Project Id: 15BITGP0126
(For Office Use only)

Agent-based Simulation of Cooperative Driving Maneuvers of Autonomous Cars

A PROJECT REPORT

Submitted in partial fulfillment for the award of the degree of

B.TECH

in

Information Technology

by

Aditya Raj 11BIT0155

Under the Guidance of

- Prof. Dr. Jörg P. Müller (Director Department of Informatics, Clausthal University of Technology)
- Dr. Maksims Fiosins and Dipl.-Wirt.-Inf. Malte Aschermann (Researcher Institut für Informatik, Clausthal University of Technology)
- R K NADESH (Assistant Professor(Sr.) VIT University)

Project Id: 15BITGP0126
(For Office Use only)

School of Information Technology & Engineering

06 - 2015

DECLARATION BY THE CANDIDATE

I hereby declare that the project report entitled "Agent-based simulation of cooperative driving maneuvers of autonomous cars" submitted by me to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the degree of B.Tech.(Information Technology) is a record of bonafide project work carried out by me under the guidance of Prof. R K Nadesh. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore Signature of the Candidate

Date: 21.05.2015 Name: Aditya Raj

Project Id: 15BITGP0126
(For Office Use only)

School of Information Technology & Engineering [SITE]

CERTIFICATE

This is to certify that the project report entitled "Agent-based simulation of cooperative driving maneuvers of autonomous cars" submitted by Aditya Raj (11BIT0155) to Vellore Institute of Technology University, Vellore, in partial fulfillment of the requirement for the award of the degree of B.TECH in INFORMATION TECHNOLOGY is a record of bonafide work carried out by him/her under my guidance. The project fulfills the requirements as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature

Prof.R.K.Nadesh, Programme Chair
INTERNAL GUIDE

Prof.R.K.Nadesh
Programme Chair
B.Tech(Information Technology)

The project work is satisfactory / unsatisfactory

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

The internship opportunity I had with the research group headed by Prof. Dr. Jörg P. Müller at TU Clausthal, which includes highly motivated and passionate undergraduate colleagues; Christoph Kuper , Xin Zhou; and postgraduate seniors; Dipl.-Inf. Philipp Kraus, Sophie Dennisen, Thomas Hornoff, Nils Armbrecht, was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me though this internship period.

I am using this opportunity to express my deepest gratitude and special thanks to Prof. Dr. Jörg P. Müller, who in spite of being extraordinarily busy with his duties, took time out to hear, guide me in mathematically modeling the project and keep me on the correct path and allowing me to carry out my project at his esteemed organization.

I would like to express my gratitude to my supervisors Dr. Maksims Fiosins and Dipl.-Wirt.-Inf. Malte Aschermann for providing me enormous support during the tenure of the project via a regular weekly meet up to carry forward the progress of the project.

I am also thankful to the IAESTE organization for the financial support.

I am also grateful to the Agent Technology Center, Dept. of Computer Science at Czech Technical University in Prague, especially Prof. J. Vokrinek and Martin Schaefer, for providing me a chance to work on the AgentDrive simulator along with the provision of the expertise, and technical support in the implementation of autonomous maneuvering.

It is my pleasure to express with deep sense of gratitude to Prof R.K. Nadesh, Assistant Professor(Sr.) at VIT University for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor.

Nevertheless, I express our gratitude towards my family and colleagues for their kind co-operation and encouragement which help us in completion of this project.

Place: Vellore

Name of the Student: Aditya Raj

Date: 22.05.15

Table of Contents

List of Figures

Abstract

This bachelor thesis discusses the aspect related to Agent-based simulation of cooperative driving maneuvers of autonomous cars with a basic Java Applet simulation which is also integrated with a real time simulation platform AgentDrive and OpenDS. The model based on Applets is coded on Java Platform that provides 2D visualization of simulation environment. It also facilitates us to test several aspects of autonomous maneuvering. This includes a basic thread based model to control lifecycle of each agent. Further, to achieve co-operation among these agents, the concept of planning from second principle is utilized. Considering the two important interconnected applications, automated execution of maneuvers for driverless vehicle/autopilot and an Intelligent Driver Assistance system, as the prime goal, this thesis also presents a compelling overview to achieve the goal on a merged platform of a realistic traffic simulator and an open source driving simulator, AgentDrive and OpenDS correspondingly. The existence of common platform in both the simulation environments facilitated the merging and testing of former approach with the latter one. Also to lay down the strong basic foundations of deploying agents to solve maneuvering problems, future works with required changes in existing efficient maneuvering and path finding algorithms is highlighted.

1. Introduction

1.1. General Introduction

Traffic collision is one of the significant concerns affecting the whole world. Despite having strict traffic laws, the situation is not improving statistically. A rapid increase in the statistical figures for number of causalities due to traffic accidents every year demands the design and development of intelligent highway systems.

Da Lio and et. al. [1] made an important remark on the current transportation systems. He pointed out the fact that by replacing motor vehicles with horses, we have compromised in terms of communication capabilities of the carriage and the carrier. An intelligent being with the ability to demonstrate communication capabilities with its master had been used in transportation for thousands of years. But even after huge development in automobile industry to go along with the need of the hour for faster transportations, the gap remains unabridged. This means that the cooperation (we focus on this aspect, even if we know that interaction regards also "competition") occurs between two "sentient" systems; in our case, one is the human agent (the driver) and the other one is not anymore the animal, but the machine agent.

To develop a cost-efficient and an artificially intelligent solution towards efficient management of incoming and outgoing traffic by developing a strong co-operative planning among the vehicles has been a desperate effort area for many of the computer science researchers. Co-operative planning is defined as the principle of involving mutual assistance in working towards a common goal. Human drivers perceive limited and approximate information of the traffic surrounding them, so it becomes unfeasible to avoid traffic collisions in many situations. With limited computational speed and accuracy of human brain, the predictions with the perceived approximate information about the speed and position of vehicles on the highway may lead to colliding intentions to reach a certain goal. Agreeing on a consensus by allowing the predictions to be polled before execution is one of the better ways to avoid these kinds of common intent conflicts. For sharing such predictions with intelligent vehicles in traffic, we need to focus on the domain of designing intelligent systems using Multi-agent Simulations. We first need to go with

simulations because it is economically viable to test the scenario before actual deployment of planning principles and communication technologies on highways.

The development of automated and autonomous systems to enhance safety and prevention of accidents on the highways is the research trends in emerging technology. This project aims to design a cooperative driver model, which would improve the usage of the capacities of highways during reactive control of the users. These enhancements could provide more efficient usage of the highway infrastructures, which will reduce the probability of occurrence of accidents and provide safety. The proposed methodology is demonstrated through experimental setup.

1.2. Problem Statement

The objective of the project is to:

- Implement a cooperative driver model that is based on a vehicle trajectory planning augmented by techniques from the domain of distributed artificial intelligence and already utilized in multi-agent systems.

- Provide a multi-agent traffic simulation environment that can serve as a framework for evaluation of various co-operative planning strategies.

- Also, to Provide a solid base for a further research towards defining representative quality metrics and comparison of different approaches.

- Work with existing driving simulators and traffic simulators to demonstrate the principle of co-operative and safe maneuvering with multiple autonomous vehicles.

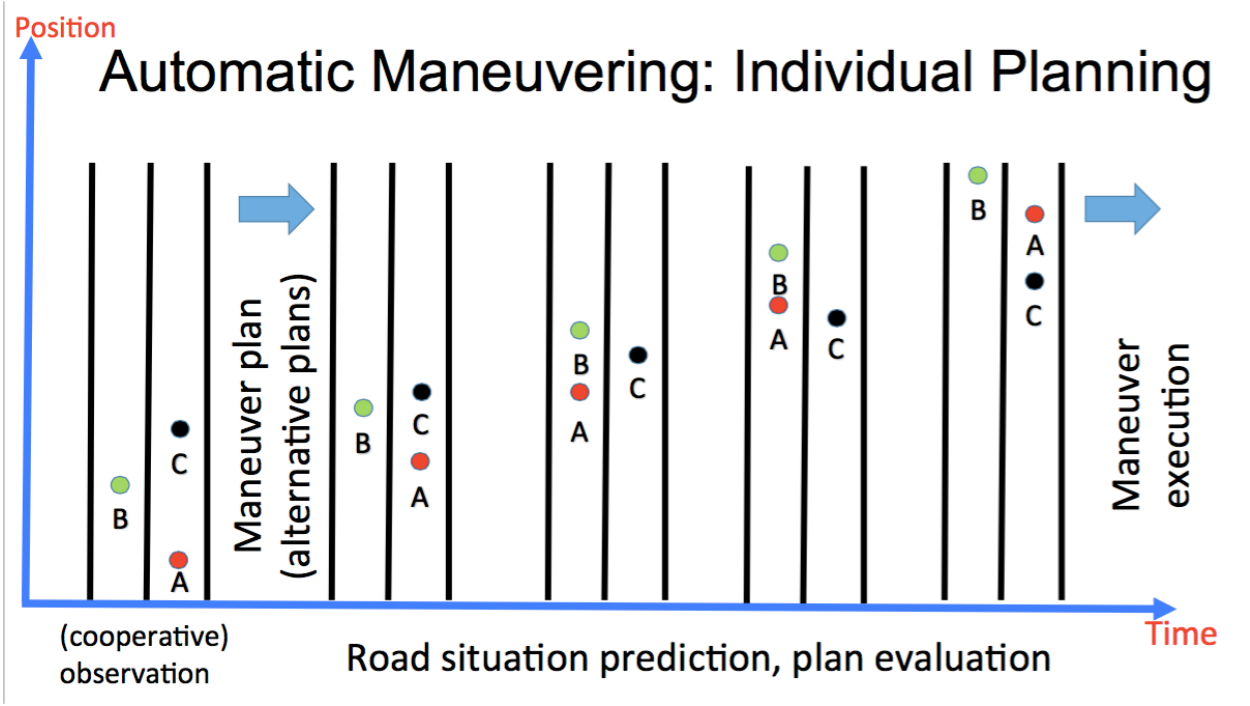


Figure 1 Current Problem

Explanation:

- In the diagram shown above, a practical and general approach to solve maneuvering problem is demonstrated. The vehicle wants to overtake vehicle C and thereby tries to construct a safe maneuver plan, given that the vehicle B is in lane left to A. The vehicle A waits for a safe plan to maneuver C, and until then it platoons vehicle C.

- After finding a safe plan, it changes lane to left lane. Further it crosses vehicle C and then comes back to right lane.

1.3. Proposed System Overview

Cooperative maneuvering aims at combining two recent technological developments:

1. The availability of automated vehicles, that build models of their environments and plan trajectories in situations of growing complexity, and

2. The technological foundations for connecting these vehicles, with each other and with traffic management using Vehicle-to-X (V2X) communication.

Cooperation builds on communication and automation and will enable new functions, increasing safety and efficiency of traffic through explicit coordination of (automated) driving maneuvers, covering the whole maneuver lifecycle from selection of suitable maneuvers over negotiation of parameters to maneuver execution, evaluation, and (if necessary) cancellation.

As Cooperative observation and situation prediction modules already exist, so it is not required to reinvent the wheels from scratch. Planning from the second principles is our focus area. According to this principle, several possible types of the maneuver; Straight Maneuver, LeftLaneManeuver etc.; are available in the library; they should be instantiated whenever a lane change is required.

In this project, I will investigate and evaluate different agent-oriented simulation platforms (e.g., MATI, AgentDrive) that enable modeling and simulated execution of cooperative maneuvers. In doing so, selected maneuvering scenarios will be specified, modeled and simulated with special view of the representation of maneuvers and the core local and collective decision-making functions for planning and coordination.

A practical approach to maneuvering vehicles:

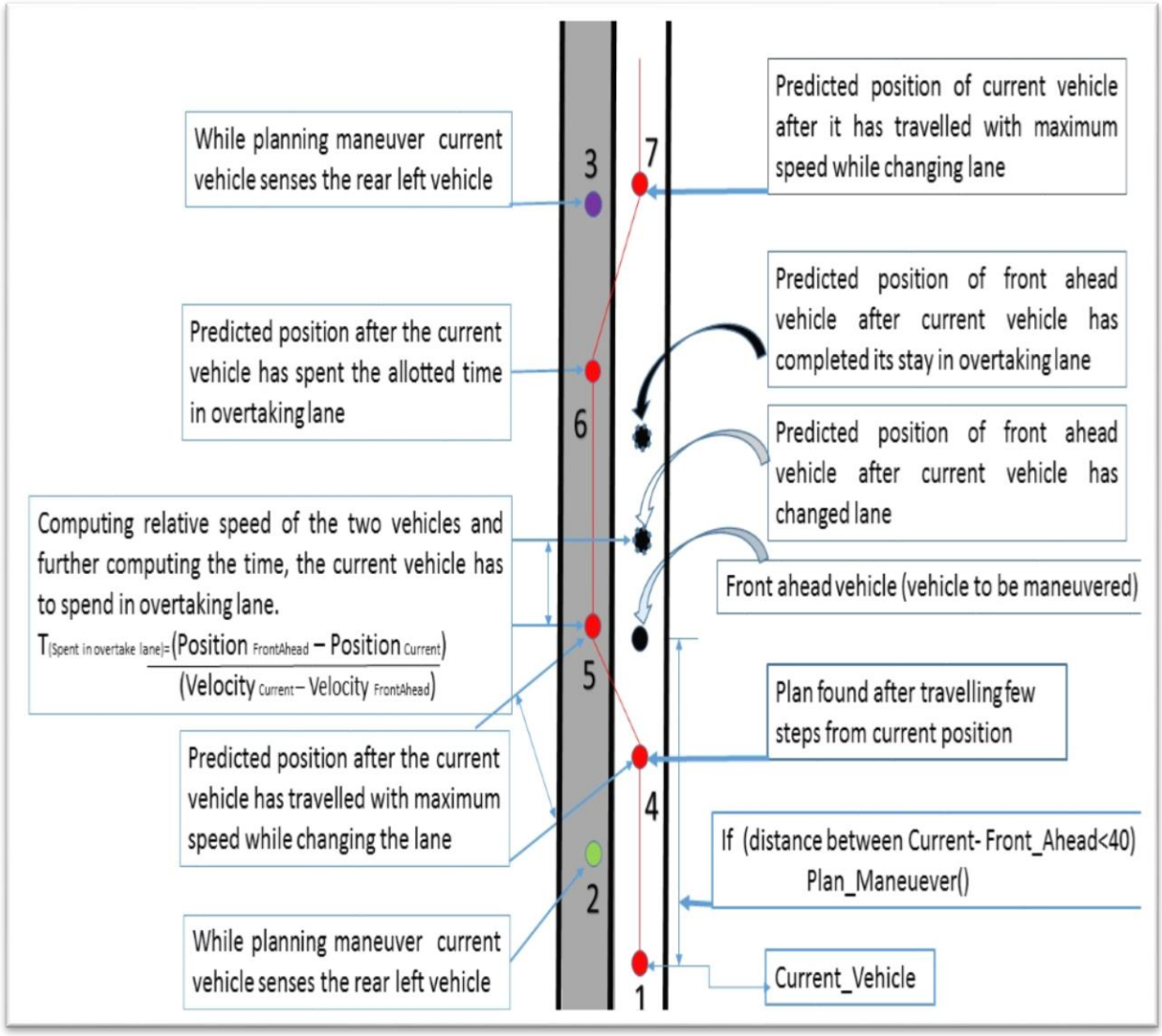


Figure 2 Practical Approach To Maneuvering

The human body is the general inspiration for artificially intelligent systems. For developing most of the intelligent systems, we imitate human observations and intuitionistic planning by replacing the sense organs with sensors, actuators and other communication devices. Interpersonal and Inter-bodily communications are the basics for carrying out activities comfortably in everyday life. For interpersonal communications linguistic, written and other sign language approaches exist. Intra-bodily communications is achieved by nerves, which act as a wire to transmit signals to the organs for the apt response. Keeping the same co-ordination and

communication basics in mind, to achieve co-operative driving, we deploy sensors to sense the neighborhood and further develop a response to the current situation. If a plan can be predicted by the mutual agreement between the co-operating vehicles, then it can be sent wirelessly for execution.

A human plans to maneuver a vehicle in front of it by hit and trial approach. This approach reduces the possibility of speed optimization as it almost seems impossible to predict the conditions, like speed, position, duration to maneuver and so on, while maneuvering. So there is a need to deploy an efficient algorithm to do the task efficiently. Replacing mental calculations with computer calculations is the only way to deal with this problem. A general approach to maneuver can be explained by the following points:

1. We Sense, if the vehicle in front is travelling at speed lower than our maximum speed.

2. If yes, then we look for a gap in left lane and also see if a high-speed vehicle is not approaching in left lane. Also if a rough estimate suggests that after maneuver we can safely return to current lane, maneuvering begins.
3. For performing a maneuver, we accelerate and change to left lane and further we go on accelerating (until the vehicle reaches its maximum speed) to overtake the vehicle.

4. After successful overtake and also having reached a safe distance for returning back to current lane, a right lane change is performed.

We can optimize this scenario once we exactly know our neighbors and further try to do a co-operative planning by precisely calculating the lane change speed, overtaking speed and duration of overtake for the vehicle planning to maneuver. On comparing the human approach to our current algorithm, it can be easily found out that a lot of similarities exist.

For a vehicle willing to perform maneuver, sensors act as the vehicle's eyes and thus provide exact location and speed of rear left and front left cars. Thus based on the calculations shown at every step in the above diagram, a maneuver is planned and executed.

2. Background

2.1. Background knowledge of Agents:

Defining Agent: An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through sensors. Agent's behavior is described by agent's function that maps any given precept sequence to an action. Tabulating agent functionalities can be an infinite list and therefore a bound on such an infinite list, by restricting to useful scenarios, can solve the purpose of associating an agent to a particular activity. The agent model described here ([2] and [3]) can be classified based on PEAS (Performance, Environment, Actuator, Sensor) criterion to mathematically model the behavior of the agent.

