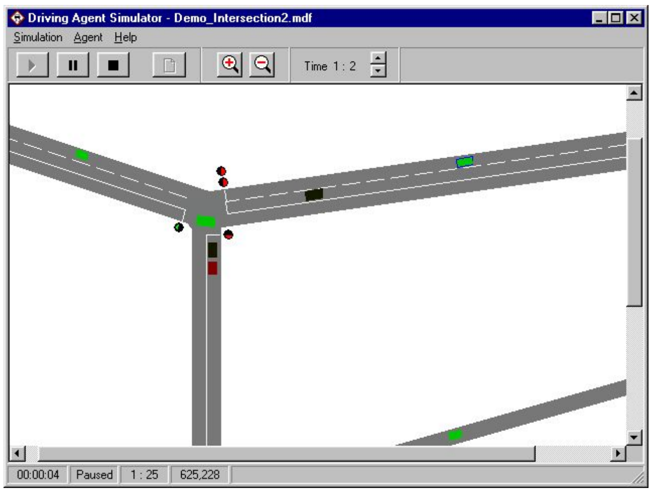
INTELLIGENT DRIVING AGENTS

The agent approach to tactical driving in autonomous vehicles and traffic simulation



Master's thesis of Patrick A.M. Ehlert

Delft University of Technology
Faculty of Information Technology and Systems

January 2001

TU Delft

Ehlert, Patrick A.M. (P.A.M.Ehlert@twi.tudelft.nl or patrickehlert@yahoo.com)

Master's thesis, January 2001
"*Intelligent driving agents:*
The agent approach to tactical driving in autonomous vehicles and traffic simulation"

Delft University of Technology, The Netherlands
Faculty of Information Technology and Systems
Knowledge Based Systems Group

Keywords: Artificial intelligence, intelligent agents, multi-agent systems, autonomous vehicles, tactical driving, driving behaviour, traffic simulation, microscopic traffic simulators

Preface

This thesis describes the research that I have done for my graduation project at the Knowledge Based Systems (KBS) group of the Delft University of Technology, headed by Prof. dr. H. Koppelaar. Several researchers of the KBS group are involved in the TRAIL research school, headed by Prof. dr. ir. P.H.L. Bovy. The TRAIL school does research on transport, infrastructure and logistics, and in this report I have tried to make a contribution to that area.

Project

The project described in this report started about a year ago with a suggestion of my supervisor Leon Rothkrantz to study intelligent agents (smart autonomous computerised entities) driving vehicles in an urban environment. He asked me to give it a thought whether or not I wanted to work on this for my graduation project. I promised him that I would look into it and started reading about agents and autonomous vehicles. During this first analysis phase I became more and more enthusiastic about the subject and we decided to design a model of a tactical-level driving agent. After the design we needed a method to evaluate the developed model, so next we did some research on simulation techniques. While studying traffic simulations it became clear that the developed agent model could very well be used in traffic micro-simulations, simulations that use separate entities with their own functionality to model a system. Our agent design was then expanded to contain different types of driving behaviours so that it could be used to simulate various types of drivers. After spending some time searching for a suitable simulator to test our agent, we decided to create our own prototype simulation program. The result of all this lies before you in the form of this report.

Report overview

The report is organised into three parts. The first part describes the theory behind the project and functions as a short introduction to intelligent vehicles, agents, and multi-agent systems. The second part describes the design and implementation of a tactical-level driving agent, the way it can be used in traffic simulations and the simulator that was created to test the agent. In the third part the project is evaluated and some conclusions are presented.
The last appendix of this report contains a paper about the use of the driving agent in microscopic traffic simulations. This paper functions as an extensive summary of this report.
I have tried to make this report accessible to everyone with a little knowledge of computers and programming. Difficult terms and concepts are explained and a glossary can be found at the end of the report. Some parts in the design of the agent will probably be difficult to grasp for people with no programming experience, but I am confident that the larger part of this report can be well understood. Hopefully my family and friends can now better comprehend what I have been working on all this time.

Acknowledgements

First of all, I would like to thank Leon Rothkrantz for his guidance and advice during the project. I have really enjoyed our weekly ‘chats’. Furthermore, André van Bunnik, Stefan Visser, Wilco Bloom, and Rob van de Laar are thanked for their help and advice on a number of programming problems and for proof reading several parts of this thesis. Thanks also to Christian van der Geer for his comments on the first part of this report. Last but certainly not least, special thanks are due to my family and friends for their moral support.

Patrick Ehlert
Delft, January 2001

Summary

Autonomous vehicles can reduce or solve problems such as traffic accidents, congestion, and CO_2 emissions. Although there is a lot of interest in autonomous vehicles, practical applications are still rare, mainly because of the limited sensor technology. Expectations are that the driving support systems that are currently introduced in the passenger-car market will gradually evolve and take over more and more driving tasks in time. The ultimate driving support system will be a fully autonomous and intelligent vehicle that automates all driving tasks.

