of 85 percent is used for best scenario and 15 percent is used for worst scenario. First case represents the entrance of autonomous vehicles in urban city

traffic and last case represents the dominant times of autonomous vehicles. Remaining cases are used for representing transitional periods.

4.4 Test Cases

All cases simulated using stochastic approaches. As mentioned before, simulation and all objects take random parameters which obey normal distribution. For example, manually driven vehicle 1 travels from node 67 to node 23 in first simulation, but second simulation it travels from node 45 to node 89, and also initial parameters like initial velocity, driving skills are initialized in a range from simulation to simulation randomly.

Every day, city traffic changes randomly and it does not follow a pattern, that is why everything in simulation has random parameters. To analyze effect of manually driven vehicles on full autonomous cars, different proportions are used for simulation input. Same proportions simulated more than one times to obtain statistically meaningful data and to obtain generic trends that are not dependent on randomness. For example, in one of the simulation every car may start from same neighborhood and travels to same region and this may cause a high traffic density for manual cars.

Number of agents in simulation is limited to 300 because of performance issues, and proportion of different driver types in total manual drivers is kept the same in every simulation. For example, half of the drivers in urban traffic act like aggressive drivers.

All test cases and proportion of agents in these cases are shown in Table 4.1.

Table 4.1 : Test cases and proportions

	%15 Auto.	**%25 Auto.**	**%50 Auto.**	**%75 Auto.**	**%85 Auto.**
autonomous	45	75	150	225	255
aggressive	128	113	75	38	23
cautious	77	68	45	23	14
delayed	51	45	30	15	9
Number of Total Cars:	300	300	300	300	300

4.5 Test Results

Every test case was simulated five times to get better results and reduce chance or randomness effect in results. In addition to obtain some results, average of all five simulation results is used. Table below shows the instantaneous speeds of randomly chosen vehicles for first 120 simulation steps. Every vehicle is picked from different vehicle types to show characteristics of speed control and to reflect the real city traffic, this sample speed profiles queried from the simulation result that has low proportion of autonomous vehicles in it. Values are in units per step that is about 50 km/hour in real urban traffic. That is also urban speed limit and autonomous car travels without exceeding it. If autonomous vehicle's detection area is suitable for speeding, it travels nearly at 50 unit/step in simulation environment. That means in one simulation step vehicle travels 50 units. Recorded velocities of sample vehicles are shown in Table 4.2.

Table 4.2 : Recorded speed profiles

Simulation Step	Delayed (unit/step)	Cautious (unit/step)	Aggressive (unit/step)	Autonomous (unit/step)
10	32	38	53	50
20	32	35	53	50
30	32	37	50	48
40	32	37	50	48
50	35	37	50	50
60	31	37	50	47
70	29	36	50	47
80	33	36	53	49
90	33	36	53	49
100	29	34	54	49
110	29	38	47	48
120	26	40	52	48

As seen above autonomous vehicles has more stable velocity profile than others as expected. Autonomous vehicle never exceeds urban speed limit and travels at

maximum allowed speed if all conditions are convenient for speeding. Also an aggressive driver has a stable speed profile when compared with others but they are a bit impatient if urban speed limit is considered. Its driving skills may be better than autonomous vehicle but this does not mean that it can speeding. Cautious type driver has the most desirable speed profile I think, stable and not close to urban speed limit.

One random vehicle was chosen from every group and first 120 simulation steps are recorded. Graphical representation of this speed profiles are shown in Figure 4.2.