2.1.1. Specifying the task environment for our agent:

Agent Type	*Performance Measure*	*Environment*	*Actuators*	*Sensors*
motionAgent	Safe, travel at maximum speed, follow traffic rules	Autonomous vehicles, Manually-driven vehicles,	Steering, Accelerator, Brake, Display	Speedometer, accelerometer, Keyboard, VehicleSensor
PEAS description of the task for an automated vehicle				

a) Performance Measure: We would like our automated driver to drive at maximum possible speed for maximum duration without collisions. Minimizing trip time, avoiding violations of traffic rules and minimizing fuel consumption is required while driving on straight highway. In case of conflicts among performance measures trade-offs can be included.

b) Environment: The autonomous agent, motionAgent, currently deals with straight highway following European traffic regulations and the situation can be customized for any number of lanes on the highway. The roads can contain other autonomous vehicles and a manually driven

car. The environment can be adjusted to suit snowing or raining conditions by steering up/down the parameters like acceleration, mass, deceleration etc. of an individual car.

c) Actuators:

For an automated vehicle, the actuators include those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition to it we will need to output a display screen and perhaps some way to communicate between the vehicles.

Actions: Agents are assumed to have a repertoire of possible actions available to them, which transform the set of environment. Say, an action making a change in the X Co-ordinate of the agent will transform the current state of the agent to ChangeLeft or ChangeRight, depending on the positive or negative changes made in the current X Co-ordinate of the agent.

We have thus modeled an agent's decision function as from sequences of environment states or percepts to actions. This allows us to represent agents whose decision making is influenced by history. But it does not completely resemble the idea of maintaining states.

These agents have some internal data structure, which is typically used to record information about the environment state and history. An agent's decision-making process is then based, at least in part, on the set of all internal states of the agent. The perception function see for a state-based agent is unchanged, mapping environment states to percepts as before:

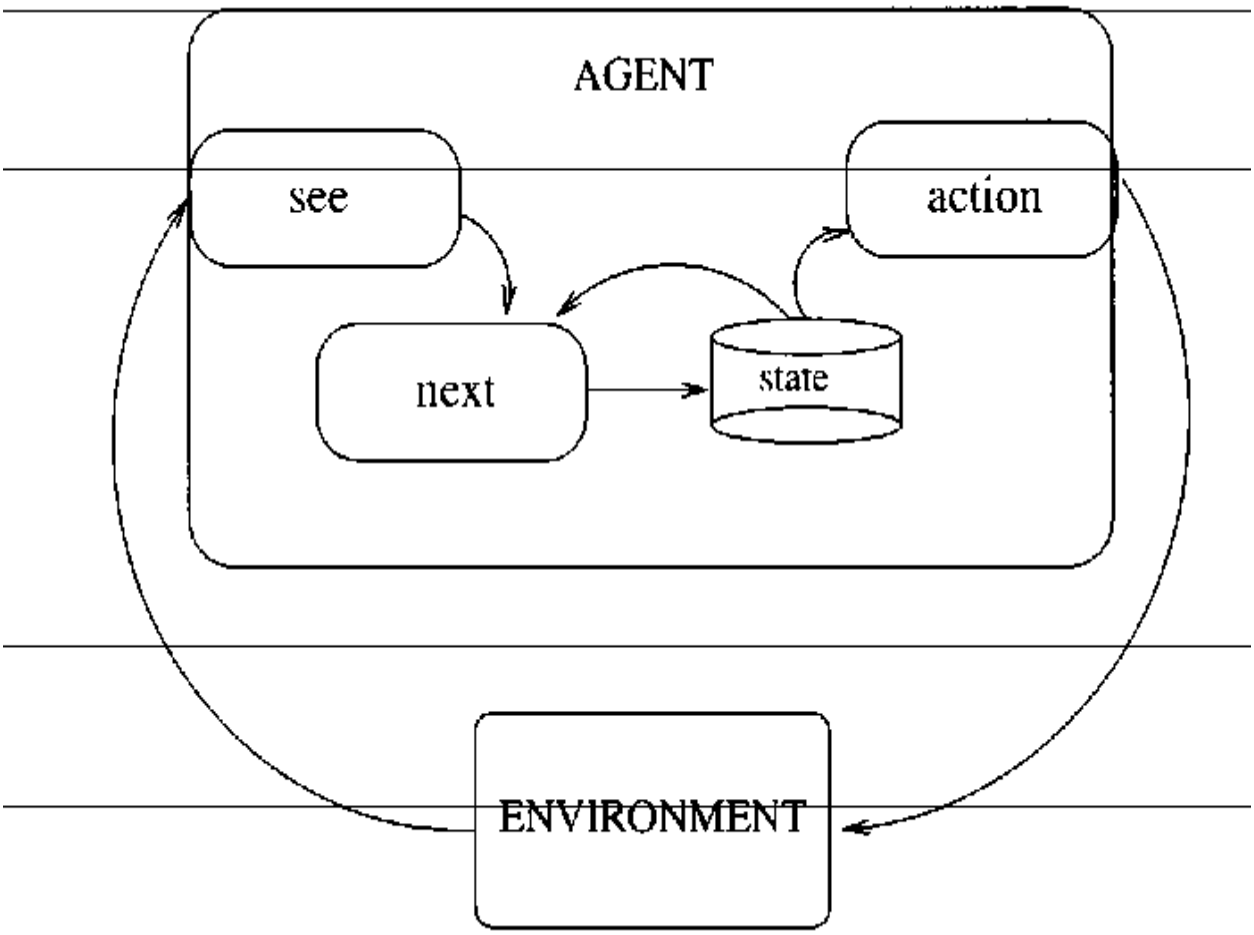


Figure 3 Agents that maintain state

Here the perception function "see" works via the vehicle sensors and these sensors work all the time (except when we already have planned a certain sequence of actions for maneuvering) to check if there are obstacles in the sensor radius. The action-selection function action is defined as a mapping

$$action : I \rightarrow A_c$$

from internal states to actions. Next function maps an internal state and precept to an internal state:

$$next : I \times Per \longrightarrow I$$

2.1.2. The Structure of Agents

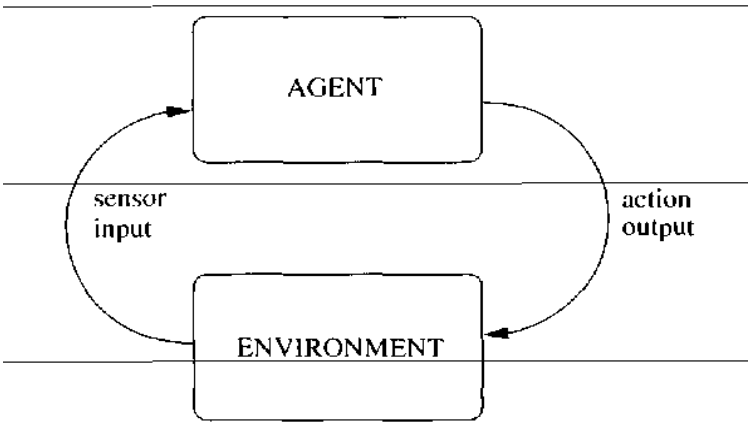


Figure 4 Structure of an agent

Agent programs generally follow the same framework: they take the current precept as input from the sensors and return an action to the actuators. Only the current precept is taken, as nothing more is available from the environment: if the agent's actions need to depend on the entire precept sequence, the agent will have to remember the precepts. This is how our motionAgent works. It keeps track of the precept sequence and then uses it to index it into the table of actions to decide what to do. To build a rational agent in this way, a table that contains the appropriate action for every possible precept sequence must be constructed.

To make a table-driven-agent work, we need to write programs that to the extent possible, produce rational behavior from a smallish program rather than from a vast table.

For example: In the agent notion we are going to use, the precept sequence is planned and stored by the plan_maneuver method. This method plans a certain precept of actions and stores it as a sequence of waypoints for the agents in future.

2.2. Background Knowledge of How Agent Drive Works:

2.2.1. Interaction of Agent Drive and OpenDS

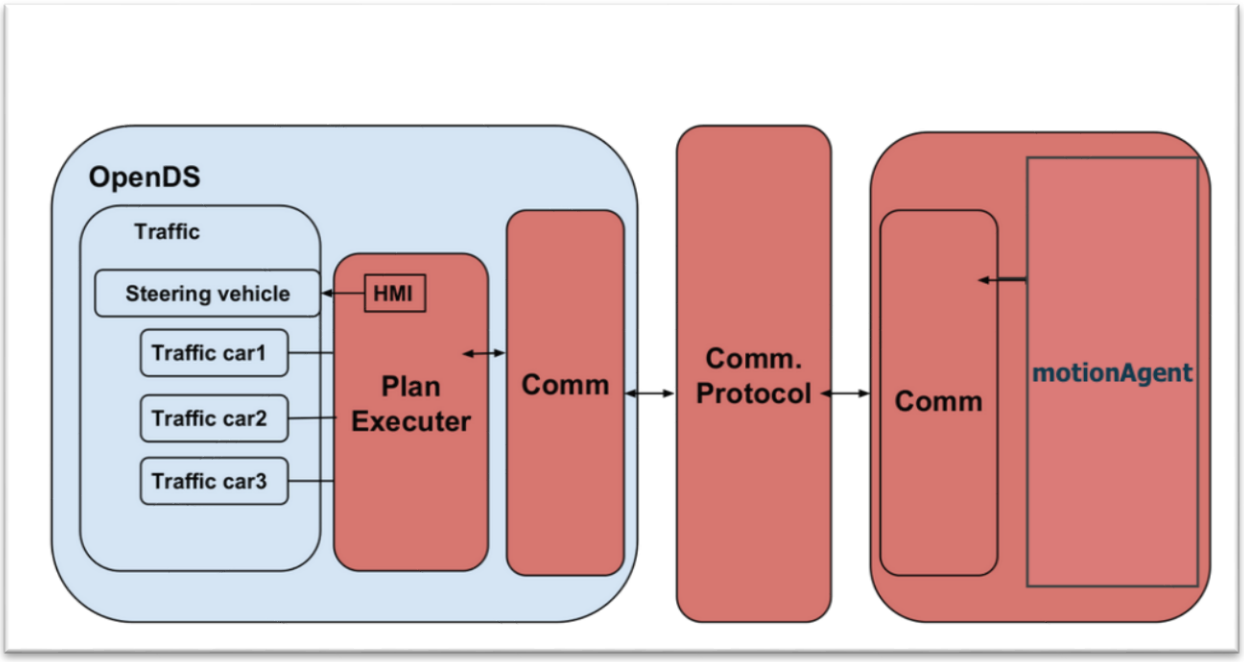


Figure 5 AgentDrive and OpenDS

Open DS is a open source driving simulator with realistic vehicle, engine, and environment physics(OpenGL/jMonkeyEngine) , has ability to connect to steering controls in a real car, has Interface for exchanging data and instructions with other applications ad also allows creation of both polished static 3D environments as well as dynamic driving tasks. So the compatibility of AgentDrive with OpenDS can be established via a communication protocol, in this case sending plans via plain text messages. So if we start from right to left in the diagram shown above; the implementation can be easily observed. The motionAgent comes up with a plan in the form of WayPoints and further sends it to the communicator in AgentDrive. This communicator sends the plan in the form of plain text messages to OpenDS via the communication protocol. OpenDS on receiving the plans, sends it to plan executer which in response steers the traffic cars corresponding to the plan received for the respective Car ID.

A more generalized view of interaction between the them:

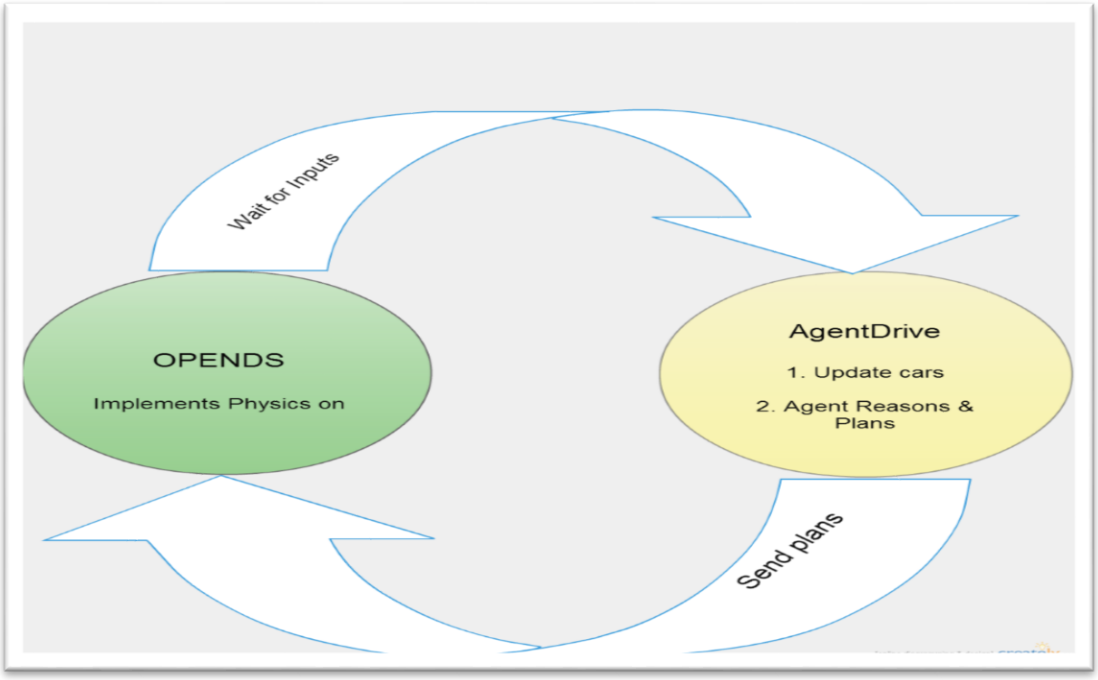


Figure 6 Agent drive interacting with OpenDS

Plans:

- *Speeds at which vehicles should move.*
- *The timestamp for each agent.*
- *Waypoints for the vehicles.*

Inputs:

- *The plans mentioned above serve as an input for the OpenDS.*
- *The communicator in OpenDS waits to receive plan from AgentDrive in the form of waypoints.*

2.2.2. Architecture:

The agents work on the architecture shown below. The highest layer is the maneuver layer which defines the working of the agent with different possible instantiations of the available maneuvers from the maneuver library. The different possible maneuvers that can be instantiated from maneuver library are explained below:

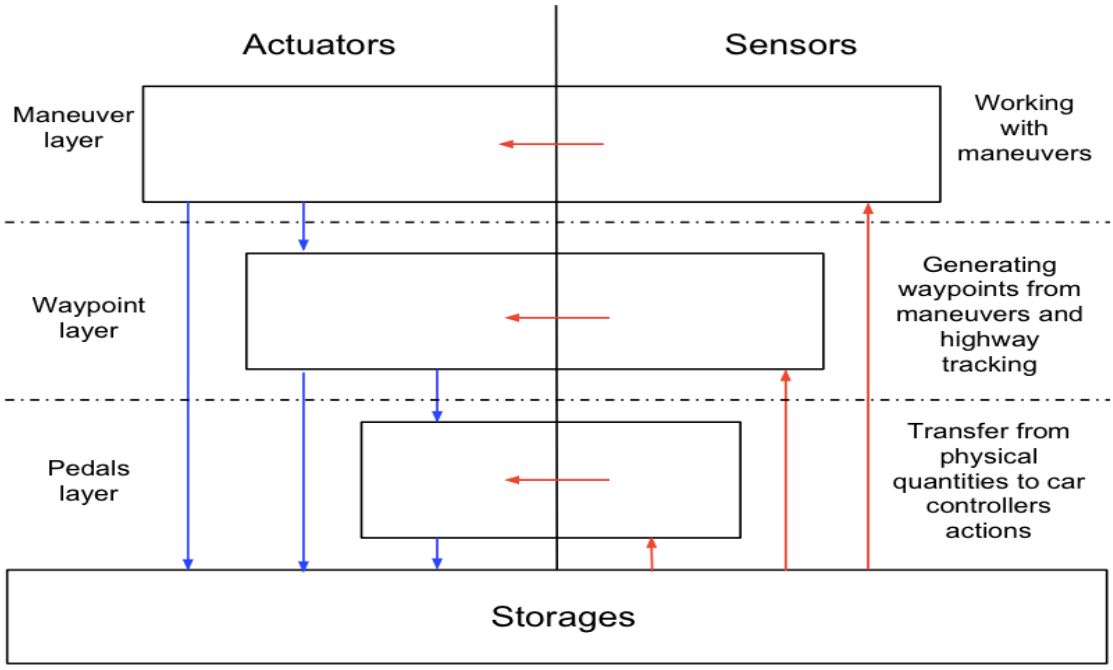


Figure 7 Three layer structure of Actuators and Sensors

Straight maneuver

The vehicle does not change it's lane and continues to move at constant speed without any acceleration or deceleration.

Turning left maneuver

The lane of the vehicle is changed by one to the left. The velocity remains constant.

Turning right maneuver

The lane of the vehicle is changed by one to the right. The velocity remains constant.

Acceleration maneuver

The lane of the vehicle is not changed. The acceleration is constant until the maneuver ends or the maximal velocity of the vehicle is reached.

Deceleration maneuver

The lane of the vehicle is not changed. The deceleration is constant until the maneuver ends or the vehicle stops.