In this thesis, we present a general model of an intelligent agent that can control an autonomous vehicle in different environments. An intelligent agent is an autonomous, computerised entity capable of sensing its environment and acting intelligently based on its perception. Our driving agent performs tactical-level driving which consists of all driving manoeuvres that are selected to achieve short-term objectives.

Besides controlling real-life vehicles, intelligent agents can also be used to control simulated vehicles in traffic simulations. We have expanded the functionality of our driving agent so that it can simulate different driving styles. To ensure fast reaction times, the agent’s driving task is divided in several competing and reactive behaviour rules. Each set of behaviour rules deals with a different aspect of driving in an urban environment. The agent is implemented and tested in a prototype traffic simulator program. The simulator models multi-lane roads, intersections, traffic lights, light controllers and vehicles. Every vehicle in the simulator is controlled by a driving agent and each agent is equipped with its own set of behavioural parameters to simulate individual drivers. Preliminary experiments have shown that the agents exhibit human-like behaviour ranging from slow and careful to fast and aggressive driving.

Table of Contents

Chapter 1: Introduction

The subject of this study is the use of intelligent agents controlling autonomous vehicles and their application to traffic simulators. In the first section the problem setting is discussed, followed by a brief explanation of several problem-related terms such as the concept of tactical driving (section 1.2), intelligent vehicles (section 1.3), and traffic simulators (section 1.4). Section 1.5 describes the goals of the project and methods used to reach those goals. The last section, which is section 1.6, gives an overview of the structure of this report.

1.1 Problem setting

The automobile can be viewed as one of the greatest inventions of the 20th century. In the Netherlands somewhere between 80% and 85% of all kilometres travelled is done by car and in other countries this number is even higher [TRAIL 1999]. However, in the last decade the drawbacks of using cars have become more and more apparent.

One problem is that accidents happen frequently, often with fatal consequences. Estimates are that more than 90% of all driving accidents are caused by human errors such as fatigue, inattention, or intoxicated driving [Smiley and Brookhuis 1987].

A second problem is that emissions from cars affect the environment in an unsolicited manner. Scientists suspect that CO_2 emissions are responsible for the 'greenhouse effect' and slowly heat up the earth's atmosphere.

In the last two decades, the use of automobiles has intensified dramatically and traffic jams have become common. Expectations are that the mobility of people will continue to increase in the future, despite possible traffic-reducing trends like teleworking and growing Internet use. More and more methods are tried in an attempt to solve this problem. Some examples are dedicated lanes, variable message signs, and 'rekeningrijden', a proposal for an automated toll system in the Netherlands.

The use of autonomous vehicles driven by intelligent agents can reduce or solve the problems mentioned above. An intelligent driving agent can control a vehicle on its own, without a human directly controlling it. Since computers normally do not make mistakes (the user or programmer does!) and do not have problems like distractions or fatigue, the intelligent vehicle can drive more safely than humans, thus reducing the amount of accidents. Recent studies have shown that the use of intelligent driver support systems can increase road capacity and reduce traffic jam occurrence on motorways [Minderhoud 1999], [TRAIL 1999]. Expectations are that fully autonomous vehicles can increase road capacity further. Research has also proven that by 'putting the pedal to the metal' vehicles can produce more than 40 times the emissions of normal driving. Intelligent driving agents can make sure that both the speed and acceleration of a vehicle remain under certain limits, reducing the total amount of CO_2 emissions.

Other uses of intelligent driving agents can be found in traffic simulators, which are computer programs that simulate traffic flow. Traffic simulators are used to study traffic bottlenecks, the effects of certain flow-improving measures, and human driving behaviour. Intelligent agents can simulate the driving behaviour of individual drivers, each with its own individual peculiarities and driving style. This way, the vehicles in a traffic simulator will behave more realistically, the simulator is more flexible, and the interaction between different drivers can be studied.

1.2 The driving task

Driving in traffic is to a great extent a matter of dealing with incoming sensor information and adjusting the speed and course of a vehicle accordingly. To perform the driving task, several sources of information are available, for instance:

- Drivers *observe their environment.* They notice vehicles nearby, look at traffic signs and watch out for the police when they are driving too fast. Most driving decisions are based on the observations of a driver.
- A driver can get *information from the vehicle* itself. The dashboard of a car shows speed, fuel level, oil temperature etc., but a driver can also get information from the noise of the car's engine or exhaust.
- The *cognitive abilities and skills* of a driver can be seen as an 'internal' information source. Based on experience and common sense, every road user can anticipate to situations that are likely to happen. Every licensed driver has learned how to drive (although in some cases we have our doubts) and thus has knowledge about driving.
- Naturally, *other information sources* can also be available. Examples include the radio traffic services or variable message signs that give information about the current traffic situation.