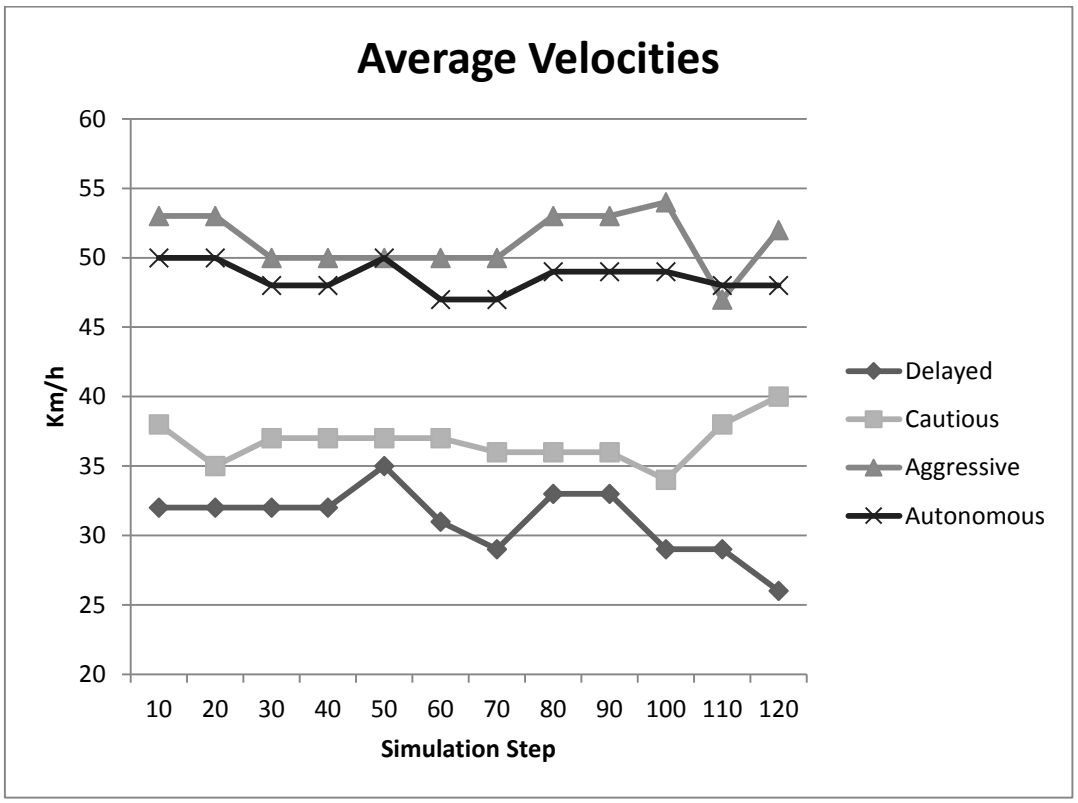

Figure 4.2 : Recorded velocities (unit per step)

As shown in graph, velocity records of vehicle groups are as expected, these shows that vehicle agents behaves according to their characteristics.

For all test cases, instantaneous velocities of all vehicles are logged and SQL queries are written to obtain average velocity of all vehicle types according to changeable proportion of autonomous vehicles. All choices and behaviors are logged in to tables and some relations are defined between tables to store different kind of datas with same simulation identity and vehicle identity. To gather different datas from different tables, some sql built in functions are used.

Average velocities per driver types are calculated using results of five different simulation runs, Table 4.3 shows average velocities of related driver types.

Table 4.3 : Average velocities in simulations

	%15 Auto.	%25 Auto.	%50 Auto.	%75 Auto.	%85 Auto.
aggressive	39,12	40,09	39,48	40,23	39,87
autonomous	22,23	24,88	31,18	41,53	43,82
cautious	34,31	35,36	35,95	36,27	36,33
delayed	26,05	27,37	32,09	32,67	29,02

Every vehicle had their own routes in every simulation and this was based on randomness. Most probably every vehicle has different routes in every new simulation and also different than other vehicles. There is 101 nodes in simulated city map, and one chosen point can be a start node or end node, so traveling from A to B is not equal to traveling from B to A. These are two different routes and also choosing 2 different points from city map is a combination

$$C(101, 2) * 2 = 10100\, routes$$

Five different simulation gives another combination. Therefore, pulling same route in every simulation from this combination list has very low probability. So it is assumed that every vehicle travels different route in every simulation. And average speed of all this simulation runs are shown in Figure 4.3.