The second layer is the Waypoint layer. In this we can specify waypoints directly for each individual agent to guide them towards their next step. We can say that the Waypoint layer is the abstraction layer for maneuver layer. Maneuver layer completely depends on Waypoint layer to implement the possible instances in maneuver library. We can say that maneuver layer is a short extension to Waypoint layer in which we apply Newtonian physics in the maneuver methods to find the next Waypoint for an agent and thus instead of specifying the Waypoints manually at each cycle, the instantiated library does this itself by repeatedly calling the required method at every agent cycle.

The third layer is the Pedal layer which controls the steering actions required for an agent to reach the specified waypoint or the instantiated maneuver library.

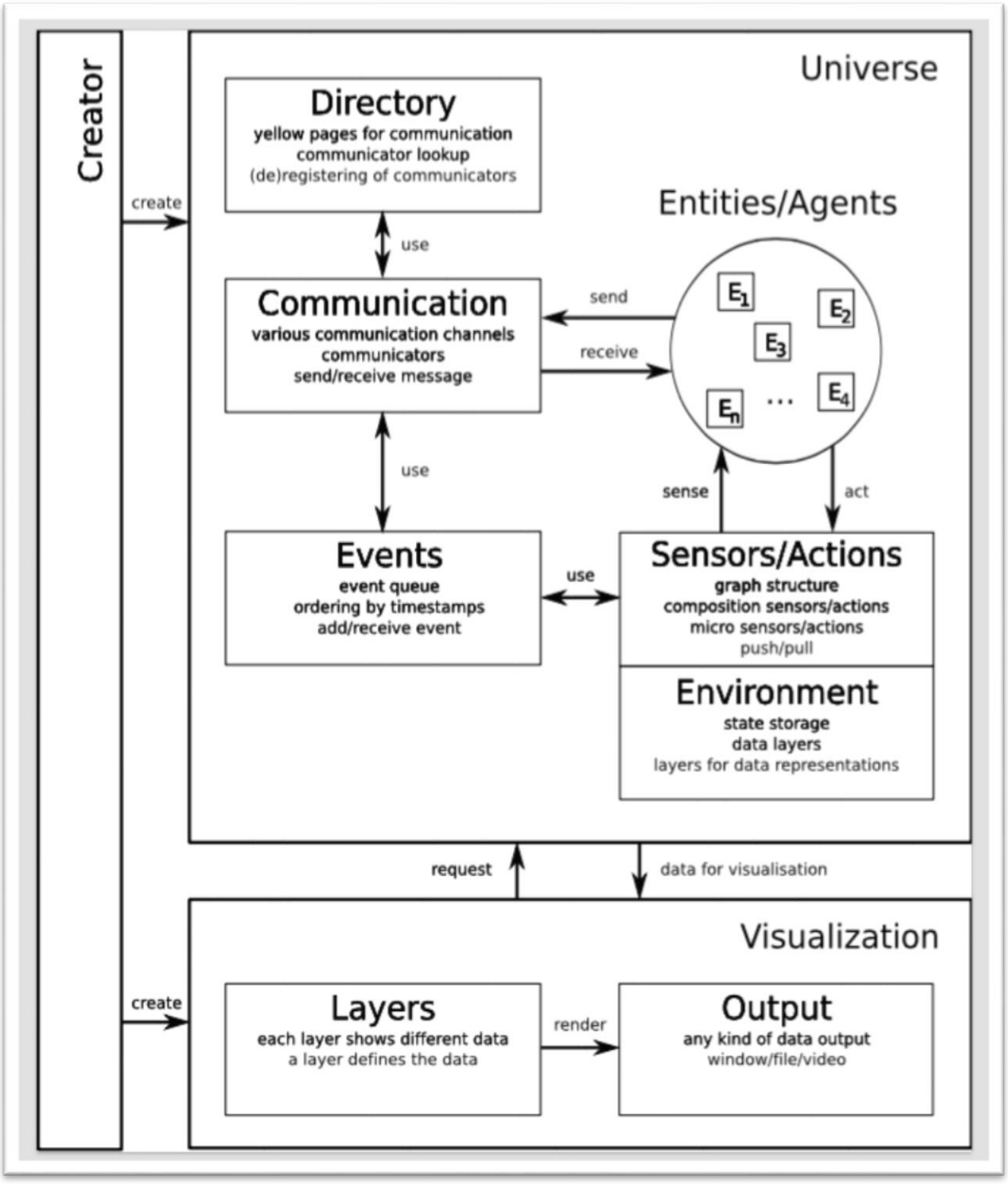


Figure 8 High level Design for AgentDrive

The diagram shown above shows the internal framework on which AgentDrive and OpenDS works. It is also explained below:

- *Creator*: Responsible for Initialization of the environment, entities, visualization etc.

- *Configuration*: For initializing of the application

- Event Queue: An event loop in which new events can be added and removed. The newly added event is added at the end after processing.

- *Environment*: Consists of sensors, Actuators, Storage (Defines Data structure for holding position, velocity etc.)

- *Simulation*: The core of simulation is the event queue. Prioritized on time, an event is created and processed. Additionally, simulation queue provides us the facility to pause n stop. Each element of the simulation can use the events as it wants.

- *Visualization*: Vis - lite 2D visualizer using Java2D backend of Google Earth connector, vis3d - 3D visualizer using Java Monkey Engine as a backend

- *Communication*: Sending plans as messages Traffic Simulator + Driving Simulator (OpenDS)

- Integrated Structured Road Network Data with realistic Physics and Visualization.

- Integrated Human in the loop test. Allows for Car control techniques, cooperative car coordination and user-driven vehicles.

3. Literature Survey

Modeling of the driver behavior is studied for many years and the first concept was published in 1950 by Reuschel [6]. This model was a car-following model , i.e., the model that addresses only management of distances between cars using acceleration and breaks. At least two approaches can be taken to the study of the driver-vehicle car-following function. One is to use physical models, e.g. Newtonian-forcemass-acceleration models (Herman, 1961). A second approach, and the one taken in this paper, is curve-fitting from empirical data in order to describe the driver-vehicle system in terms of inter-vehicular dynamics. The car-following model was adopted by many researchers, and therefore, this concept was widely studied. A representative model is the GM model, which was proposed by researchers at the GM Research Laboratories. Although this model had been one of the most popular models, another types of car-following models were proposed as: spacing model [7], Ohio model [8], psychophysical model [9], and others (explained below).

General Motors (GM) model: The basic philosophy of car following model is from Newtonian mechanics, where the acceleration may be regarded as the response of a matter to the stimulus it receives in the form of the force it receives from the interaction with other particles in the system. Hence, the basic philosophy of car-following theories can be summarized by the following equation

$$[Response]_n \propto [Stimulus]_n$$

for the nth vehicle (n=1, 2, ...). Each driver can respond to the surrounding traffic conditions only by accelerating or decelerating the vehicle. As mentioned earlier, different theories on car-following have arisen because of the difference in views regarding the nature of the stimulus. The stimulus may be composed of the speed of the vehicle, relative speeds, distance headway etc, and hence, it is not a single variable, but a function and can be represented as,

$$a_n^t = \mathcal{f}_{sti}\left(v_n, \Delta x_n, \Delta v_n\right)$$

where, $\oint_{sti}$ is the stimulus function that depends on the speed of the current vehicle, relative position and speed with the front vehicle.

The car following model proposed by General motors is based on follow-the leader concept. This is based on two assumptions; (a) higher the speed of the vehicle, higher will be the spacing between the vehicles and (b) to avoid collision, driver must maintain a safe distance with the vehicle ahead. The acceleration of the follower vehicle depends upon the relative velocity of the leader and the follower vehicle, sensitivity coefficient and the gap between the vehicles.

Spacing Model: In some cases, an empirical distribution of the locations of the sites within the sequences may be known. Using such a distribution will allow the algorithm to focus on the most probable locations for sites. To specify a spacing distribution, enter a value for each possible position in the sequences starting at position 1 up to the maximum sequence length. Spaces, tabs and newlines are ignored. The distribution will be normalized, so it is not necessary that the values sum to 1. This distribution will apply to all sequences. The default is a uniform distribution (each position is equally likely a priori). The spacing model only applies when the recursive sampler is used.

Psycho-Physical model: Psycho-physical vehicle-following models determine reactions of vehicles in traffic processes with regard to changing the acceleration of vehicles. Reactions happen if differences in speed and distance to the predecessor occur

A general acceleration model was proposed by Gipps [10]. This model was designed for carfollowing and free-flow conditions. Gipps also published [11] the first lane change decision model. Another important model from the family of driver modeling is the gap acceptance model. This model decides whether a gap is suitable for lane change. Besides, an integrated driving modeling framework was proposed by Tomer [12].

Early research on Inter vehicle communications (IVC) began in the 1990s, inspired by research in the area of intelligent transportation systems (ITS) initiated by the Department of Transportation (DOT) in the U.S. and by the PROMETHEUS project of the EUREKA program

funded by the European Commission. With the decreasing cost of components for communication and positioning [e.g., global positioning systems (GPS)] in the recent past, IVC became more attractive. Various research projects were initiated [13 -15], some explicitly focused on IVC, others considering IVC as one of many possibilities for data distribution. Increasing interest in roadside-to-vehicle communications and IVC also led to various standardization efforts worldwide, e.g., for a suitable air interface.

There has been many efforts to study agent based simulation and optimization of Urban transit systems. The idea of demand and supply model, where passengers are considered to be in demand side and the operators are in supply side, to minimize the travelling cost and also at the same time maintain the desired level of service along with minimized system cost was proposed by Yang and Bell [16 - 18] as a bilevel model. In a bilevel model [19], the upper level aims to minimize system operation costs by finding the optimal settings (such as the service frequency) given the equilibrium passenger flow and the lower level aims to get the equilibrium passenger flow with the consideration of passengers' route choice behavior under certain network design. These two levels interact with each other.

One of the most comprehensive works is the cooperative collision warning systems (CCW) developed by the PATH project. PATH has already implemented CCW in a real test system [20]. The prototype of CCW has provided functionalities of a variety of safety services, such as forward collision-warning, blind-spot, and lane-change situation awareness. The authors also studied the accuracy requirements for communication in such CCW system. However, this study relies too much on infrastructure and accessories. To achieve a precise localization, DGPS is used as the source of position estimation. The precision is then enhanced by integrating the measurements of Vehicular Sensors and matching estimated positions with digital map information. The PATH CCW systems employ differential Global Positioning System (GPS), Controller Area Network (CAN), and digital maps to provide precise motion state, but such device or infrastructure requirements may not be universally available at the current time.

Consensus problems have a long history in computer science and form the foundation of the field of distributed computing [21]. Formal study of consensus problems in groups of experts

originated in management science and statistics in 1960s (see DeGroot [22] and references therein). The ideas of statistical consensus theory by DeGroot reappeared two decades later in aggregation of information with uncertainty obtained from multiple sensors1 [23] and medical experts [24]. Distributed computation over networks has a tradition in systems and control theory starting with the pioneering work of Borkar and Varaiya [25] and Tsitsiklis [26], Bertsekas, and Athans [27] on asynchronous asymptotic agreement problem for distributed decision making systems and parallel computing [28].

There had been several attempts [29] to demonstrate several automated models in the following fields:

- Vision-based road following
- Following magnetic nails
- Following a radar reflective strip
- Radar-based headway maintenance
- Ladar-based headway maintenance
- Evolutionary systems
- Close vehicle following (platooning) Cooperative maneuvering
- Obstacle detection and avoidance
- Mixed automated and manual driving
- Mixed automated cars and buses
- Semi-automated maintenance

The 1997AHS Demo had seven demonstration scenarios, designed to showcase different technologies and different functions:

- Platoons, with closely-spaced vehicles following buried magnets
- Free agents, with cars and busses using vision and radar
- Evolutionary, showing how this technology can be introduced incrementally for driver assistance
- Control transition, using both vision and buried magnets
- Alternative technology, using a radar- reflective strip for lateral control

- Infrastructure diagnostic, checking the accuracy of the magnets
- Heavy trucking, using radars for smart cruise control and driver warning

The scholarly articles in the field of co-operative driving build up quite a long list and thus opening up a wide spectrum of ideas to researchers across the globe. Several transport companies and automotive industries have shown their interest in co-operating and partnering up with the university researchers to get assistance in analyzing the various driver assistance simulation models before final installation.

It clearly shows that a lot has to be done yet to optimize the existing algorithms or we need to come up with new possibilities in this domain.

4. Design and Implementation of Overtaking Maneuvers in Agent Drive

4.1. Mathematical modeling of algorithms

The algorithm for path-planning need to be modeled in a proper way so that we come with a good prediction and planning algorithm and further plan to have a realistic outlook of simulation scenario. So, this section is divided in two parts: Prediction and Planning algorithm and No-linear lane changing. They are further explained below:

4.1.1. Prediction and planning algorithm

To predict a successful maneuvering several parameters need to be computed based on the current state of the agent.

Let v be the maximum allowed speed of the agent.
Let u be the current speed of the agent.
Let acc be the acceleration of the agent.

A successful planning is made in following few steps:

Finding the time to lane change (t)

$$Length\ of\ non-linear\ path\ (s) = \int_{-a}^{a} \sqrt{1 + \frac{dy}{dx}^2}$$

$$Where\ a = g(0.1), where \left\{ g(x) = \frac{v}{2 * \log \frac{x}{lanegap - x}} \right\}$$

$$and, y = \frac{lanegap}{1 + e^{\frac{-2x}{v}}}$$

$$s_1 = \frac{v^2 - u^2}{2 * acc}$$

$If\ s - s_1 > 0$	$If\ s - s_1 = 0$	$If\ s - s_1 < 0$
$t_1 = \frac{v - u}{acc}$ $t_2 = \frac{s - s_1}{v}$ $t = t_1 + t_2$	$t = \frac{v - u}{acc}$	$t = \frac{-u \pm \sqrt{u^2 - 4 * \left(\frac{acc}{2}\right) * (-s)}}{2 * (\frac{acc}{2})}$

Finding the approximate position of the neighbors after the predicted time to lane change taking in account the worst-case scenario in which the neighbors travel with maximum speed.

$$Rear\ left\ after\ lane\ change\ duration\ (rl') = Current\ position\ of\ rear\ left - t * Maximum\ speed\ of\ Rear\ left$$

$$Rear\ front\ after\ lane\ change\ duration\ (rf') = Current\ position\ of\ rear\ front - t * Maximum\ speed\ of\ Rear\ front$$

$$Front\ after\ lane\ change\ duration\ (f') = Current\ position\ of\ front - t * Maximum\ speed\ of\ front$$

$$Current\ vehicle\ after\ lane\ change\ duration\ (c') = Current\ position\ of\ the\ vehicle - t * Maximum\ speed\ of\ current\ vehicle$$

Determining the maximum allowed overtake speed of the agent

Time taken to collide with front $left$ vehicle$(t_3) =$

$$\frac{Relative\ distance\ between\ \ Current\ and\ front\ left\ after\ lane\ change\ duration}{Relative\ speed\ between\ Current\ and\ front\ left\ after\ lane\ change\ duration} =$$

$$\frac{c' - fl'}{Overtake_{speed} - Max_{Speed} of\ front\ left}$$

Time taken to collide with front vehicle(t_4)

$$= \frac{Relative\ distance\ between\ \ Current\ and\ front\ after\ lane\ change\ duration}{Relative\ speed\ between\ Current\ and\ front\ after\ lane\ change\ duration}$$

$$= \frac{c' - f'}{Overtake_{speed} - Max_{Speed} of\ front}$$

$Equating\ t_3\ and\ t_4\ to\ find\ the\ maximum\ allowed\ overtake\ speed\ of\ the\ agent$

$$If\ Overtake\ speed > Maximum\ allowed\ speed$$

$$Overtake\ Speed = Maximum\ Speed$$

$Else\ Continue$

4.1.2. Non-Linear Lane Changing

The figure shown below pictorially represents how we actually change lane during a maneuver. The shown path mathematically maps to a special type of function called “Sigmoid Function”.

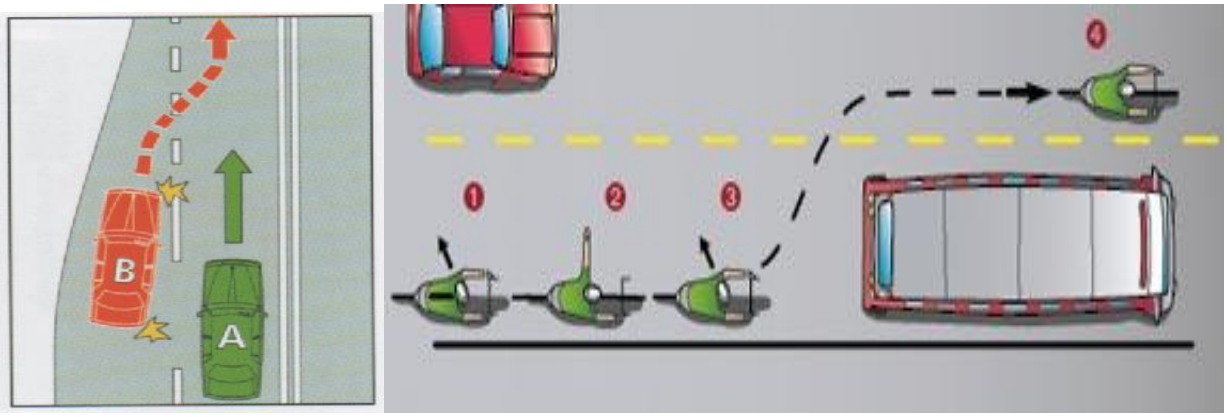


Figure 9 General Overtake Scenario

Sigmoid functions can represent the realistic lane changing in automated vehicles. To make it possible, mathematical functions can help to parameterize the stretch in and stretch out of the curves depending on the current speed of the vehicle. A vehicle with higher speed needs more time to change lanes whereas if an agent plans to travel at lower speed then it occupies less space to perform lane change.