The driving task can be separated into a hierarchical structure with three levels: the strategic, tactical, and operational level. At the highest or strategic level, goals are determined and a route is planned based on these goals. In the last five years several commercial car-navigation systems have become available. These systems allow drivers to enter a destination into an onboard computer and direct them to their destination using the Global Positioning System (GPS).
The tactical level deals with manoeuvres that are selected to achieve short-term objectives. Here the driver is primarily dealing with other road users, traffic systems and keeping his vehicle on the road. His behaviour is dictated by the current situation, but is also influenced by the goals set at the strategic level.
The actual actions that are performed by the driver form the lowest or operational level of driving. It involves the control of the vehicle by using its steering wheel, pedals and other car controls. Typical operational-level actions include steer left, accelerate, and shift to second gear.
A lot of research has been done on both the strategic (planning) and operational level (robotics). However, the decisions required at the tactical level are much more complicated and no perfect control system has been found to date.

Table 1: The driving task and types of knowledge

	Strategic	**Tactical**	**Operational**
Knowledge-based	Navigation in unfamiliar area	Controlling a slipping car on icy roads	Student driver on first lesson
Rule-based	Choice between familiar routes	Passing other cars	Driving an unfamiliar car
Skill-based	Commuter travel	Negotiating familiar intersections	Road following around corners

Another way of looking at the driving task is to divide it into types of knowledge needed to perform the task. A distinction is made between knowledge-based, rule-based, and skill-based performance (see also Table 1). With knowledge-based performance the driver has to think about what he wants to do and how to do it. The opposite is skill-based performance where the driver performs his highly practised task automatically without consciously thinking about it. Rule-based performance can be classified in between. The driver has experience doing the task and most of the time a predetermined set of rules can be applied that have proved to be successful previously. The process of choosing a rule is done more or less consciously but once a rule is chosen, the appropriate actions are carried out automatically.
Most task elements are performed simultaneously. A driver needs to handle both speed and heading, and at the same time watch out for other vehicles. An experienced driver can do this without much effort, whereas student drivers find this much more difficult. This has to do

with the distinction between automatic and controlled behaviour [Hoedemaeker 1999]. Automatic behaviour is parallel by nature, very fast, independent of workload, and people are unaware of any processing going on. Controlled behaviour is serial by nature, much slower, workload dependent, and there is awareness of processing.

1.3 Intelligent autonomous vehicles

Many discussions in the field of Artificial Intelligence (AI) deal with the definition of the term ‘intelligence’. We will not discuss any of this here, but instead present the reader the meaning of intelligence as it is used in this report, and more specifically the meaning of intelligence related to autonomous vehicles.

We define intelligence as the ability of an entity to perform a given task. Intelligence reveals itself in problem-solving behaviour. Especially humans are very good in problem solving. Just take a look at all the machines and tools we invented in the last fifty years. But what is it about humans that makes them intelligent? One aspect certainly is that they possess knowledge about the way the world works. Knowledge can be seen as an integral part of intelligence.

If we apply this definition to vehicles, we can say that a vehicle is intelligent if it can do a certain given task, in this case (a subtask of) driving. To perform the task the vehicle should be equipped with knowledge about driving. Ideally, an intelligent vehicle would be able to drive itself without a human being in direct control, i.e. it would be autonomous[1]. However, in practice this is hardly the case. There are some cars that possess a bit of intelligence, for example in the form of a navigation system that helps the driver find his way, but the driver still has to control the car. In the last two decades autonomous guided vehicles have been applied to automatically transport materials or people, but these types of vehicles need several adjustments to their environment to function. The ideal intelligent autonomous vehicle would be able to drive on today’s roads without any special modifications to the existing infrastructure.

1.4 Traffic simulators

Traffic simulators can be used for different purposes. The most common are traffic analysis, testing new vehicle-control algorithms, and studying human behaviour while driving.

1.4.1 Traffic analysis

Infrastructure improvements are very costly and any improvement must be carefully evaluated for its impact on the traffic flow. Computer traffic simulations form a cost-effective method for making those evaluations. Simulations can be used to evaluate the modifications not only under normal circumstances, but also in hypothetical situations that would be difficult to create in the real world. Obviously, the used simulation model needs to be accurate in modelling these situations and in predicting their outcome.

Most traffic simulators that are used today are macroscopic simulators, i.e. they use mathematical models that describe the flow of all vehicles. These models are often derived from fluid dynamics and treat every vehicle the same. Only the more advanced models can differentiate between vehicle types (e.g. cars, trucks, and busses) and even then all vehicles are treated equally within one vehicle type.