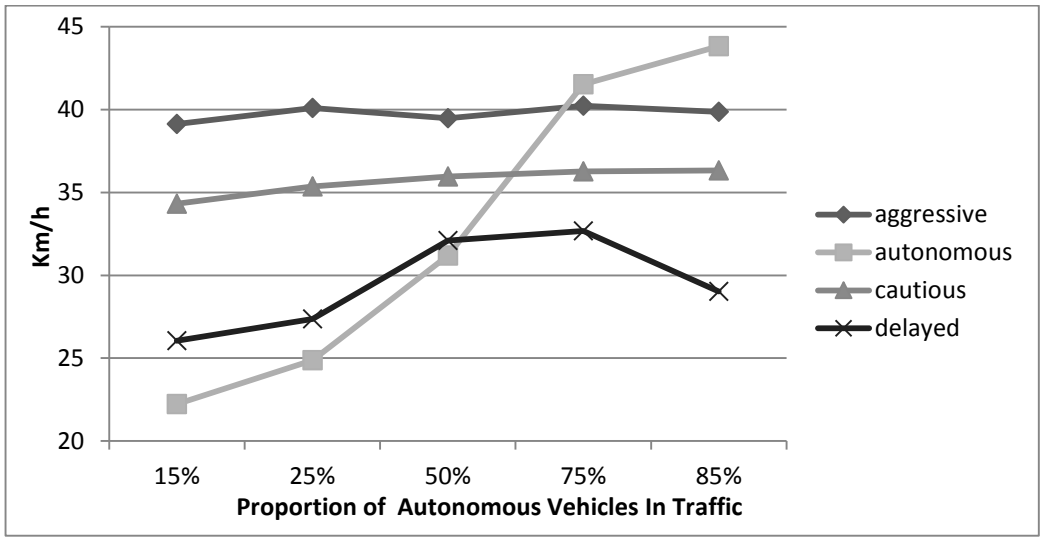

Figure 4.3 : Average velocities of vehicle groups

To summarize, for first case that is worst one with 15 percent proportion, autonomous vehicles has the lowest speed average in five different simulation run. Proportion of autonomous vehicles increases, their journey in simulation environment gets easier and more comfortable while the others remains nearly same. It is interesting that, cautious driver's journey in simulation also getting more comfortable.

Also time of vehicle's born and die events in simulation are logged to calculate real travel time. It differs from expected, according to traffic situations and especially other car's behaviors. Using SQL built-in function avg() all delays are calculated by grouping them according to their driver types. Table 4.4 shows average delays.

Table 4.4 : Average delays in percentage

Autonomous Vehicle's Proportion In Traffic	Autonomous vehicle's average delay in percentage
%15 Autonomous	28,93 %
%25 Autonomous	25,67 %
%50 Autonomous	16,13 %
%75 Autonomous	7,94 %
%85 Autonomous	7,12 %

As it seen, while proportion of autonomous vehicles in traffic increases, average delay times decreases. Delay in steps does not reflect the real delay, so percentage delay is used. Every vehicle calculates its estimated travel time while route-planning behavior runs. Moreover, at the end of the journey, vehicles log real travel time too. In this case, delay is the difference between estimated and real travel time. Only delay of autonomous vehicle's journey may be calculated in this simulation, because manually driven vehicles cannot calculate estimated travel time.

To detect the situation they may cause an accident, relevant data are logged in MySQL, if an autonomous car is surrounded with manually driven cars and they are closer than following distance, this is logged as dangerous attraction. In addition, following an aggressive or being followed closer by an aggressive driver are identified to be dangerous. These dangerous attractions are logged and average dangerous events per vehicle is shown in table 4.5.

Table 4.5 : Dangerous actions

Autonomous Vehicle's Proportion In Traffic	Average of Dangerous Events per Autonomous Vehicle
%15 Autonomous	8,17
%25 Autonomous	7,79
%50 Autonomous	5,41
%75 Autonomous	3,26
%85 Autonomous	1,83

4.6 Findings

When the result summaries of these five configurations are compared, there are several findings observed depending on the proportion of autonomous cars in simulation. Every configuration simulated five times and then a general observation is done by considering all results.

First, speed profiles show that choosing velocity behavior close to expected, autonomous vehicles has a stable profile unless elderly does not. Delayed or elderly driver has difficulties while controlling vehicles speed. Cautious driver's speed profile also reflects a cautious human driver, does not have instantaneous peaks in profile. The best, autonomous cars cruise control technology mimics a very experienced, cautious driver profile.

While proportion of autonomous cars in simulation rises, their travels become more comfortable. Average speed profile of every configuration proves this. In the first configuration, that has the lowest autonomous car proportion, all autonomous car's average speed is about 22 unit/step. And while proportion increases, average speed also increases. In the last configuration, which is the best scenario for autonomous vehicle, average speed reaches about 43 unit/step. This shows that, with the increasing number of autonomous cars in the traffic, travelling becomes easier and less time consuming.

Recorded delays also support last finding, every autonomous vehicle have an expected travel time before starts journey, and after vehicle reaches its destination point, end of journey is logged. Delays for autonomous cars are becoming shorter

from first configuration to last. Therefore, autonomous vehicles get familiar to traffic environment while proportion of them increases.

It is observed that, with ITS technology implementation, number of dangerous attractions in urban traffic decreases. Autonomous vehicles are safer than manually driven ones, and they are equipped with high technology sensors and warning system. These features make autonomous vehicle's detection capabilities wider than human drivers.