The above graph explains the fact that stretching in and stretching out of functions can be controlled by adding an external factor, in this case speed of the agent. A generalized equation of the form.

$$y = \frac{1}{1 + e^{-x}}$$

is transformed to

$$y = lanegap * \frac{1}{1 + e^{\frac{-x}{v}}}$$

The factor lane gap restricts the range of the function from $\{0\ to\ lanegap\}$, and the factor Agent Speed Stretches In the function when of lesser value and Stretches Out the function when of higher value.

4.2. Constraints and Tradeoffs

The goal of the project is to come up with a solution to let the agents drive at maximum speed and dodge collisions. Other constraints and tradeoffs are summarized below:

- Following traffic rules for maneuver planning and maximizing overall speed. In this case we have followed European driving conventions (Left wheel drive).

- Minimizing trip time: For this the average speed of the journey should be very close to the maximum possible speed of the vehicle. Following traffic rules and minimizing trip time can interfere and therefore following traffic rules is prioritized before shortening duration of trip.

- Finding a plan to execute maneuvers in the agent life cycle: It is necessary for an agent to calculate the next Waypoint for driving within the agent simulation cycle. Otherwise the agent may miss the next Waypoint and further can circumambulate around the previous Waypoint. It is highly necessary to avoid any unending decision making control statements.

- Limitation of executing the stored plans within a range of time-frame and not at the exact time frame: On finding a certain sequence of maneuvers, a cross check is required to detect the limiting convergence of predicted Waypoint to the current position. The limit has to be considered every time while providing the first waypoint and then further providing the another one after checking the checkout at previous checkpoint.

- Approximation in finding Asymptotic convergence to S-Shaped maneuvering: The limiting convergence applies even to the sigmoid left or right lane maneuvers. The vehicles actually never reach the specified Waypoint rather they try to converge to the specified position asymptotically but actually never reaches it. So, a limiting parameter is again required in case of lane change maneuvers.

- Exponential back-off planning to prevent agents from executing plans at the same point of time. It is quite likely that agents may have planned maneuvering in conflicting spaces. Then First come first served policy holds and others have to wait until the maneuver completes. The worst case scenario happens when maneuver lasts for miles and others wait for minutes for their turn to execute maneuver.

- Critical section overlap delays execution of plans: If an agent finds it's overtaking plan in the region already reserved by some other agent executing its overtaking process; then agent has to wait until the reserved region say Critical section is set free by the owner. If the overtaking is planned for another few minutes, then all the other conflicting agents will have to wait and platoon the vehicle ahead of it. This is a big trade-off between driving safely and driving at high speeds.

- The maneuver is executable on the highway, i.e. no driving out of the highway.

4.3. Challenges

Programming autonomous agents in an integrated platform; OpenDS and AgentDrive; require specific and at some instances, matching configuration parameters at both the ends. These parameters include sensor radius, safe distance, vehicle id, acceleration and deceleration of the vehicle and so on. To generalize the aspect of finding a path to drive at maximum speed; standard libraries with one of the possible maneuvers like Straight Maneuver, Acceleration maneuver and so on. Mathematical calculations based on surrounding traffic decide the future plans for an agent. Further at every cycle, the challenge is to update the beliefs and re-transmit the existing plan or define a new plan for the agent. Further it may be possible that while the agents queue up in a lane because of obstructions ahead, the planning zone of maneuvers find a conflict with each other and then broadcast the plans at the same time. This may end up with no solution as no one gets the authority to decide who goes when. One solution to this problem can be including a delay parameter every time an agent goes in the planning loop. To give a realistic touch to Change lane maneuvering, mathematical function like sigmoid method can help the

agent to find the waypoint corresponding to the defined function. The challenges can be summarized as follows:

- Modeling traffic simulation with realistic parameters
- Path-planning and defining a set of standard libraries
- Re-model the given scenario automatically on updates in beliefs and plans of each agent
- Communicating a common map model
- Performing automated maneuvers with real-time constraints.
- Maneuver representation in the library
- General algorithms of maneuver instantiation/evaluation
- Assigning waypoints corresponding to maximum speed, as vectors in AgentDrive give negative velocity and their length gave positive velocity
- Avoiding multiple conflicting planning by multiple agents at a time
- Implementing S-shaped lane change maneuver to show a realistic model.

4.4. Initial Phase of Work: With Java Applets Simulation

Before getting the feel of realistic simulation environment, I was working on a JAVA compiler, to develop everything from scratch using the concept of JAVA Applets and run a general simulation traffic environment. A more general definition of Applets on Wikipedia is "A Java applet is a special kind of Java program that a browser enabled with Java technology can download from the Internet and run."

4.4.1. Architecture Specifications

The basic architecture of the applet simulation to represent automated maneuver follows the basic layout of an applet life cycle as shown above. The parameters are initialized in the init method and further a new thread defined in start method starts the applet. To control the flow of simulation, threads are filtered by their names. Further, each thread loops continuously within it's boundary to achieve the following goals:

- To update different sections of the program simultaneously, such as updating x and y co-ordinate for all agents together.

- To handle manual instructions by feeding in the inputs to the thread.

- To Maintain a universal timer to synchronize agents.

- To keep the simulation running until we manually shut down the applet window

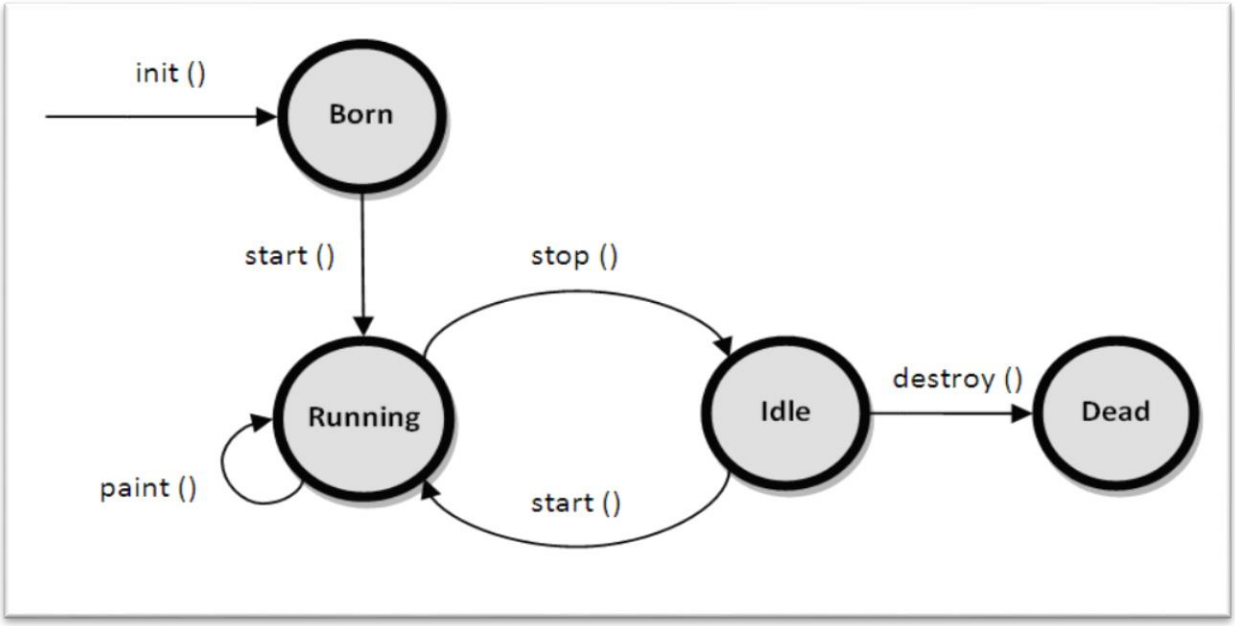


Figure 10 Java Applets life cycle

4.4.2. Algorithms Used:

- *Exponential Distribution of vehicles*: In this the time between events in a Poisson process [4], i.e. a process in which events occur continuously and independently at a constant average rate, is referred.

- *Nagel–Schreckenberg model*: In each step, the following four actions are conducted in order from first to last, and all are applied to all cars [5]. In each action the updates are applied to all cars in parallel.

- *Acceleration*: All cars not at the maximum velocity have their velocity increased by one unit. For example, if the velocity is 4 it is increased to 5.

- *Slowing down*: All cars are checked to see if the distance between it and the car in front (in units of cells) is smaller than its current velocity (which has units of cells per time step). If the distance is smaller than the velocity, the velocity is reduced to the number of empty cells in front of the car – to avoid a collision. For example, if the velocity of a car is now 5, but there are only 3 free cells in front of it, with the fourth cell occupied by another car, the car velocity is reduced to 3.

- *Randomization*: The speed of all cars that have a velocity of at least 1, is now reduced by one unit with a probability of p. For example, if p = 0.5, then if the velocity is 4, it is reduced to 3 50% of the time.

- *Car motion*: Finally, all cars are moved forward the number of cells equal to their velocity. For example, if the velocity is 3, the car moves 3 cells.

4.4.3. High-Level Design

This section refers to work flows and data flows between component systems. It gives the overall System Design in terms of Functional Architecture and Database design. This is very useful for the developers to understand the flow of the system.

The high level design is explained below**:**

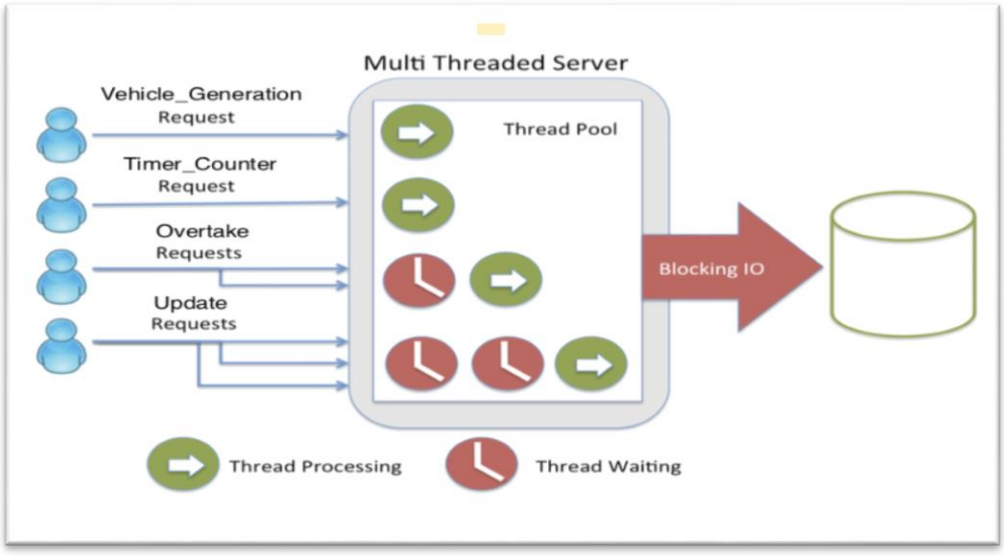


Figure 11 High level design for Java Applets

- Update_requests: Each agent (vehicle) is autonomous and therefore responsible for manuevers and lane change during its life-cycle. So, a thread associated with each vehicle controls its forward motion, backward motion, lane change and overtaking. The update request is processed at discrete time steps. This cycle of each thread updates the Y co-ordinate of the vehicle associated with it. To examine the effectiveness of the simulation, only a small segment of a road is simulated for now. So, once the agent moves out of the road, it is dereferenced and subsequently collected in a queue. This garbage queue helps in collecting trashes and there-by making the whole process memory efficient. How I have planned to re-use the trash will be explained in next section. The agent keeps a check on preceding and successive vehicle and then after plans to update its speed accordingly. The agent follows Nagel-Schreckenberg driving model to achieve collision-free environment and thus speeds up or down the agent based on the collected parameters.

- Vehicle_generation Requests: As soon as a vehicle steps out of the lane it enters a queue, which captures its allocation slot. This index can further be used to initialize a new vehicle with different set of parameters like length of vehicle, color, speed, lane number, vehicle number, X location, Y location, Maximum speed and Current speed. The basic idea behind such a wide range of simulation parameters is to make the situation resemble to a actual traffic scenario. If queue is empty, vehicles are generated from a default queue containing a maximum of 20 vehicles per lane multiplied to number of lanes we desire. Thus efficiency in terms of space is achieved by maintaining a queue as described above.

  These vehicles, collected either from garbage queue or a fresh one, are generated exponentially based on Poisson distribution with rate of change λ (varying from 0 to 1). So by this model the distribution of waiting times between successive changes (with k=0) is: $1 - e^{-x}$ .At each half a millisecond the counter is incremented and further subsequent checks are made that if the condition satisfies the Poisson equation.

- Overtake_requests: Overtake when required, depending of current traffic scenario or more precisely, the status of succeeding and preceding vehicle, is activated by computing a overtake plan in advance and further instructions plus assistance to overtake are given to maneuver the succeeding vehicle.

  The algorithm to overtake is activated when any of the mentioned conditions are met:

  - Speed of vehicle greater than the succeeding vehicle.
  - Failure of succeeding vehicle.
  - If succeeding vehicle jams the entire lane with a speed reduction factor much considerably.
  - Manual over-rides given by the user.

Snapshots of the Java Applet Simulation:

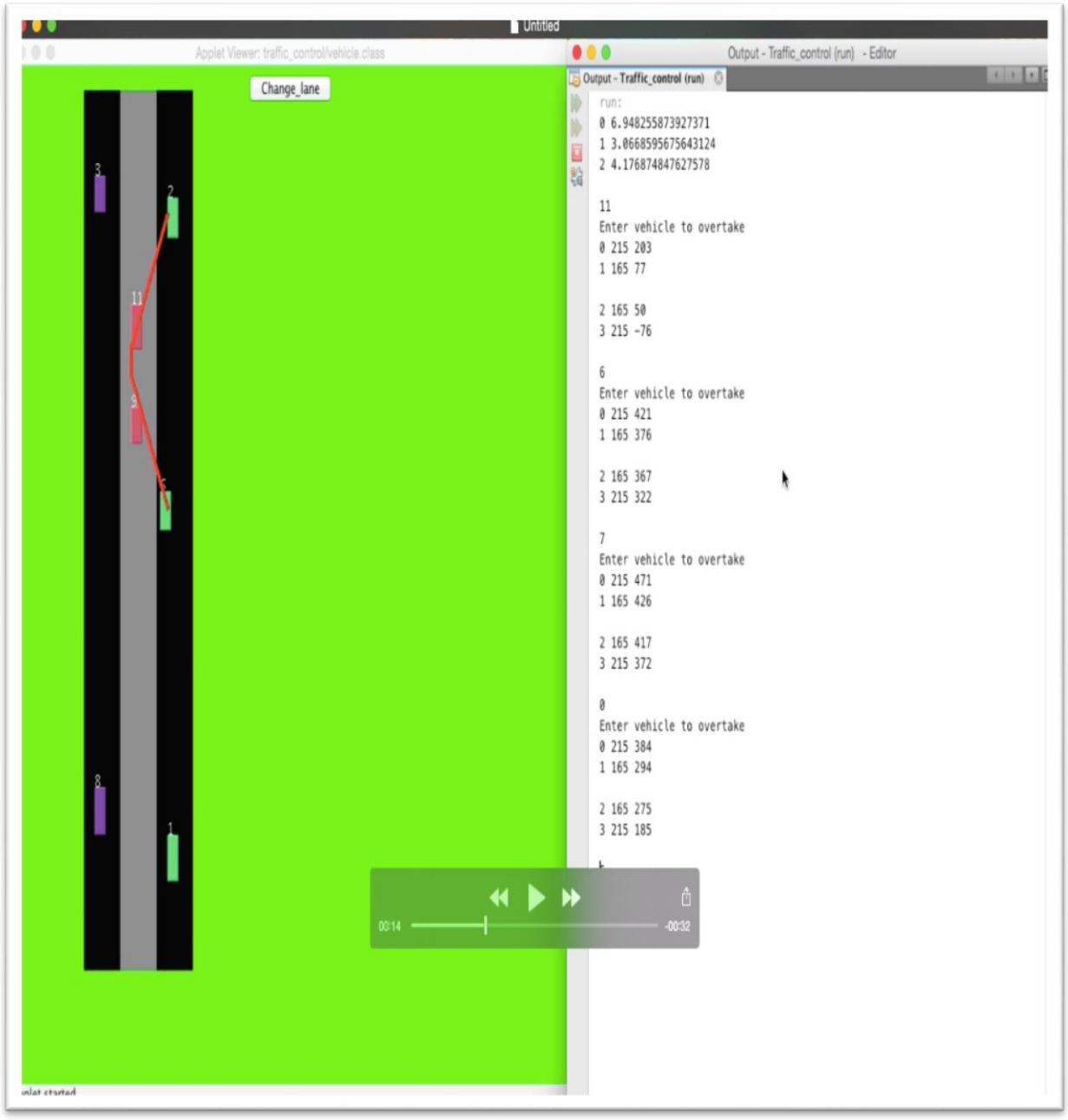
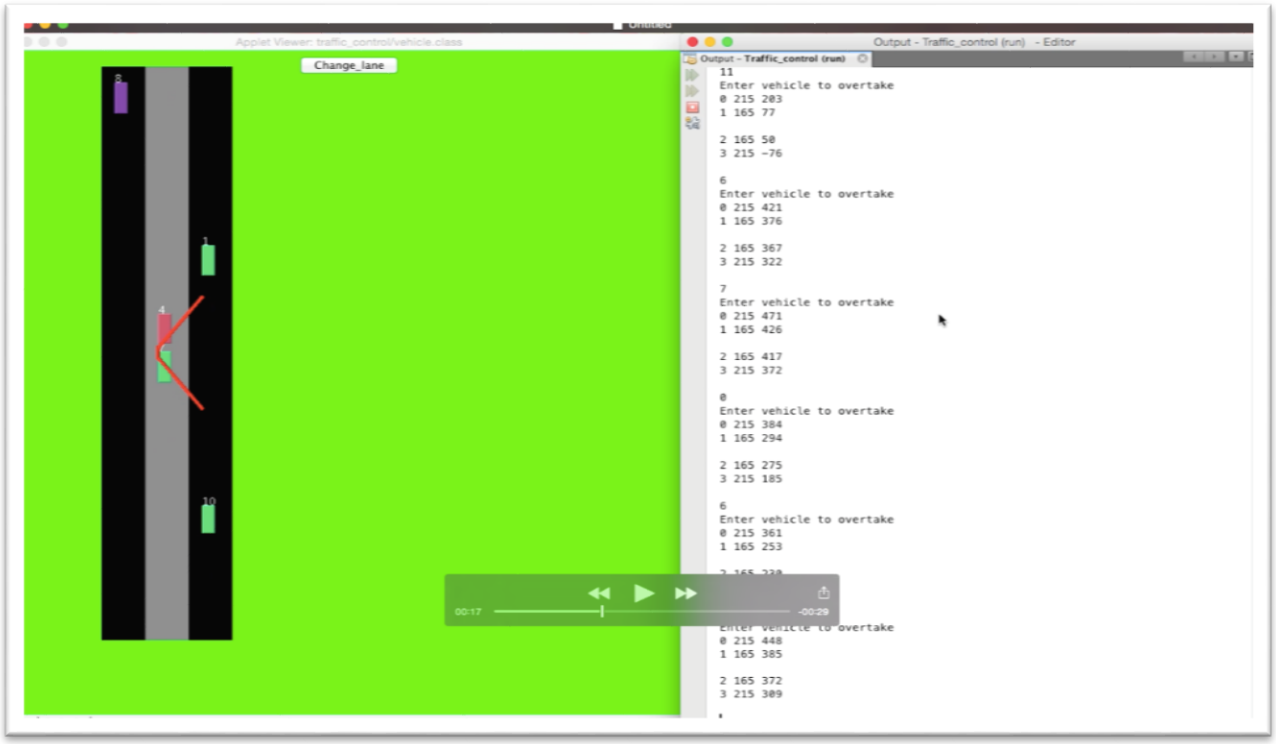