In real life many different types of vehicles are driven by different kind of people, each with their own behaviour and peculiarities, thus making traffic flow rather unpredictable. In microscopic simulations, also called micro-simulations, each element is modelled separately, allowing it to interact locally with other elements. Each element can be seen as an individual with the resulting traffic flow being the emergent behaviour of the simulation. Microscopic simulators are able to model the traffic flow more realistically than macroscopic simulators.

[1] Note that in this thesis the terms autonomous vehicle and intelligent vehicle denote the same.

1.4.2 Testing new algorithms

Another use of traffic simulators is testing new control algorithms for intelligent vehicles. To accurately test these algorithms it is necessary to put the vehicle in several potentially dangerous situations. For obvious reasons this cannot be done in real life, but simulation provides a safer and cheaper way.
Not many traffic simulators allow this kind of testing, but a very good example of a simulator especially designed for testing tactical-level driving algorithms is the Simulated Highways for Intelligent Vehicle Algorithms (SHIVA) simulator [Sukthankar et al 1996].

1.4.3 Studying human behaviour

Some driving simulators are used to study human driving behaviour. Usually, these simulators consists of a large screen showing the simulation environment and several controls that are used by the person under study, for example a steering wheel and pedals. Examples of research topics in this area include evaluation of road design, evaluation of in-car electronic devices, behavioural assessment and driver training. An example of a simulator used for these purposes is the COV driving simulator of the University of Groningen [RUG-COV driving simulator 2000].

1.5 Project description

The goal of our project is to study intelligent agents controlling an autonomous vehicle and their application to traffic simulation. An agent is an autonomous computerised entity capable of sensing its environment and acting intelligently based on its perception. In this thesis, the emphasis will be on agents that perform tactical-level driving. We will distinguish two different uses for the driving agent. The agent can be seen as the first step towards the creation of a real-life intelligent vehicle. For this purpose we have designed a general model of an agent that can be used to control different types of vehicles. Second, we have equipped our agent with several possible driving styles so it can be used to model individual drivers in traffic simulators. The main focus of our research will be on agents driving simulated vehicles in an urban environment. The choice for an urban environment was made since this is one of the most difficult scenarios for a driving agent to be working in. In a city a lot of unexpected events can happen and the agent has to deal with many different situations.

1.5.1 The driving agent

The driving agent forms the intelligence and control system of an autonomous vehicle. The agent gets information via sensors embedded in the vehicle or in the agent itself and makes decisions based on this information. We have made several assumptions:

- We will assume that the required sensor and signal processing technology has reached high standards and is sophisticated enough for our application. Note that in reality this is not the case, as we will discuss in more detail in section 2.2.
  Low-level issues, such as the inner-workings of sensors or interfacing to computers and the conversion and sampling of sensor data, will not be covered in this report.

- Since well-working navigation systems already exist, we will assume that the driving agent gets its orders from either such a system or a special planning and navigation agent.

- The agent does not rely explicitly on communication with other vehicles or agents, because it should be capable of functioning autonomously.

Since it is very difficult to develop a system that works perfectly and can deal with all possible circumstances, it should be reasonably easy to expand or adjust the agent. This way, adaptations can be made and more functionality can be added to the agent later on without the need for completely redesigning it. Also, the agent must be able to function without any special modifications to its environment. With environment we mean the surroundings of the agent, excluding the intelligent vehicle it controls.

1.5.2 Simulation

The adaptability and flexibility of intelligent agents makes them ideal for modelling different types of vehicles and driving styles in microscopic traffic simulations. In this project we will study the use of agents as drivers in a micro-simulator. Part of this will be to equip the agent with a variety of behaviours and driving styles.
The designed agent will be tested in a simulator. Since our study on existing simulators did not reveal any usable simulator (they were either not suitable or not publicly available), the decision was made to create a simple prototype simulator program to show the possibilities of the driving agent.
The different phases done in this project are summarised in Table 2.

Table 2: General overview of the different phases of the project

1. Study and analysis of agent technology and autonomous vehicles
2. Design of a general tactical-level driving agent model
3. Study and analysis of traffic micro-simulators
4. Design of dynamic agent behaviour
5. Design of a traffic simulator prototype
6. Implementation of the simulator prototype program
7. Implementation of the driving agent (embedded in the simulator program)
8. Testing the implementation
9. Evaluation of the results

1.6 Report structure

The structure of this report is as follows. Chapter 2 shows how an intelligent vehicle works and gives some examples of intelligent vehicles that are used today. Chapter 3 and 4 discusses the concept of intelligent agents and multi-agent systems. Chapter 5 continues our discussion about intelligent agents with several design and implementation techniques.
The design of our intelligent driving agent is presented in chapter 6 and chapter 7 shows the changes we made to this design in order to create different types of agent drivers that can be used in traffic simulations. Chapter 8 discusses the implementation of the simulation program that was used to test the agent, as well as the implementation of the driving agent itself. In chapter 9 we will discuss the results of our investigation and in chapter 10 some conclusions and recommendations are presented.