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 Future Work

In this study, imaginary city map is generated to simulate urban city traffic. This imaginary map increases complexity in path planning and route following behavior. Every single node connects to at least two other nodes in map. So there are always alternative paths between two points in the city. This map would be changed with a real city map taken from Google Maps or new drawn real map to analyze real situations in urban traffic.

Three main driver types for manual vehicles are implemented in this simulation, aggressive, cautious and elderly. New driver types would be add in simulation environment like beginner, intoxicated, incautious etc. This would increase variety in driver characteristics in traffic. Also number of behaviors would be increased to mimic real human driver.

All cases simulated on a workstation and because of performance problems, at most 300 vehicle agents are added in this traffic environment. In a high performance computer, more than 100.000 vehicle agents would be simulated and result would be analyzed again. Another approach may be useful, synchronizing all agents and computation of every agent's behavior one by one according to others.

Intelligent routing or optimal routing algorithms would be implemented and tested in this environment. Vehicles may be taken as a group, and reaching destinations must be assumed like a group goal. This feature may be a good solution for autonomous vehicles to stay out of traffic jam.

As for future perspectives, these simulations should be tested in a real case study, which will thereby determine the real practicality of this study. One suggested work is to study new coordination methods for autonomous vehicles within common urban traffic scenarios. And another goal is to make software agents more like human drivers except autonomous ones.

5.2 Conclusion

Traffic simulation for urban traffic is a complex and unpredictable environment in which all conditions may change instantaneously and cause another effects. Autonomous cars will be available in near future, and a transition period is inevitable. This period must be well analyzed and prepared for both human drivers and autonomous vehicles. It is possible that traffic may be a chaotic environment when autonomous vehicles meet manually driven vehicles.

In this study, a multi agent simulation is designed and developed in which autonomous vehicles and manually driven vehicles share same environment and travel on same routes. Traffic data is collected and broadcasted by central server agents, all autonomous agents used this data to know real traffic situation and other vehicles positions. Autonomous vehicle's all behaviors are logged in to database and related results are obtained.

Results show that, route following behavior of autonomous vehicles is directly related with its proportion in urban city traffic. Autonomous vehicles travel freely, organized, faster and safer while they become dominant in traffic. Their speed profile becomes more stable, average travel speed increases and delay in expected travel time decreases. In addition, their increased percentage improves the cautious driver's speed profile. This means, autonomous vehicles may overcome the congestion problem.

By the help of ITS structure we propose, it is observed that ITS technology would solve congestion and traffic safety problems in near feature.

REFERENCES

[1] **Url-1** <*http://www.nytimes.com/2010/10/10/science/10google.html?pagewanted=all&_r=0*>, date retrieved 15.03.2013.

[2] **Url-2** <*http://www.darpa.mil/*>, date retrieved 17.03.2013.

[3] **Sejoon Lim and Daniela Rus**, 2012, Stochastic Distributed Multi-Agent Planning and Applications to Traffic.

[4] **Javed A. and Sejoon L. and Daniela R.**, 2012, Congestion-aware Traffic Routing System Using Sensor Data.

[5] **José Luis Ferrás Pereira,** An Integrated Architecture for Autonomous Vehicles Simulation, Phd Thesis, PORTO University, Portugal

[6] **Url-3** <*http://sites.ieee.org/itss/*>, date retrieved 17.03.2013

[7] **Url-4** <*http://en.wikipedia.org/wiki/Electronic_toll_collection*>, 17.03.2013

[8] **Bertini, Robert L. and Monsere Christopher M.**, 2005. Benefits of Intelligent Transportation Systems Technologies in Urban Areas: A Literature Review, Final Report, Portland State University, Portland.

[9] **Xue Y. and Jie L. and Feng Z.**, A Vehicle-to-Vehicle Communication Protocol for Cooperative Collision Warning

[10] **Schagrin M.,** Vehicle-to-Vehicle (V2V) Communications for Safety, <*http://www.its.dot.gov/research/v2v.htm*>, date retrieved 01.04.2013

[11] **Schagrin M.,** Vehicle-to-Infrastructure (V2I) Communications for Safety,

<*http://www.its.dot.gov/research/v2i.htm*>, date retrieved 02.04.2013

[12] **Claes R. and Holvoet T.,** 2011, A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems

[13] **Ghiani G. and Guerriero F**., 2003, Real-time vehicle routing Solution concepts, algorithms and parallel computing strategies

[14] **Chem S., Sun Z. and Bridge B.**, 1997. Automatic Traffic Monitoring by Intelligent Sound Detection, *In Proceedings of the IEEE Conference on Intelligent Transportation Systems*.