Figure 12 Snapshot 1



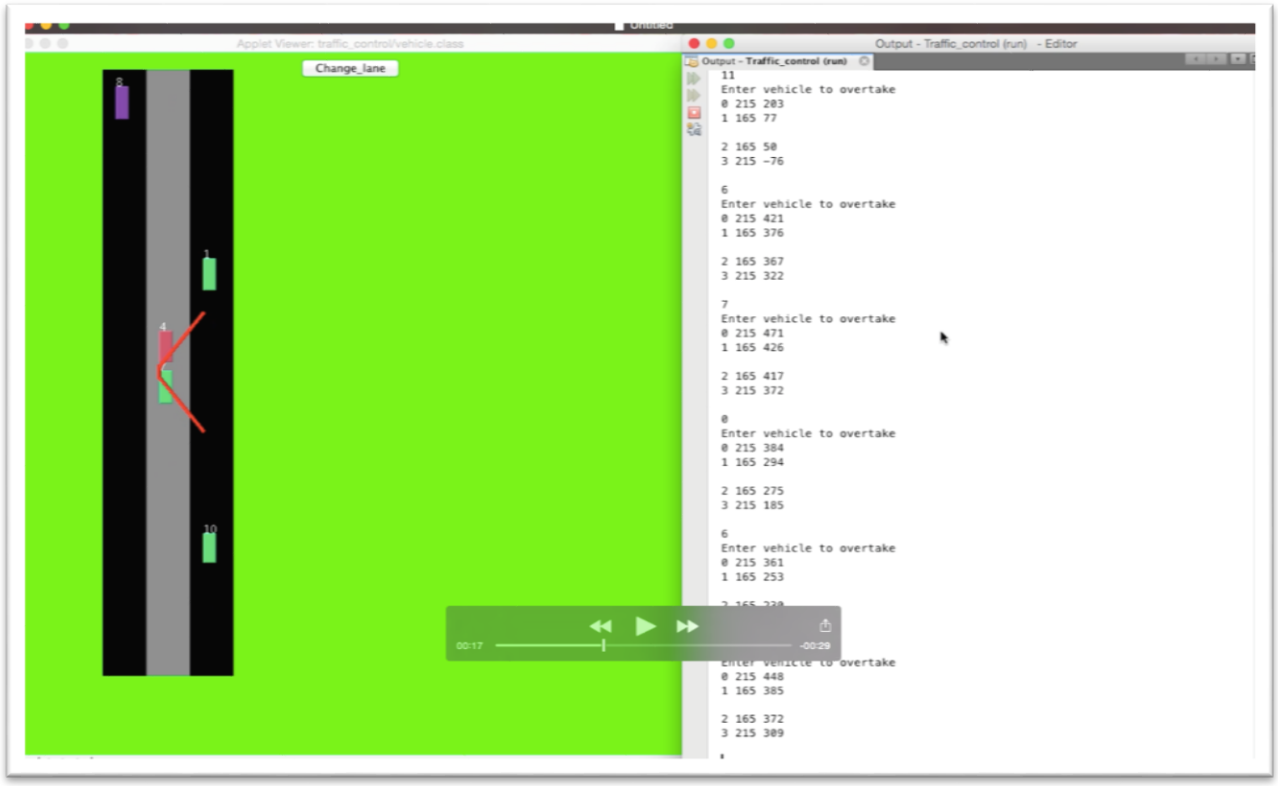


Figure 13 Snapshot 2

4.5. Final Phase Of Work: With Agent Drive

After getting some idea of how to design a basic simulation environment, I started my work with Agent Drive and OpenDS.

4.5.1. Classes and Methods used in Agent Drive

This section will specify the actual logic for each and every component of the system. Class diagrams with all the methods and relation between classes will also be covered under this section.

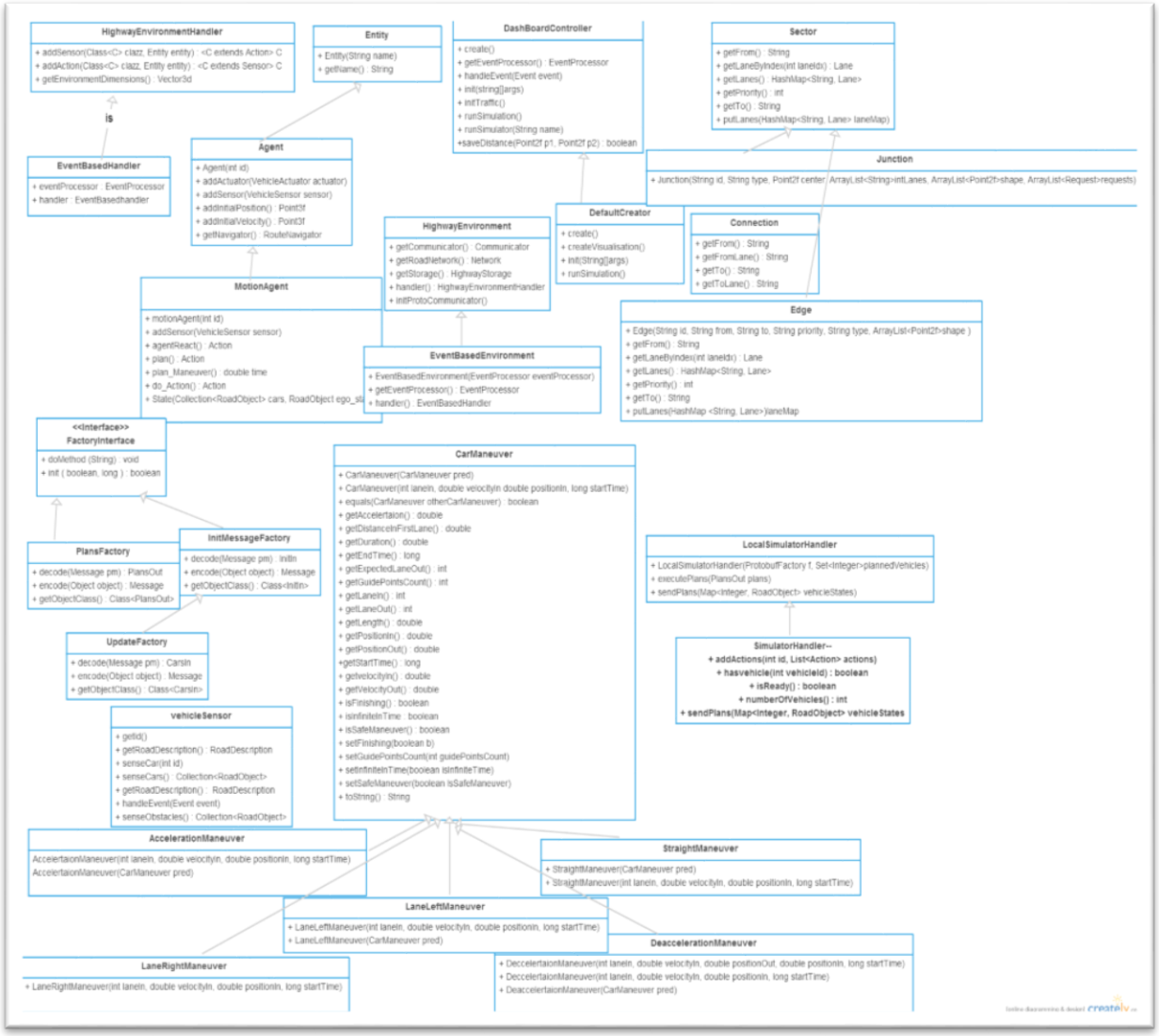


Figure 14 Class Diagram for AgentDrive

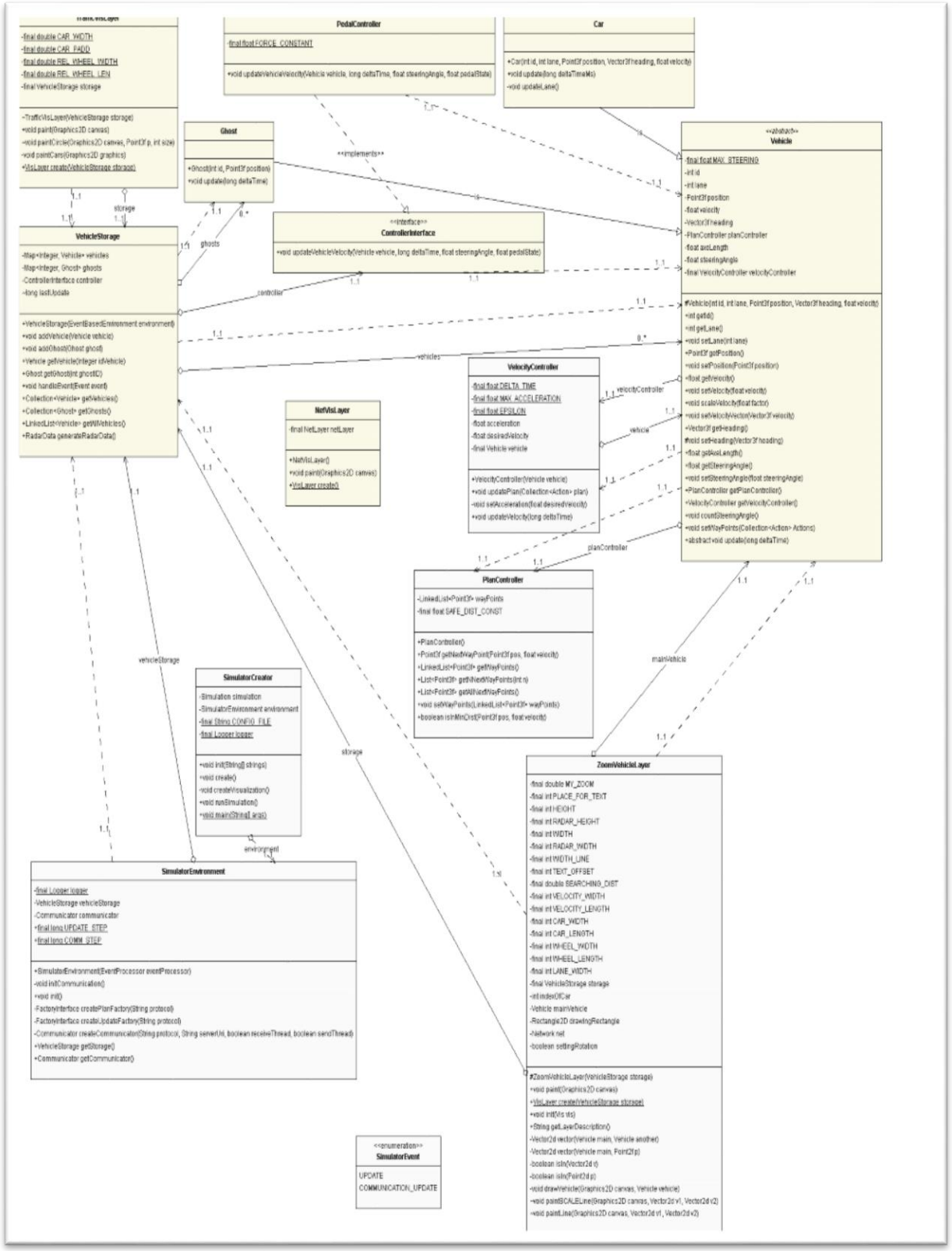


Figure 15 Class diagram for OpenDS

- **Usage of Classes**

1. *Agent Class:*

An agent class defined under the package "cz.agents.highway.agent" is responsible for defining plans for an agent. And we define plans for an agent in terms of Waypoints. We can re-define the way of implementation to meet our expectations from an agent.
An agent class need to extend class "Agent" under same package. This enables the agent to sense Position, Velocity and other parameters of their neighboring vehicles. Moreover, the agents are initialized for each vehicle specified with unique id in a XML file. For example; for the scenario straight-highway the file is located in "SourcePackages/Other Sources/src/main/resources/nets.highway-straight/highway-straight.rou.xml". And as we can see in the directory that there is also a file named "highway-straight.net". This basically initializes a lane with several parameters like id, index, speed, length, shape (specified in the form of a rectangle where the first two parameters are the bottom left co-ordinates and the other two are top-right co-ordinates.

2. *Creator Package:*

This package defines three classes named, DashBoardController, DefaultCreator and Main Class. DashBoardController extends DefaultCreator Class; which is responsible for creating the simulation environment as defined in the file specified in the second run argument of AgentDrive. The configuration file referred in run argument say (test.groovy) of AgentDrive is defined at location "settings/groovy/local/test.groovy". This file saves parameters for simulating the environment and stores several constants like acceleration, deceleration, simulation time step, simulation speed and so on.

a. *DashBoardController*: It manages launching of simulators, and their synchronization with AgentDrive and sending simulator appropriate plans and updates. For this we need to define methods to initialize traffic and even distribution of vehicles, create agent for every single vehicle.

b. *DefaultCreator*: It defines methods to run then simulation, create visualization and register various layers of visualizations (Fps layer, Help Layer), create environment and so on.

3. *Package Highway environment road net:*

This package consists of classes that define connection, edges, junction etc. between the lanes on a road network. For this hash maps are required to map a string to edge or junction. Also , kd-Trees are created and filled with point - lane pairs for fast look up of the cars current lane based on its x, y coordinates only.

4. *Package Highway Maneuver:*

Several classes defining various possible types of maneuvers like straight maneuver, acceleration maneuver, and deceleration maneuver and so on are defined in this package. The maneuvers provide waypoints to the vehicle and the vehicle then reacts to the waypoints given by following them.

So several parameters like acceleration, duration of maneuver, entry lane, and exit lane and so on are required when initializing an instance of any of the possible maneuvers. All the maneuvers extend the class CarManeuver and therefore it would be good to instantiate an object of the required maneuvers and then refer it to by initializing an instance of CarManeuver class. For example:

CarManeuver sm= new StraightManeuver();

As required for the agent, we can later change the maneuvering types for an agent.

5. *Package highway.storage.plan:*

This package acts as a storage for the Actions, ManeuverAction, Plans, WayPointActions etc. Thus storage classes facilitate retrieving the CarId, TimeStamp, Plans, Speed, Position and so on.

- **Main Methods used**

Important Classes:

• *Agent Class*: This class is the super class for any type of agent that plans and provide WayPoints for the vehicles.

• *DashBoardController*: It is responsible for distribution of vehicles and defining initial conditions for a vehicle.

• *ManeuverTranslator*: This class translates maneuvers generated by sub-classes of Agent to waypoints

• *RouteNavigator*: This Class used for car navigation on given route.

• *CarManeuver*: It acts as a super class for several possible types of maneuvering.

• *State*: It stores the carId of the vehicles in the neighborhood of the current vehicles.

• *VehicleSensor*: It provides methods to sense the current car or a collection of cars and further return the position, velocity and other aspects related to the vehicles.

• *Connection, Junction, Edge, and lane*: It defines a structure holding connections through junction loaded from sumo.net.xml file.

• *Edge, Junction, Lane, and Request*: It is basically a structure holding data about an (edge/junction/lane/Request data of a junction) loaded from sumo.net.xml file.

• *Network*: It provides the following data: edges, junctions, lanes, connections, tunnels, bridges. It also provides a converter from x, y coordinates to a specific lane.

Important methods:

• *Agent React*: The method defined in class extended by AgentClass. It is responsible for returning Waypoints using the class WPAction (which extend Action Class).

• *Translate*: Translates a maneuver to waypoints.

• *Plan_maneuver:* This method tries to pre-plan and predict possible co-ordinates on each state change.