PART I:

THEORETIC BACKGROUND

Chapter 2: Intelligent vehicles

This chapter describes what intelligent vehicles are and presents some example applications. After an introduction in section 2.1, section 2.2 briefly explains some of the sensor technology used in intelligent vehicles. Section 2.3 and 2.4 discus two different forms of intelligence in vehicles that are applied today. In the last section, section 2.5, some examples are given of new applications that are still under study.

2.1 Introduction

The field of Intelligent Transport Systems (ITS) deals with systems that apply information, communication, and control technology to improve the operation of transport networks. Part of ITS are intelligent vehicles, vehicles equipped with sensors, a control system, and actuators. Intelligent vehicles are capable of performing (a part of) the driving task. The sensors process information about the vehicle's environment and send data to the control system. The control system contains the intelligence and reasoning needed for the task and determines the appropriate action for the current situation. This action is then sent to the vehicle's actuators, or to the user in the form of a message containing relevant information (see also Figure 1).

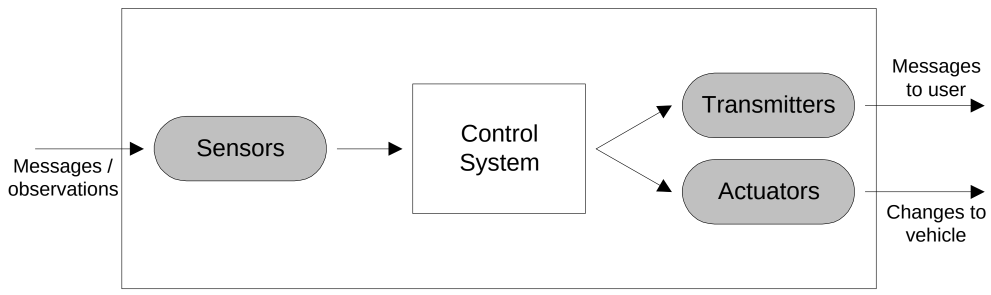


Figure 1: Basic layout of an intelligent vehicle

Although there is a lot of interest in the application of intelligent vehicles, in practice they are rarely used. This has to do with several problems, for example:

- There is a *lack of sufficiently fast and reliable control systems*. Humans have the ability to understand the behaviour of dynamic entities observed around them and can usually predict their impact on the driving task with little effort. Automating this reasoning process has turned out to be very complicated due to the large amount of possible circumstances that need to be programmed and the short amount of time available for determining the correct action.
- The necessary *sensor technology is not as advanced* as we would like. Image processing and recognition are still too difficult and slow for real-time usage. It is hard to determine the type or exact position of an object, especially under difficult conditions such as intense sunshine, rain, or snow.
- *Making constraints or changes* to the working environment can partially compensate the lack of good sensor technology. Often autonomous vehicles use magnetic strips or transponders in the road to enhance navigation. The disadvantage is that these adaptations are not always feasible and make the introduction of intelligent vehicles on the public road a very costly process.
- *Safety* is a very important issue. Companies are reluctantly to introduce intelligent vehicles quickly onto the market because the technology has yet to prove itself. The public will not accept any accidents with intelligent vehicles and therefore they cannot be applied until extensive testing is done.

- Many people dislike the idea of being out of control, so *public acceptance* has to change before intelligent vehicles can be introduced on a large scale. This is also the main reason that most trains still use human drivers, although technically it is possible to do without.
- *Legislation* may need to be changed in order to determine liability in case of an accident involving intelligent vehicles. It is impossible to produce software that is bug-free, so responsibility issues need to be regulated. Also, it needs to be clear what requirements must be met to be allowed to drive an intelligent vehicle on the public road.

Due to the problems mentioned above, experts in the field expect that autonomous vehicles will probably not be applied on the public road for at least two decades [TRAIL 1999]. Currently, several subtasks of driving are automated by intelligent driver support systems, a topic that will be discussed later in this chapter. Most experts assume that the continuing development of driver support systems will be evolutionary, meaning that the automation of vehicle driving tasks will gradually increase in time.

2.2 Sensors

A sensor is a primary converter that transforms information about the subject of measurement to an electronic signal. The job of a sensor system is to recognise different objects, pinpoint their exact position relative to the intelligent vehicle, and possibly determine several other important properties of the object, such as its speed, size and orientation. Besides information about other objects, the intelligent vehicle requires data about its own condition, which is also gathered via sensors. Examples are speed, acceleration, wheel angle, and fuel level.