[15] **Forren J. and Jaarsma D.**, 1997. Traffic Monitoring by Tire Noise. *In Proceedings of the IEEE Conference on Intelligent Transportation Systems.*

[16] **Nishizawa S., Cheok K., Young W. and Zhao W**., 1997. Traffic Monitor Using Two Multiple-beam Laser Radars. *In Proceedings of the IEEE Conference on Intelligent Transportation Systems.*

[17] **Url-5** <*http://en.wikipedia.org/wiki/Dijkstra's_algorithm*>, 22.03.2013

[18] **Url-6** <*http://en.wikipedia.org/wiki/Floyd–Warshall_algorithm*>, 23.03.2013

[19] **Url-7** <*http://en.wikipedia.org/wiki/Johnson's_algorithm*>, 22.03.2013.

[20] **Url-8** <*http://en.wikipedia.org/wiki/Bellman–Ford_algorithm*>, 25.03.2013.

[21] **Url-9** <*http://en.wikipedia.org/wiki/A*_search_algorithm*> 23.03.2013.

[22] **L. Fu, D. Sun, L.R. and Rilett,** Heuristic shortest path algorithms for transportation applications: State of the art, *Computers & Operations Research*, 33, 11, 2006: 3324-3343.

[23] **Guzolek J. and Koch E**. Real-time route planning in road network. Proceedings of VINS,September 11–13,1989. Toronto, Ontario,Canada, pp. 165–9.

[24] **Hart, P.E. Nilsson, N.J. Raphael, B**. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *Systems Science and Cybernetics, IEEE Transactions* , 4, 2, 1968: 100-107.

[25] **Url-10** <*http://www.docstoc.com/docs/11896525/Simulation-of-Dijkstra-Routing-Algorithm*>, 03.04.2013.

[26] **Klasen Volker,** 2010, Verifying Dijkstra's Algorithm with KEY, *Master Thesis*, Koblenz University, Germany

[27] **Bigus, J.P.** (2000). Agent Building and Learning Environment. *Proceedings of the International Conference on Autonomous Agents 2000,* Association for Computing Machinery, p108-109. Barcelona, Spain.

[29] **Lange, D. B. and Mitsuru, O**., 1998. Programming and Deploying Java Mobile Agents Aglets. 1st. Addison-Wesley Longman Publishing Co., Inc.

[30] **Poslad, S., Buckle, P. and Hadingham R.**, The FIPA-OS agent platform: Open Source for Open Standards, 2000: 55 .

[31] **Url-11** <*http://jade.tilab.com/*> , 30.04.2013

[32] **Jeon H., Petrie C., Cutkosky M.,** JATLite: A Java Agent Infrastructure with Message Routing, , *IEEE Internet Computing*, 4,2, 2000.

[33] **Url-12** <*http://www.eclipse.org/*> , 03.03.2013.

[34] **Url-13** <*http://www.fipa.org/specs/fipa00037/SC00037J.html*>, 30.03.2013.

[35] **Url-14** <*http://jade.tilab.com/doc/programmersguide.pdf*>, 22.03.2013.

[36] **Bellifemine F., Caire G. and Greenwood D.,** 2007. Developing Multi-Agent Systems with JADE, p. 59.

[37] **Url-15** <*www.mathworks.com/help/matlab/ref/delaunay.html*>, 27.04.2013.

[38] **Url-16**

<*http://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Heuristic_search/Astar_Search*>, 22.04.2013.

[39] **Stuart Russell and Peter Norvig** (2009). *Artificial Intelligence: A Modern Approach (3rd Edition)*, Prentice Hall

[40] **Demir M.,** Developing intelligent agents for traffic education, *Phd Thesis*, Gazi University, Ankara

[41] **Url-17** <*http://www.oodesign.com/singleton-pattern.html*>, 17.02.2013.

[42] **Ko Joonho**, 2004, Characterization of Congestion Based on Speed Distribution - A Statistical Approach Using Gaussian Mixture Model.

[43] **Öcal K.,** A-Star Based Route Planning Using Real-Time Road Traffic Density Data, *Master Thesis*, Atılım University, Ankara

CURRICULUM VITAE

Name Surname:	Oğuz Ali Ekinci
Place and Date of Birth:	Burdur, 1986
Address:	İstanbul
E-Mail:	oguzaliekinci@gmail.com
B.Sc.:	System Engineer/ Automation and Control Engineer