• *addSensor*: Adds a sensor to the MotionAgent and further provides the possibility for actuators to act, given a list of actions.

4.5.2. Path Planning with State Changing Architecture

The behavior of the state based agent is described with the help of following diagram:

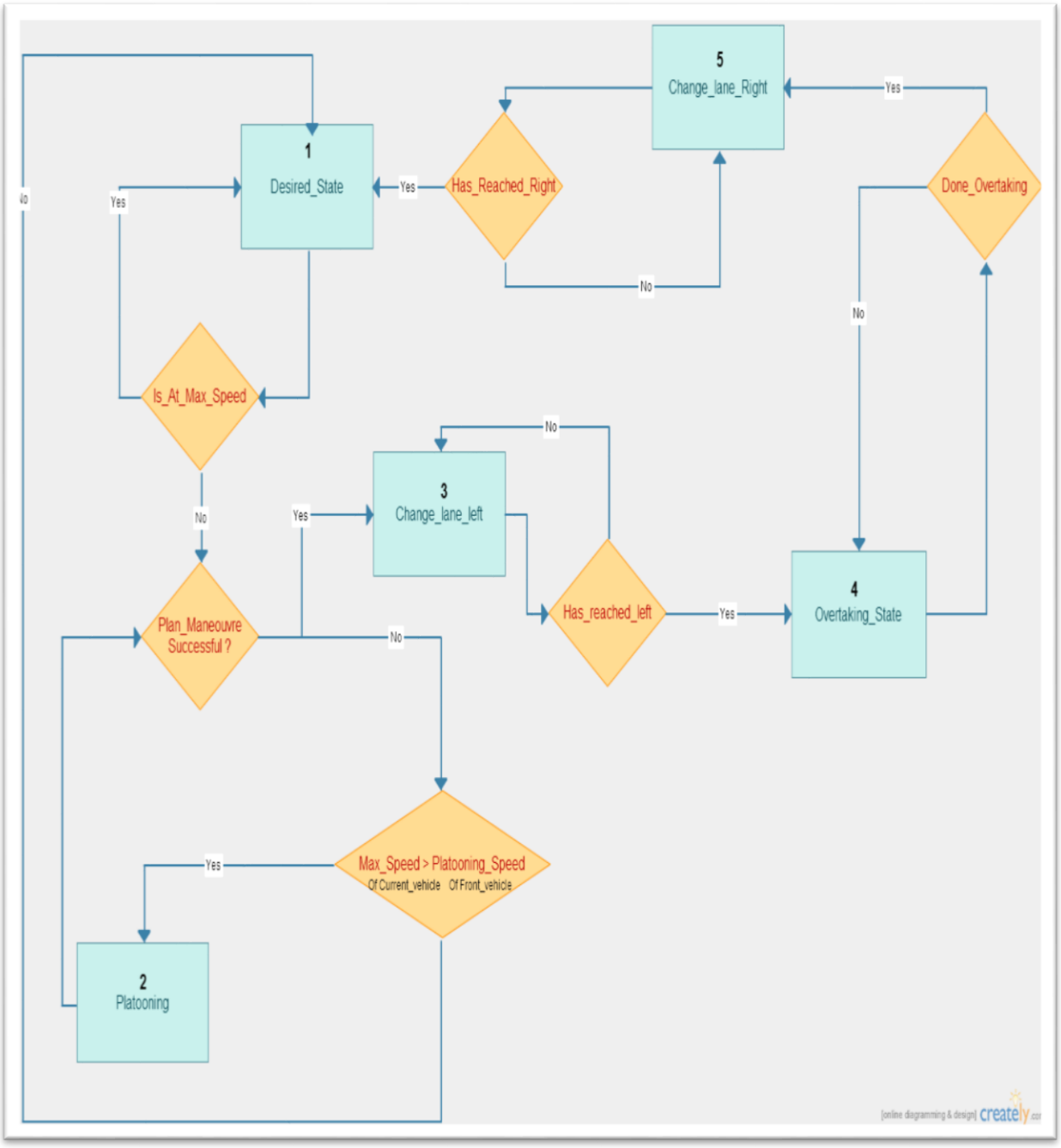


Figure 16 State Diagram to swap between maneuvering states

The agent starts in DesiredS (desired state), and the agent begins by choosing an action (Move straight with highest possible speed) to perform on the Desired state. With a planned certain sequence of actions will actually result to continue in the Desired State– though of course, the agent has to make a prediction for the set of actions. As a result of these actions, the environment can respond with a number of possible states from the mentioned set E. On the basis of this second state, the agent again chooses an action to perform. The environment responds with one of a set of possible states, the agent then chooses another action, and so on.

- **Environment for motionAgent:**

Environments are assumed to be history dependent. In other words, the next state of an environment is not solely determined by the action performed by the agent and the current state of the environment. The actions made earlier by the agent also play a part in determining the current state. For Example, an agent can only be in ChangeLeft state if it's previous state was Platooning. The current state in which the agent is, will also be determined by the actions like changing co-ordinates. To minimize the non-deterministic nature of the environment and also to include driving assistance system, a prediction algorithm based on the observations in terms of speed and position of the neighbor agents, predicts the state of the agent at a given time, checks for the possibility of applying several sequential actions and thus helps the agent to reach the Desired State with maximum speed (DesiredS).

So, in short a problem can be defined formally by five components:

- Initial State: The initial state of an agent is DesiredS. And every agent starts with zero initial velocity from a specified X-Y co-ordinates.

- A description of possible actions given to the agent. Given a particular state S, ACTIONS(s) returns the set of actions that can be executed in S. For Example, In DesiredS the applicable actions are:
  - GO_LEFT_AT_MAX_SPEED,
  - GO_RIGHT_AT_MAX_SPEED,
  - GO_STRAIGHT_AT_MAX_SPEED,

- GO_STRAIGHT_AT_PLATOONING _SPEED.

- A Description of what each action does; the formal name transition model is specified by the following example:
  - Result (DesiredS, GO_LEFT_AT_MAX_SPEED) = ChangeLeft

- A goal test, which determines whether the given state is the goal state, is carried out at every cycle. The motionAgent always tries to reach its goal state; DesiredS; and further tries to maintain it.

- The path cost function assigns numeric cost to the path. Each agent tries to search for a possible sequence of actions to achieve DesiredS (If the current state is different from DesiredS). The numeric parameter is the most optimized one if it finds such a sequence at first attempt. In other case, at next attempt it is weighted by a certain waiting time and after passing that waiting period only the agent can claim the search method.

In this scenario, since we do have complete information about the state of the environment at each point of time, we can say that the environment is “OBSERVABLE”.

For a driving environment, avoiding collisions maximizes the performance measure of all agents, so it is a “PARTIALLY CO-OPERATIVE MULTI-AGENT” environment.

Since the autonomous vehicles first agree on a consensus to execute a plan, it can be inferred that an agent knows what others are currently thinking and planning ahead in future. Because of the errors in determining the exact state and behavior of an agent after certain period of time and also because of the existence of more than one outcome for a single action, the environment is “STOCHASTIC”.

An agent has to think ahead and the current decisions could affect the future decisions, the environment is “SEQUENTIAL”.

Further, vehicle-driving is completely a “DYNAMIC” scenario as other cars and the current car keep moving, while the driving algorithm plans what to do next. Even we have maintained discrete finite number of states in the environment, Vehicle-driving is a continuous-state and continuous-time problem: the speed and location of the current vehicle and other vehicles sweep through a range of continuous values and do so smoothly over time. Taxi-driving actions (steering actions etc.) are also continuous. Inputs from sensors are discrete. So the environment can be grouped as “CONTINUOUS”.

As we can see that the agent’s state of knowledge about the “laws of physics” of the environment is known and also we can predict the probable outcomes (with some inaccuracy) on imposing certain actions to an agent, the environment can be classified as of “KNOWN” type.

In short it can be concluded that we have partially observable, partially co-operative multi-agent, stochastic, sequential, dynamic, continuous and known environment.
To formalize the abstract view of the agents, the environment can be broken down to any finite set E, of discrete instantaneous states, where E is defined as:

E = {DesiredS, Platooning, ChangeLeft, Overtaking, ChangeRight}

For an agent to remain in a particular state, it has to pass the conditional checks specified for the state. Thus, an agent can iterate in a state until the condition holds true. The states are explained below:

- *DesiredS*: It corresponds to no obstacle found in the sensor radius and therefore allows the agent to drive freely at maximum speed.

- *Platooning*: It corresponds to obstacle found state. In this state an agent either tries to find a plan to overtake or continue following the vehicle in front of it (at the speed same as that of front vehicle).

- *ChangeLeft*: As soon as an agent finds a plan to execute overtaking, it changes its current state to ChangeLeft and thus continues to remain in this state until it successfully changes it lane.

- *Overtaking:* The next phase in overtaking process is to overtake the vehicle in front of the current vehicle. So until an agent successfully completes the overtaking of the front vehicle it continues in overtaking state.

- *ChangeRight*: After a successful overtaking, the agent makes a right turn to come back to its original lane. Until it successfully makes a right lane change it continues to remain in this state.

Under these assumptions, it can be inferred that a sequence of actions is a solution to the problem. Searching for these sequence of actions is done by "Search Algorithms", which takes a problem as an input and returns solution in the form of action sequence.

- **motionAgent as a Model:**

The most effective way for the agent to handle partially observable environment is to keep track of the part of the world it can't see now. To achieve this the agent should maintain some sort of internal states that depends on the precept history and thereby reflects at least some of the aspects of the unobserved aspects of the current state.

For braking, the agent can simply observe the braking lights of front car or check if it has entered the sensor radius with the possibility of collision with the front vehicle. For other driving tasks such as changing lanes, the agents need to keep track of where the other cars are if it can't see them all at once. And for driving to be possible the agent has to keep track of where its keys are.

Updating this internal state information as time goes by requires two kinds of knowledge:

1. How world evolves independently of an agent? -: Example: An overtaking car will be generally closer behind than it actually was.
2. How agent own actions affect the world? -: Example: Steering clockwise makes a car turn right.

Our agent is therefore a Model-Based-Agent as it uses the scientific theories to build knowledge Base.

- **"motionAgent" as a model-Based Reflex Agent**

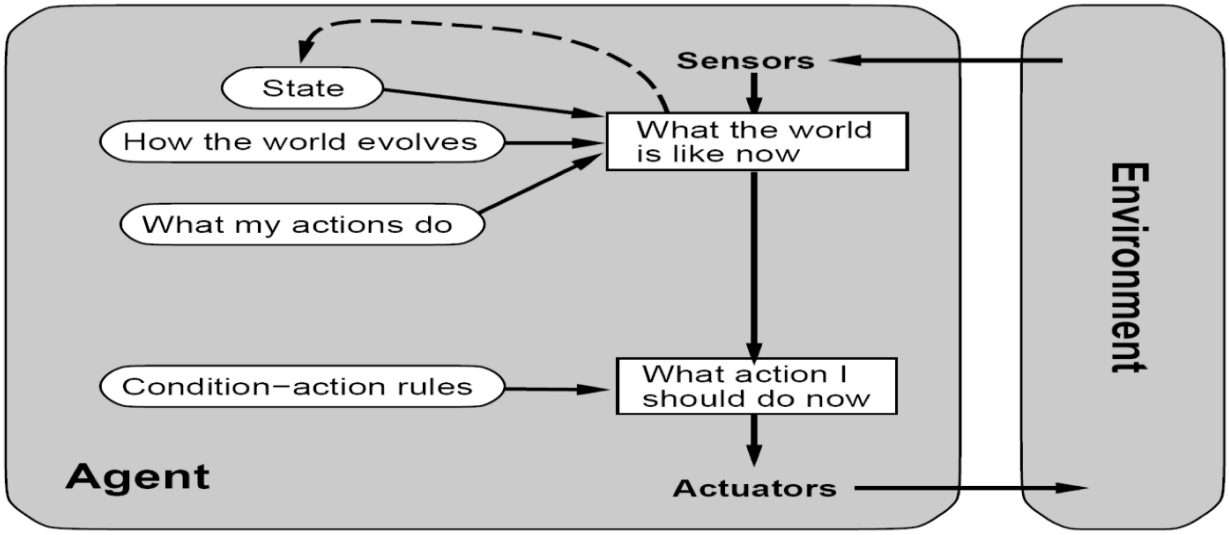


Figure 17 Reflex Agent

Our agent combines the current precept with the old internal state to generate the updated description of the current state. States are updated at every agent cycle. These states are responsible for the new internal state description. For a model to evolve as a model-Based reflex agent following questions arise:

1. What the world is like now?
2. What actions I should do now?

The first question is answered by keeping track of the neighbors at every agent cycle. The state

tracker method tracks the position, velocity and other parameters of neighboring vehicles for each vehicle. Further, best guesses are also made to determine the position after few moments of predicted sequence of actions.

The sequence of actions an agent should do is determined by the state cycle graph (Fig 19). Until the conditions for a certain state remains true, the agent keeps of repeating the same action (say change left by steering counter-clockwise).

- **"motionAgent" as a Goal-Based Agent**

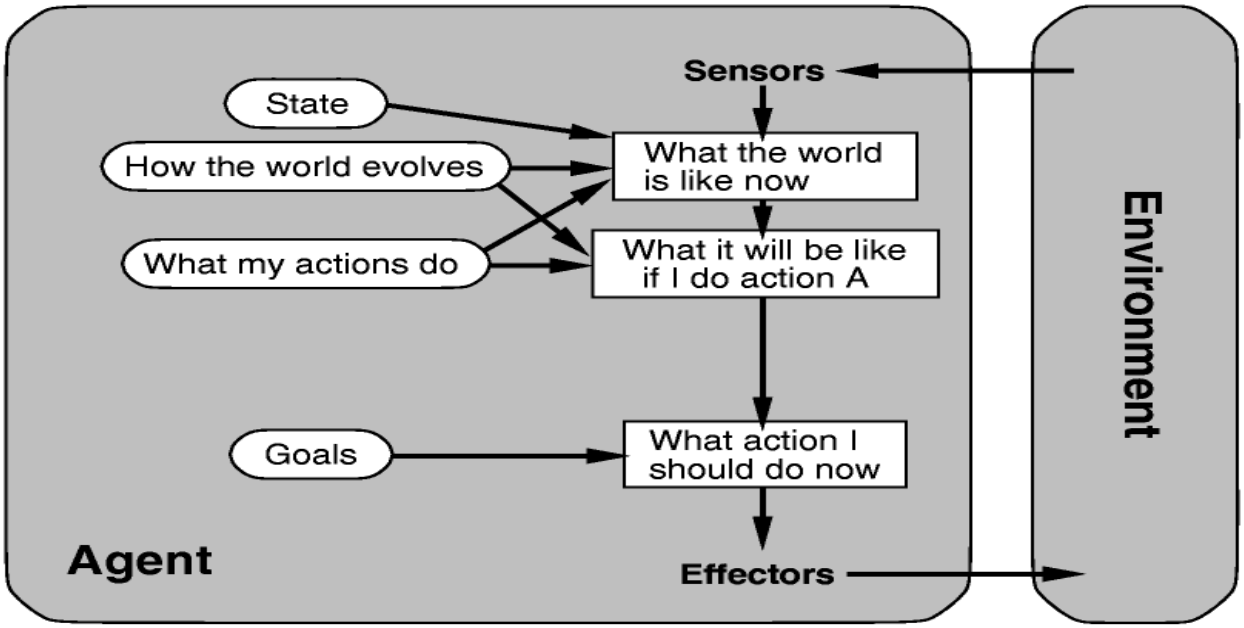


Figure 18 Goal Based Agent

It can be tricky to find a long sequence of twists and turns in order to find a way to achieve a goal. Search and planning finds action sequences that achieve the agent's goals. The questions: "What will happen if I do such-and-such?" and "Will that make me happy" has to be answered for a Goal-based agent. So it involves consideration for future. The goal for the agent is to move straight at maximum possible speed and to achieve this, it deploys search and planning algorithm, keeping in consideration for future.

Assume a scenario where an agent sees the brake lights on of the car in front of it. It can reason to slow down and further the only action available to avoid hitting is to apply the brakes, and so it does. The flexibility of goal-based agents reflects the ease to let the agents adapt to the dynamic environment. Say, instead of modifying a number of condition-rules in raining environment, it can effectively parameterize the braking efficiency and thus would apply brakes accordingly. These goal-based agents are not limited to certain bounds say a Straight-highway, rather they can be extended for non-straight highways.