2.2.1 Object recognition

Object recognition is the task of determining the position and nature of objects perceived by a camera. Object recognition has been used to detect road lines, cars, trucks, pedestrians and road signs [Tzomakas 1999].
After a camera has taken a picture, the first step in the object-recognition process is to determine the position of all interesting objects and separate them from the image background. This is called localisation or segmentation. The second step is the actual identification of the objects and deals with the assignment of a label or class to each of the segmented objects. Each object is compared to a model of known objects and the pose of the object in the image is determined.
Processing only one picture at a time is called intra-frame processing. To reduce errors inter-frame processing is used to compare the results of the object-recognition process on multiple subsequent pictures. Inter-frame processing also allows object tracking and can be used to determine the speed of moving objects.
A problem with object recognition is that it is a very difficult process due to problems with illumination variances, weather conditions, partial occlusions and the motion of the camera viewpoint.

2.2.2 Range detection

Besides camera’s, other sensors like sonar or laser-range finders are used to measure the distance and angle to nearby obstacles. The sonar or laser sends out a signal that is reflected back by an object. The distance to the object can be measured based on the time between sending and receiving a signal. The angle to the object is determined by checking which signals are returned and which are not as is demonstrated in Figure 2.

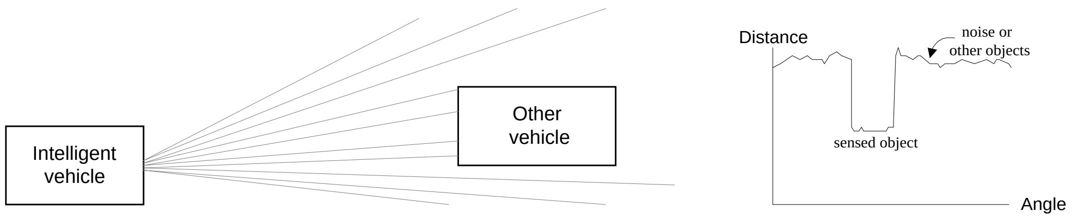


Figure 2: Range sensors can be used to return the distance and angle of an object

Although it is possible to measure the time delay between sending and receiving signals accurately, it is very hard to produce reliable and precise data. Signals will only be reflected back from surfaces that are at approximately right angles to the signal beam. Objects with flat surfaces and sharp edges reflect very little in most directions (stealth aircraft work this way). After being reflected back from a surface, the beam may strike another surface and then be reflected back to the sensor. In this case the time delay will not correspond to a physical object, but to a ‘ghost’ object [Russell and Norvig 1995]. Due to these problems sonar and laser-range finders can only be used in short proximity. Currently operational adaptive cruise control systems have a sensor range in the order of 100-150 metres, which is much less than human observation in good visibility conditions.

2.2.3 Positioning

Every wheeled vehicle can be equipped with wheel encoders that measure the revolution of its wheels. Based on this measurement, odometry can provide an estimate of the vehicle’s location that is very accurate when expressed relative to the vehicle’s previous location. This localisation technique is called ‘deadreckoning’. Unfortunately, because of slippage as the vehicle moves, the position error from wheel motion accumulates. To increase positional sensing one can use the Global Positioning System (GPS) that can measure one’s position anywhere on the earth with a precision of approximately 3 metres. Differential GPS uses a beacon with an exactly known position in combination the GPS system and can increase this precision.

2.3 Driver support systems

Driver support systems help the driver by giving information, or by partly or entirely taking over certain driving tasks. The range of driver support systems varies from purely informative systems to non-overrulable and completely interventionist systems. An example of a driver support system that gives information is the collision-warning system that is installed in certain types of cars and busses. This system alarms the driver when it is about to hit something while parking or backing up. Other used information systems are route navigation and traffic-information systems.

A driver support system that can take over a driving task is the Adaptive Cruise Control (ACC), also called Intelligent Cruise Control. An ACC functions like a normal cruise control but is also capable of maintaining a certain distance to the vehicle ahead. This is done by slowing down the car to prevent exceeding the set headway. The ACC is currently for sale in some luxury passenger cars (e.g. BMW 700 series).

A third example of a driver support system is the Intelligent Speed Adapter (ISA) that automatically reduces the speed of a vehicle if it is driving above the local speed limit. Unlike the previously mentioned systems the ISA requires communication between the vehicle and suitable information sources. These sources can be either traffic signs emitting signals or an onboard map-based information system in combination with the Global Positioning System (GPS). The ISA is not operational yet, but recently experiments have been done to test the ISA in the Netherlands.