- **"motionAgent" as a Utility-Based Agent**

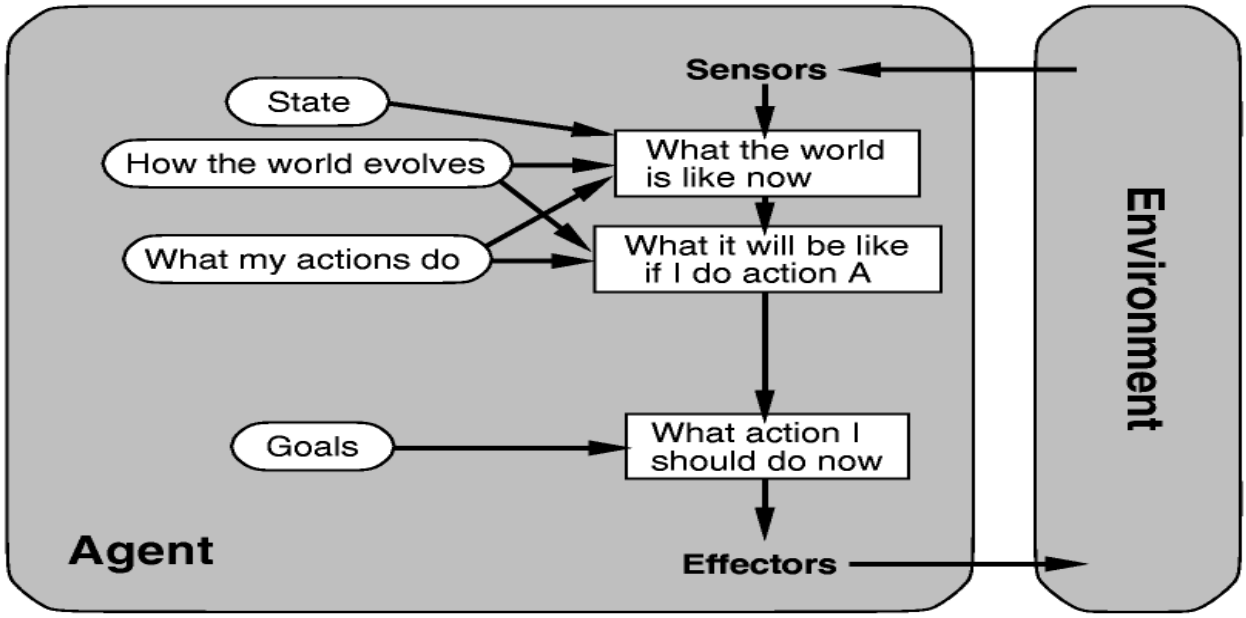


Figure 19 Utility Based Agent

Goals alone cannot generate high-quality behavior in most environments. To optimize parameters like reliability, safety, timeliness and cost, we must not be only limited to binary distinctions, say happy and unhappy states, if we talk from economists point of view, the term utility suits well. Choosing actions that maximizes the performance measure will be an optimum implementation of the project.

Is designing such an agent is simple?

1. Although the systems will be intelligent but it is not simple to design it.

2. To model and keep track of its environment tasks involve a great deal of research involved in perception, representation, reasoning and learning.

3. Choosing the maximizing-utility function is a difficult task requiring ingenious algorithms.

Finally, computational complexity also comes into play after we have chosen an ingenious algorithm.

4.6. Testing

4.6.1. Unit Testing:

1. *StateMachine*: This class defines variables to sense agent surrounding the current agent. On unit testing with several agents, it was successfully verified that the variables redefine the front ahead, front left, front right, rear left, rear right, front ahead next on every update cycle of the agent.

Figure 20 Unit testing for checking StateMachine class

2. *do_action*: On flipping between different types of maneuvers to return the appropriate waypoints, it is required to switch between the state cycle chart as shown in the figure. Unit testing on this method shows that the framework to interface Agent drive to OpenDS is ready.

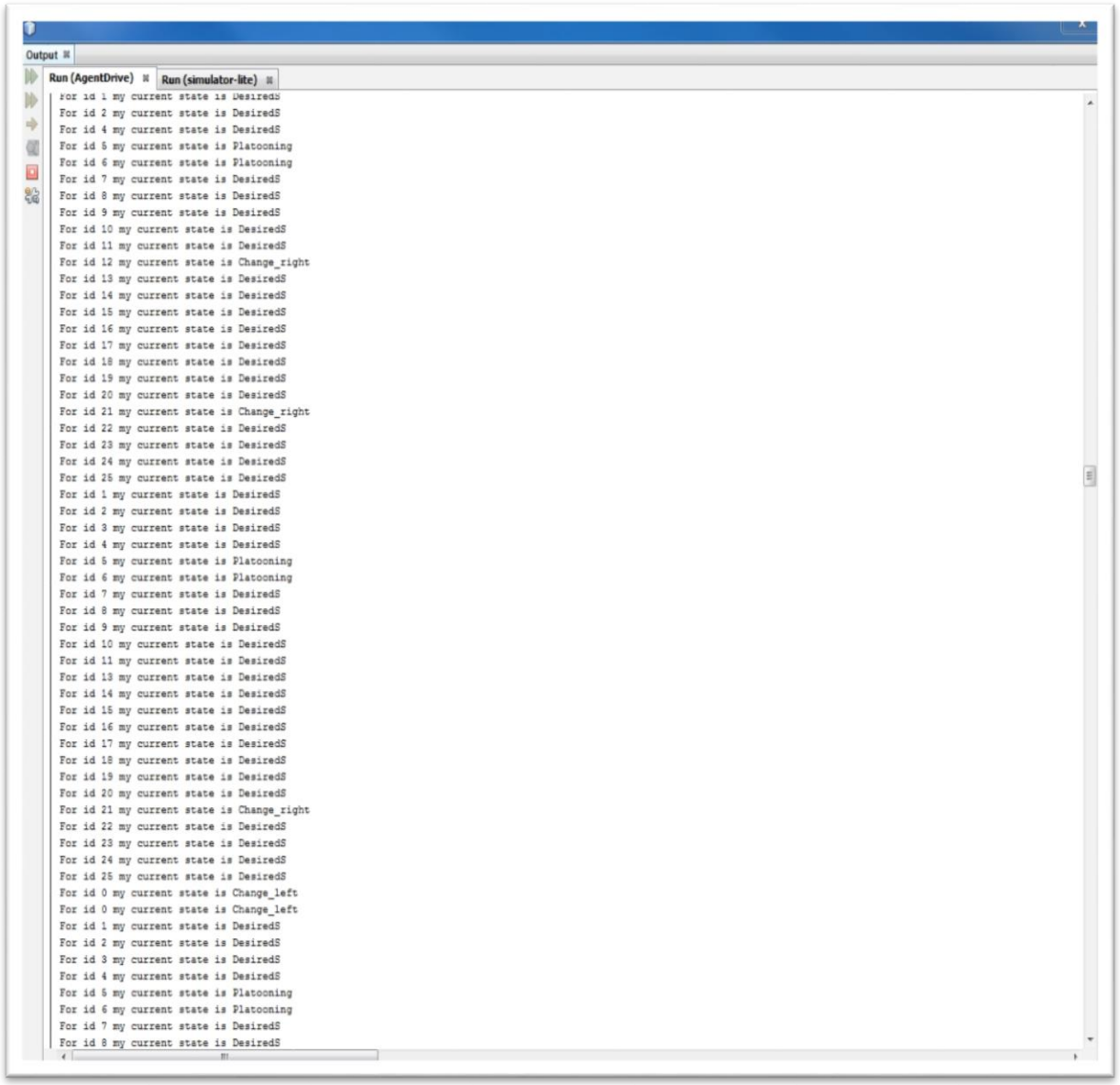


Figure 21 Unit testing for checking do_action method

3. *Conflict_Resolver*: To resolve the conflicts between several planned maneuvers, efforts are made to examine them carefully and further refine them to check all the possible conflicts between the desired reserved regions. Further, if any conflict is found, based on time priority,

the agent which requests the reservation first, is allowed to enter in the critical section with reserved space co-ordinates.

Further, to ascertain that the planning agents do not continuously raise conflicts and henceforth do not enter into deadlock situation, Exponential backoff algorithm is used. This algorithm allows re-planning after exponentially increasing time delay.

Unit testing sets the foundation to lay out the basic conflict resolving mechanism between agents.

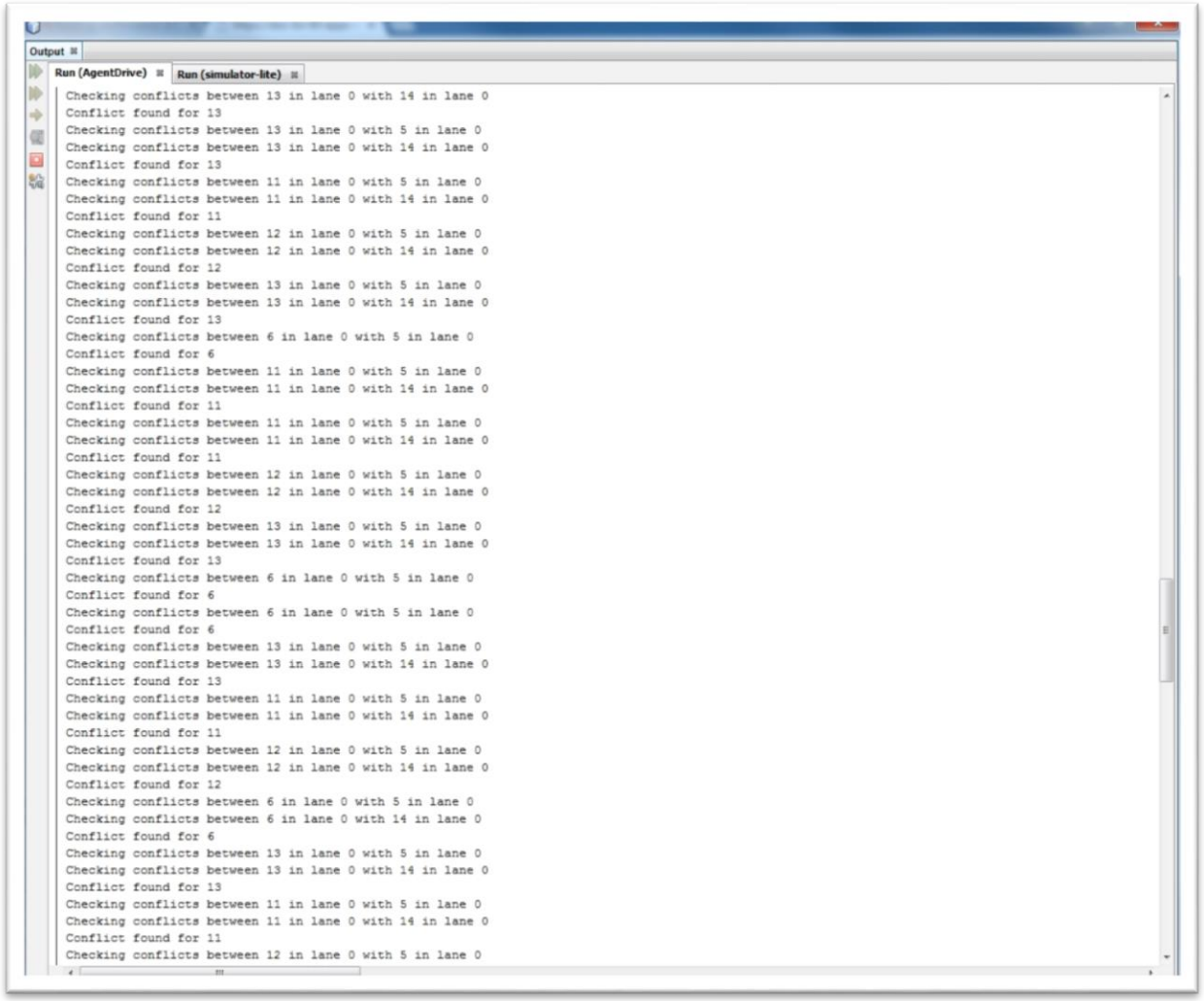


Figure 22 Unit testing for checking conflict_resolver method

4. *Sigmoid function*: To smoothen the lane change and to follow a realistic driving approach, sigmoid function (Y= Lane_gap/(1+$E^{-x/Speed}$)), is used. This method returns waypoints close to the function while changing lane, either left or back to right lane.

   It is also required to compute the inverse of the method, and further translate the parameters of the function to the corresponding agent space co-ordinates.

   Sigmoid function is tested thoroughly to enable error free integration of the agent to the Agent Drive.

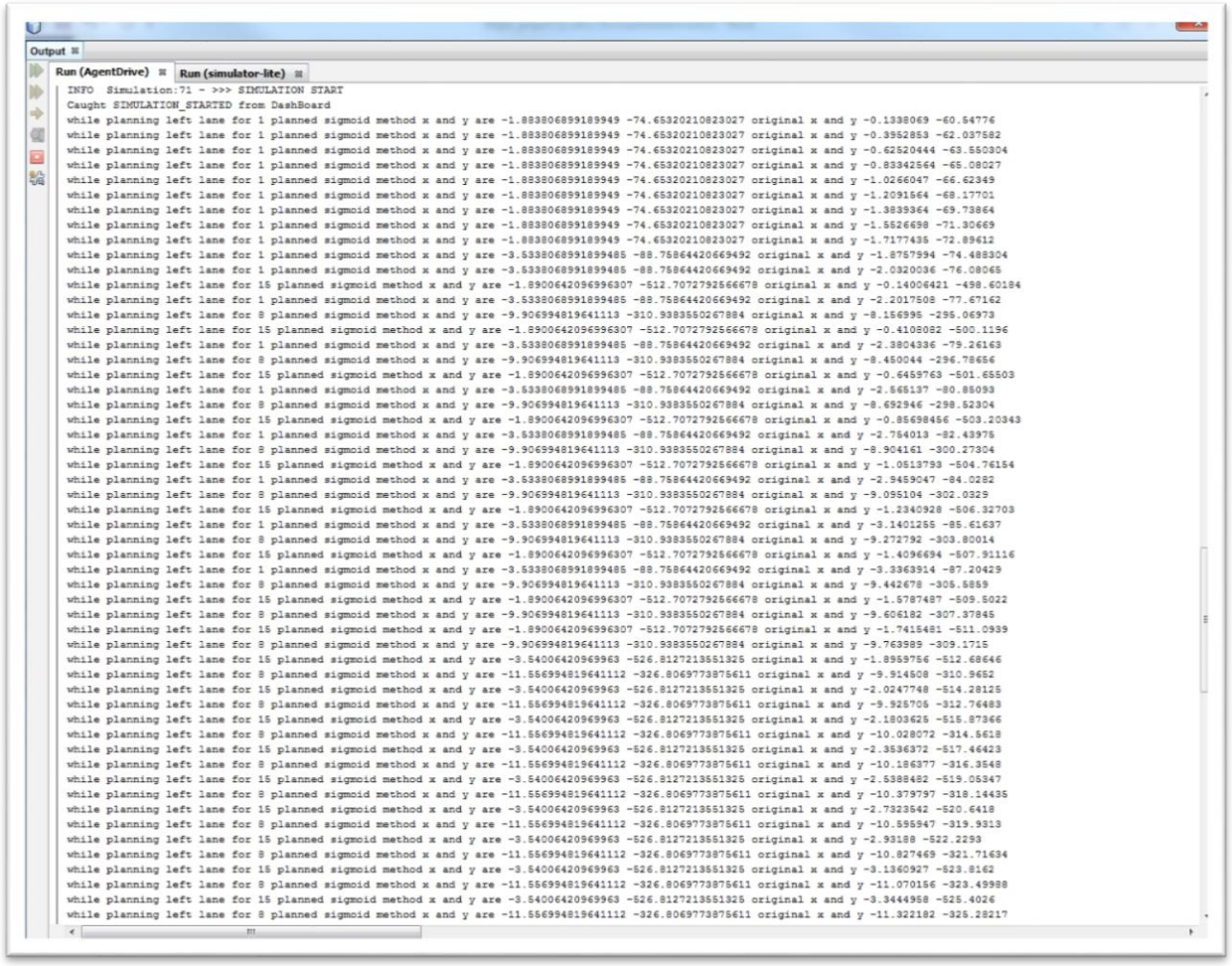


Figure 23 Unit testing for checking sigmoid method

4.6.2. Integration Testing

To work with an agent, a successful integration testing is required. The "MotionAgent" class imports packages responsible for storage and handling maneuver and events. The "MotionAgent" class inherits "Agent" class to allow working with an agent with the help of sensors in actualizing the information of the vehicle. The integration of packages to the "Agent Package" are explained below:

1. Maneuver: An instance of maneuver class returns the translation of path to waypoints by interpolating the path steps. Integration of several instances of Maneuver gave desired results on system integration. The translated instances of either of the StraightManeuver Class, AccelerationManeuver Class, DecelerationManeuver Class or LaneChangeManeuver class integrated to Action subclass of MotionAgent Class to return the translated waypoints corresponding to the specified class, works perfectly in following the specified goal.

2. RoadObjects: Every agent in the simulation environment has an identity to distinguish it from other objects. The vehicles, specified here as RoadObjects, stores all the updated information (position, speed, lane etc.) of an agent. Integrating it with MotionAgent is done with the help of sensors.

3. ExtendedRoadObjects: In order to implement a sliding window of reserved area for executing safe maneuvers, this class holding the maneuvering plans for each agent, is instantiated as an array of objects with the array index equal to the agent id of road object. Also it stores the critical section value for each agent. The critical section value is set true when an agent start executing maneuver within its planned domain.

4. Sensors: As explained above, sensors help in either sensing the current state of the road object or in returning the state of collection of all cars. Doing an iteration over the returned collection can avail parameters like id, position, speed, lane and so on of the road object. Integration of sensor to every agent works accurately in all situations.

5. Simulation Cycle: For each agent, plans to OpenDS are sent at defined intervals. OpenDS waits for the plans and after receiving them in the form of plain text message as waypoints for each of them, maps to the corresponding agent. These plans must be sent at each time step of the simulation and further it is ensured to prevent infinite loops, as they can hinder sending of plans to OpenDS. Infinite loop testing after integrating the features with the MotionAgent worked with no delay in sending the plans.

4.6.3. Test Results

Added features to the extended MotionAgent class, passed unit testing as well if passed with the required parameters.

Also the Integration testing of MotionAgent with other packages and classes of the Agent Drive and OpenDS, worked perfectly.

5. Evaluation

We simulated the scenarios to a much realistic scenario as to that of the generalized java applet simulation. And for the following three conditions these simulations were done:

- A human overtakes an automated car: It simply shows how we in real traffic try to overtake a car. And for this specific case let's say the car was automatic.
- *An Automated car overtakes a manually controlled car*:
-

5.1. Output/Results:

a. *A Human overtakes an automated car*:

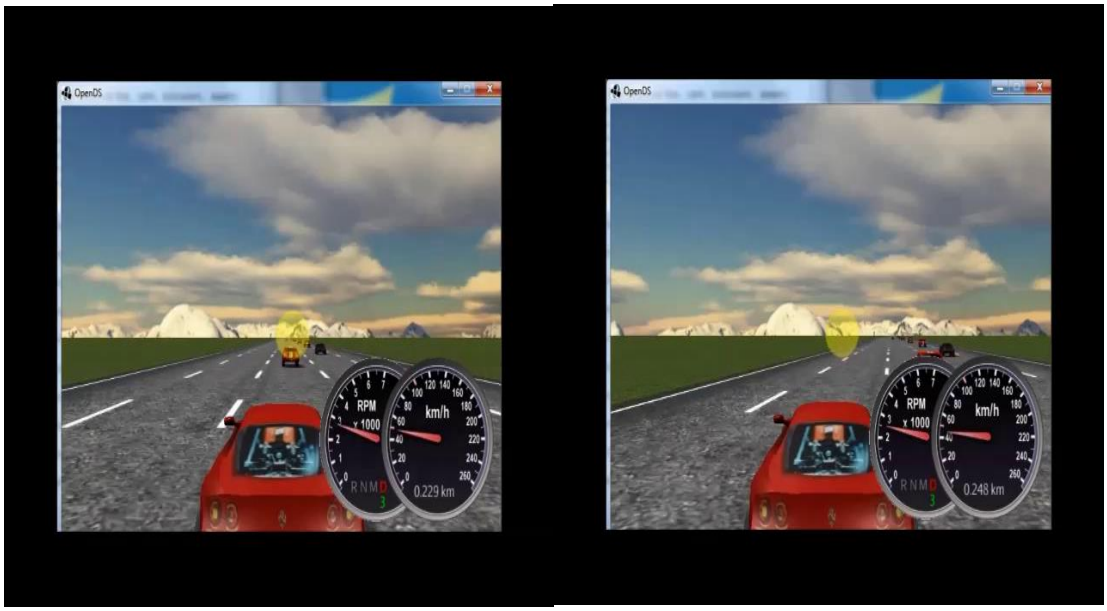
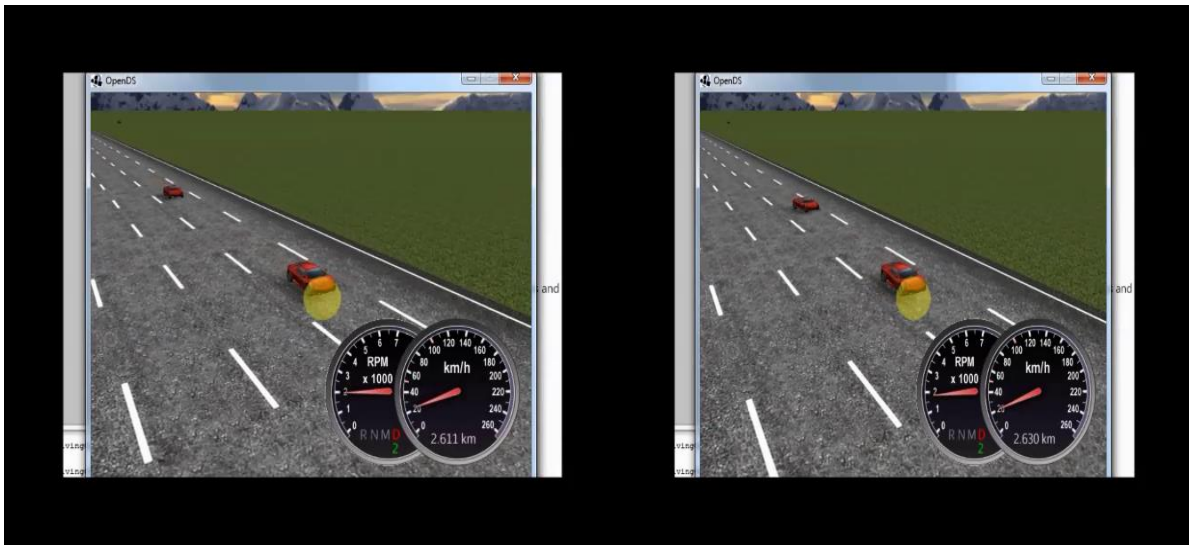
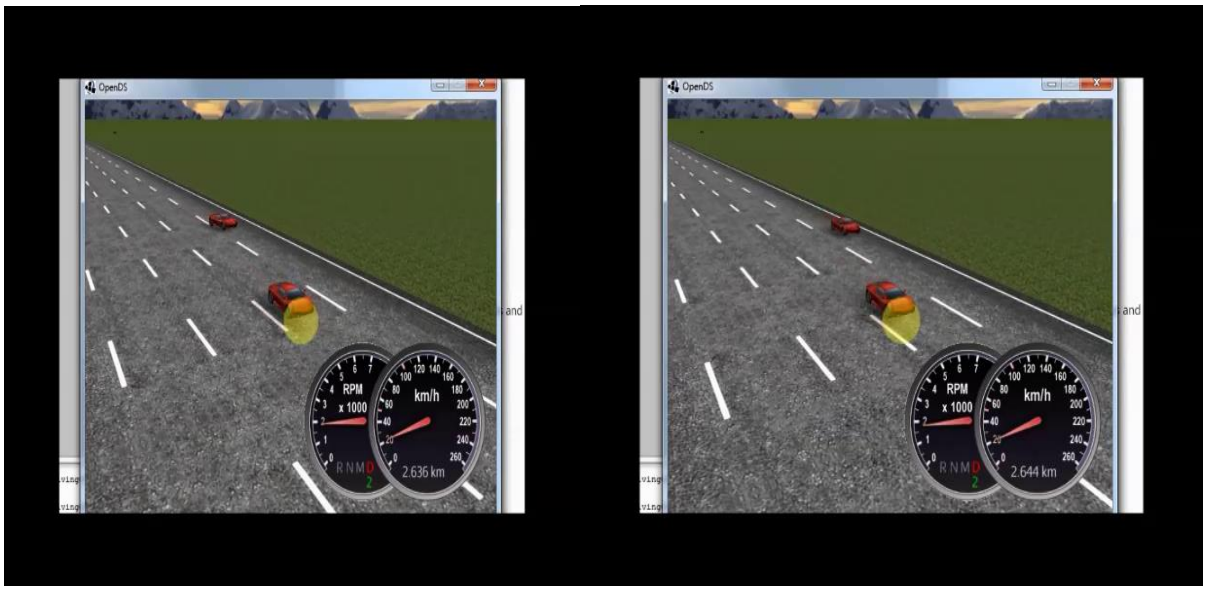


Figure 24 Screenshots in sequence as shown from 1-6 to show how a human maneuvers an automated car in AgentDrive

b. *An Automated car overtakes a manually controlled car*:





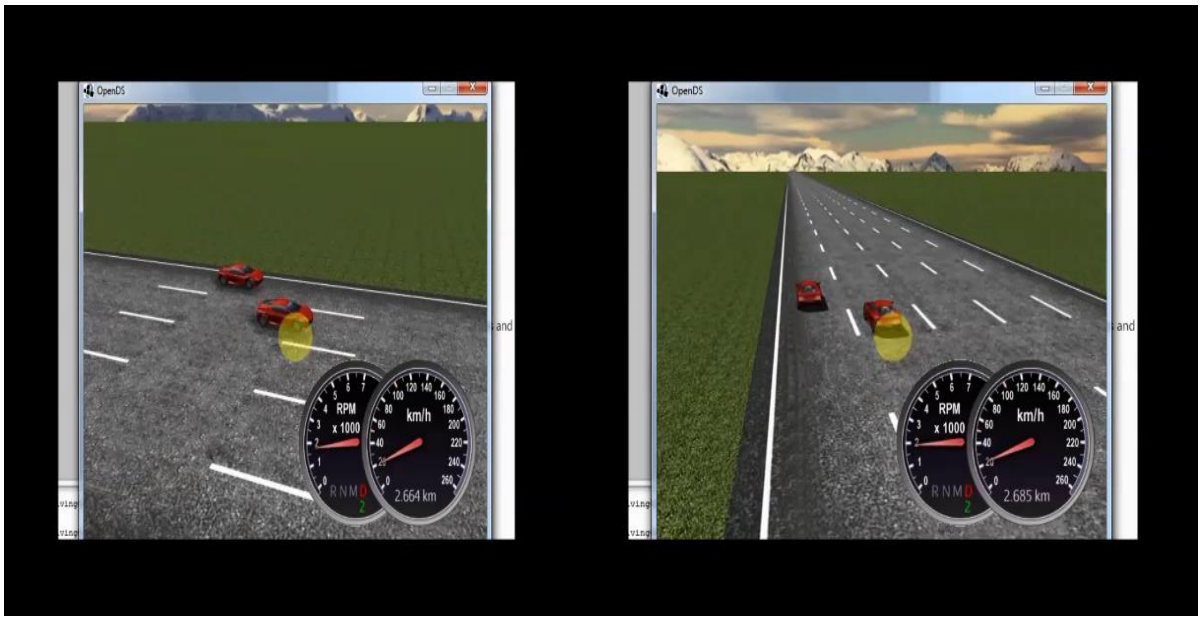


Figure 25 Screenshots in sequence as shown from 1-6 to show how a an automated car maneuvers a manually controlled car in AgentDrive

c. *An automated car maneuvering another automated car*:

Figure 26 Screenshots in sequence as shown from 1-6 to show how an automated car maneuvers another automated car in AgentDrive

5.2. Results Analysis:

a. *A human overtakes an automated car*: According to European rules, we must drive at the right lane and overtaking is allowed in the lane left to it, if the approaching vehicle has speed higher than the speed of the vehicle in front of it.

b. *An Automated car overtakes a manually controlled car*: As soon as the vehicle finds the possibility of performing a safe maneuver, it changes its current state to LeftLanechange state, otherwise it keeps on platooning the front vehicle.

c. *An automated car maneuvering another automated car*: The logic discussed in the case of maneuvering a human controlled car by an automated car applies here as well. An agent considers every other agent as an obstacle if it tries to affect its speed and therefore the planning algorithm works for every another agent.

5.3. Discussion:

a. Plan Sending: At every simulation cycle plans synchronization with OpenDS works accurately.

b. Predicting a Maneuver: The predicted planned path almost matches with the actual position after performing the safe maneuver. However, there is a need to make it more accurate, with very less error percentage.

c. Sensing a Maneuver start after finding an obstacle in the range of sensor radius (here 40metres), and further finding a plan to execute maneuver is carried out by switching states of the agent. In each state change, Agent returns the waypoints necessary to remain in the corresponding state.

6. Conclusion and Future work

6.1. Conclusion

In this chapter we summarize the contents of this thesis. We conclude the outputs of the work. General results of the experiments are followed by the ideas and proposals for the future work.

The goal of this thesis was to design and to implement a system that would allow to execute cooperative driving maneuvers of autonomous cars in driving simulator (OPenDS) and AgentDrive. The purpose of the concept is to maximize traffic output, avoid collisions and drive with co-operation and planning. The application of the concept is to allow automated execution of maneuvers for driverless vehicle/autopilot and an Intelligent Driver Assistance system. I studied and described state-of-the art of the OpenDS and AgentDrive and also the research on mathematically modeling our application. We implemented a predicted maneuvering module, collision avoidance module, integrated and adapted two collision avoidance techniques into the module. The module is developed as a part of the designed system.

We also prepared a scene and several scenarios to run the experiments in OpenDS. All the system is build on the communication protocol which was already designed and implemented.

The experiments in general show that the system as it is designed can be used for required purposes. The safety and special features of navigation depend on the vehicle sensor method implementation. The comparison of the methods is possible in the system and possibilities of analyzing the methods is revealed. The concept of the driving with the driver assistance system (based on predictions) presenting the outputs of the simulation is shown to be relevant.

6.2. Scope of Future Work

The combined interface of OpenDS and AgentDrive provides a rich simulation platform to test several specific scenarios related to the project. Creation and Instantiation of agents is not very complicated and thus it becomes a good tool for researchers to use the simulator for specific experimentation purposes. With a realistic touch in OpenDS via real time physics engine and

other driving parameters and a good link up of AgentDrive with OpenDS provides us an opportunity to explore further, the simulation tool.

The work on autonomous maneuvering although completed at waypoint layer (as explained in section), needs to be refined further with a macro and meso layer approach [30]. This will provide a better abstraction hiding the low level implementation details and further allowing the plans to be constructed hierarchically in a systematic fashion by consulting the layers of abstraction as necessary.

References

[1] M. Da Lio, F. Biral, M. Galvani, and A. Saroldi, “Will intelligent vehicles evolve into human-peer robots?” in Intelligent Vehicles Symposium (IV), 2012 IEEE. IEEE, 2012, pp. 304–309.

[2] An Introduction to Multiagent Systems Michael Wooldridge Department of Computer Science, University of Liverpool, UK

[3] Artificial Intelligence: A Modern Approach Book by Peter Norvig and Stuart J. Russell

[4] Poisson distribution: http://en.wikipedia.org/wiki/Poisson_distribution

[5] Nagel Schreckenberg model: http://en.wikipedia.org/wiki/Nagel–Schreckenberg_model

[6] A. Reuschel, “Fahrzeugbewegungen in der kolonne,” Oesterreichisches Ingenieur-Archiv, vol. 4, no. 3, p. 4, 1950.

[7] G. F. Newell, “Nonlinear effects in the dynamics of car following,” Operations Research, vol. 9, no. 2, pp. 209–229, 1961.

[8] T. H. Rockwell, R. L. Ernst, and A. Hanken, “A sensitivity analysis of empirically derived car-following models,” Transportation Research 2,pp. 363–373, 1968.

[9] R. Wiedemann, “Simulation des verkehrsflusses schriftenreihe des instituts f¨ur verkehrswesen,” Universit¨at Karlsruhe, no. 8, 1974.

[10] P. G. Gipps, “A behavioural car-following model for computer simulation,” Transportation Research Part B: Methodological, vol. 15, no. 2, pp. 105–111, 1981.

[11] “A model for the structure of lane-changing decisions,” Transportation Research Part B: Methodological, vol. 20, no. 5, pp. 403–414, 1986.

[12] T. Toledo, “Integrated driving behavior modeling,” Ph.D. dissertation, Massachusetts Institute of Technology, 2002.

[13] California Partners for Advanced Transit and Highways (PATH). Univ. California, Berkeley. [Online]. Available: http://www.path.berkeley.edu

[14] D. Reichardt, M. Miglietta, L. Moretti, P. Morsink, and W. Schulz, “CarTALK 2000—Safe and comfortable driving based upon inter-vehicle- communication,” presented at the IEEE Intelligent Vehicle Symp., Versailles, France, Jun. 2002.

[15] H. Hartenstein, B. Bochow, M. Lott, A. Ebner, M. Radimirsch, and D. Vollmer, “Position-aware ad hoc wireless networks for inter-vehicle communications: The fleetnet project,” presented at the ACM Symp. Mobile Ad Hoc Networking and Computing (MobiHoc’01), Long Beach, CA, Oct. 2001.

[16] H. Yang and M. G. H. Bell, “Models and algorithms for road network design: A review and some new developments,” Transp. Rev., vol. 18, no. 3, pp. 257–278, 1998.

[17] H. Yang and M. G. H. Bell, “Traffic restraint, road pricing and network equilibrium,” Transp. Res. Part B, Methodological, vol. 31, no. 4, pp. 303– 314, Aug. 1997.

[18] H. Yang, “Heuristic algorithms for the bilevel origin–destination matrix estimation problem,” Transp. Res. Part B: Methodological, vol. 29, no. 4, pp. 231–242, Aug. 1995.

[19] Z. Gao, H. Sun, and L. L. Shan, “A continuous equilibrium network design model and algorithm for transit systems,” Transp. Res. Part B, Methodological, vol. 38, no. 3, pp. 235–250, Mar. 2004.

[20] R. Sengupta, S. Rezaei, S. Shladover, D. Cody, S. Dickey, and H. Krishnan, “Cooperative collision warning systems: Concept definition and experimental implementation,” J. Intell. Transp. Syst., vol. 11, no. 3, pp. 143–155, Jul. 2007.

[21] N. A. Lynch, Distributed Algorithms. San Francisco, CA: Morgan Kaufmann, 1997.

[22] M. H. DeGroot, BReaching a consensus,[J. Am. Statist. Assoc., vol. 69, no. 345, pp. 118–121, 1974.

[23] J. A. Benediktsson and P. H. Swain, BConsensus theoretic classification methods,[IEEE Trans. Sys., Man, Cybern., vol. 22, no. 4, pp. 688–704, Apr. 1992.

[24] S. C. Weller and N. C. Mann, BAssessing rater performance without a Fgold standard_using consensus theory,[Med. Decision Making, vol. 17, no. 1, pp. 71–79, 1997.

[25] V. Borkar and P. Varaiya, BAsymptotic agreement in distributed estimation,[IEEE Trans. Autom. Control, vol. AC-27, no. 3, pp. 650–655, Jun. 1982.

[26] J. N. Tsitsiklis, BProblems in decentralized decision making and computation,[Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Lab. Inf. Decision Syst., Massachusetts Inst. Technol., Cambridge, MA, Nov. 1984.

[27] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans, BDistributed asynchronous deterministic and stochastic gradient optimization algorithms,[IEEE Trans. Automatic Control, vol. 31, no. 9, pp. 803–812, Sep. 1986.

[28] D. P. Bertsekas and J. Tsitsiklis, Parallell and Distributed Computation. Upper Saddle River, NJ: Prentice-Hall, 1989.

[29] THE 1997 AUTOMATED HIGHWAY FREE AGENT DEMONST by Chuck Thorpe, Todd Jochem, Dean Pomerleau Robotics Institute, Carnegie Mellon University.

[30] A Micro-Meso-Macro Approach to Intelligent Transportation Systems by David Sanderson, Dıdac Busquets and Jeremy