2.4 Autonomous guided vehicles

The most commonly used vehicles that exhibit some intelligence are Automated Guided Vehicles (AGVs), also referred to as autonomous or automatic guided vehicles. AGVs were first designed in the 1950s. The first AGV was a converted tractor that followed a guide wire on the ceiling of a factory. Later, this wire was moved from the ceiling into the floor. While the first wire-guided AGVs used induction to follow an electromagnetic field produced by the wires, the next generation of AGVs used an optical system to track a line painted or taped on the floor. However, the drawback of these electromagnetic and optical systems is their inflexibility. Vehicles have to stay on their route or they become lost. Although the inductive guide wire is probably still the most used guiding system today, recently AGVs have been equipped with more advanced navigation technology such as laser scanners, microwave transponders, ultrasonic sensors, and camera vision systems. With these advances a new generation of AGVs is being developed that is able to navigate and drive around freely.
The most common application of AGVs today is in factory automation, transporting goods and materials. An example is the Europe Combined Terminal (ECT) in the port of Rotterdam that uses 54 automatic stocking cranes and 120 AGVs to automatically transport containers to and from ships. The ECT AGVs use odometry and magnetic identification points in the ground for navigation. The FROG company that has developed the ECT AGVs, has also built the ParkShuttle AGV shown in Figure 3. The ParkShuttle is used for transporting people at Schiphol Airport and the Rivium area near Rotterdam [FROG 2000].

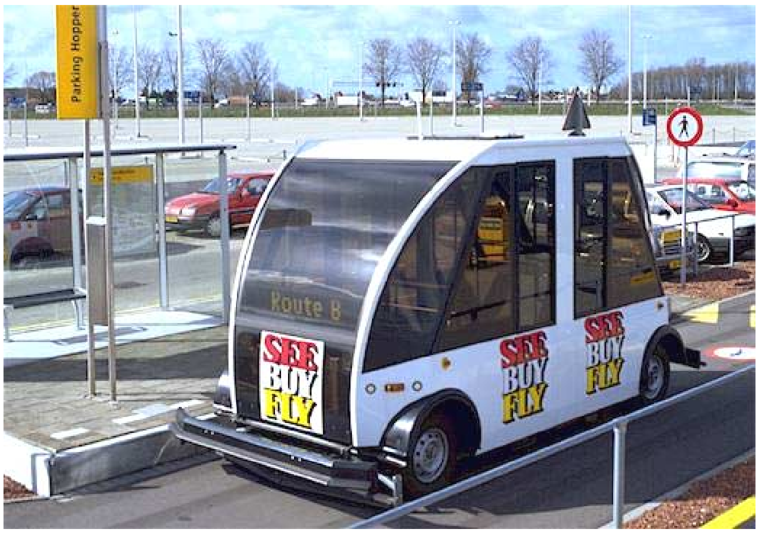


Figure 3: The ParkShuttle AGV at Schiphol airport

Recently, the Connekt Research Centre for Traffic and Transportation in Delft has started the SMAGIC-project (**SM**arter **A**utomated **G**uidance of AGVs to **I**ncrease **C**apacity) in order to increase the intelligence and efficiency of the ECT AGVs [Connekt 2000]. Besides this, Connekt is also working on a project called OLS, a Dutch abbreviation for Underground Logistic System, that deals with AGVs transporting flowers underground.

2.5 Research and new techniques

Looking at fully autonomous vehicles, almost all existing techniques and applications are still in the research phase. Current sensor technology has enabled us to create vision-based lane trackers to follow roads with clear and visible lining, and research in automatic headway control and convoying has led to vehicles capable of autonomous car following. However, putting these and other techniques together into a single application still needs to be done.
For the most part, research focuses on the application of vehicles on highways mainly for two reasons. First, highways form a relatively easy and predictable environment. They consist of

clearly separated lanes going in the same direction, have exits and entrance lanes, and signs for speed limits and directions. Also, the chance of unexpected events happening on highways is a lot smaller than in for example an urban environment. The second reason is that the gain from vehicle automation on highways is expected to be larger than from other environments. In the Netherlands about 50% of all travelled kilometres is done on highways [TRAIL 1999].

In the last few years the focus in research and development has changed towards short-term and easy-to-develop techniques instead of full automation. The belief of the early years that fully autonomous vehicles would be realised within a relatively short period has turned out to be false and expectations regarding the contribution of driver support systems have become less high. Three examples of intelligent vehicles research projects that are currently done are the Automated Highway System, ALVINN, and the SAVE project.

2.5.1 Automated Highway System

The Automated Highway System (AHS) is a concept where large numbers of intelligent vehicles can be used to improve highway capacity and reduce traffic accidents. All vehicles enter the AHS under human control. After a successful ‘check-in’ procedure, the intelligent vehicle switches to computer control. While on the AHS, several vehicles travel together at high speeds with reduced headways until they reach their destination. At this point the human driver regains control of the vehicle.

There are different competing designs for the AHS [Ioannou 1997]. Some support the idea of a dedicated- lane AHS with all vehicles on that lane being computer controlled, while others propose that autonomous and human-controlled vehicles should share the road. With the first concept, the intelligent vehicles can rely on protocols that are established using communication between the vehicle, the dedicated lanes, and other smart vehicles. To increase capacity the vehicles drive closely together like a train or convoy. The second approach has the advantage that the smart vehicles can be applied on today’s roads without the need for special modifications. The drawback is that the intelligent vehicles must be able to competently drive in uncontrolled traffic just as human drivers do, something that is not yet technologically possible.

In 1997 a demonstration of a prototype dedicated-lane AHS was given on a test track in San Diego and about two years ago a similar demonstration was held in Zouterwoude in the Netherlands. It is still unclear whether or not the AHS will actually increase capacity and improve traffic flow. Estimates are that lane capacity will be increased by a factor two or three [Minderhoud 1999], but opponents argue that the check-in and check-out procedure cannot be done fast enough and will cause more congestion.

2.5.2 ALVINN

The Autonomous Land Vehicle In a Neural Network (ALVINN) system uses artificial neural networks for autonomous robot navigation and is a project of Dean Pomerleau at Carnegie Mellon University [Pomerleau 1993]. An artificial neural network is a simplified and computerised model of a brain and consists of inter-connected elements called neurons that are capable of learning. The basic network architecture used with ALVINN is a single hidden-layer feedforward neural network, with 960 input neurons, 5 hidden-layer neurons and 30 output neurons. The output layer is a linear representation of the direction the vehicle should travel to keep the vehicle on the road or to prevent it from colliding with nearby obstacles. The centremost output unit represents the ‘travel straight ahead’ condition, while units to the left and right of centre represent sharper left and right turns. ALVINN is trained with images from a camera mounted on a driving vehicle and the corresponding actions taken by a human driver. Once trained, ALVINN can drive autonomously at a speed of up to 55 mph in environments varying from multilane paved roads to off road environments.

In a cooperating Carnegie Mellon University (CMU) project called ‘No hands across America’ two researchers let a computer system steer about 95% of the time and drove from

Pittsburgh to San Diego, more than 2800 miles [CMU Robotics Institute 2000]. It must be said that these figures are a bit misleading since all the vehicle did was follow the road. The researchers handled the throttle and brake.

2.5.3 SAVE project

The '**S**ystem for effective **A**ssessment of the driver state and **V**ehicle control in **E**mergency situations' (SAVE) is a working prototype system, embedded in a vehicle, that can temporarily take over the controls [SAVE 2000]. The SAVE system is designed especially for situations in which the driver gets unwell, falls asleep or is otherwise incapable of safely controlling the vehicle. SAVE is not equipped to make long-term decisions on its own. The system constantly monitors whether the driver is ok. If not, it will first alert the driver that something is wrong and then, after no response is detected from the driver, it will steer the car safely to the side of the road and stop.

The dominant view in literature is that of a gradual change of the present car systems, based on "*incremental absorption and slowly intensifying use of new technologies*" [TRAIL 1999]. We already see the introduction of smart but still separate systems like intelligent cruise control, collision avoidance systems and navigation systems. The next step will be to combine individual longitudinal systems into one system and after that a combination of both longitudinal and lateral control systems will follow. Once this is complete, dedicated lanes like the AHS proposal in section 2.5.1 can be introduced. Expectations are that this process will take somewhere between 20 and 40 years. The introduction of fully autonomous vehicles capable of driving outside dedicated lanes will take at least 10 more years.

Chapter 3: Intelligent agents

The term ‘agent’ has become a buzzword in the computer industry, especially on the Internet, and many software programs are given the name ‘agent’. After a short historic background on agents in section 3.1, section 3.2 discusses what agents really are. Section 3.3 describes the most commonly found agent types and in section 3.4 three agent classifications are presented. A short overview of several applications is given in section 3.5. The chapter is closed with a discussion about the advantages and disadvantages of agents in section 3.6.

3.1 Historic developments

In the mid-1950s, J. McCarthy and G. Selfridge developed the first ideas about what we now call intelligent agents. They proposed a flexible system that, when given a goal, could carry out the details of the appropriate computer operations and when it was stuck, could ask for and receive advice in human terms. The system would be a ‘soft robot’ living and doing its work within the computer. In the 1970s Carl Hewitt refined this idea and proposed a self-contained, interactive and concurrently executing object, which he termed ‘actor’. An actor “*is a computational agent which has a mail address and a behaviour. Actors communicate by message-passing and carry out their actions concurrently*″ [Hewitt 1977]. The actor object had some encapsulated internal state and could respond to messages from other similar objects.

Distributed Artificial Intelligence

Distributed Artificial Intelligence (DAI) is the study of distribution and coordination of knowledge and actions in environments with multiple entities. DAI can be divided into two subfields:

- *Distributed Problem Solving* (DPS) that solves problems by dividing work among a number of cooperating and knowledge sharing modules or nodes.
- *Multi-Agent Systems* (MAS), the study of coordination of a group of autonomous intelligent agents.

In the late 1970s Multi-Agent Systems (MAS) emerged as a subfield of DAI. In these days research on MAS focused mainly on deliberative agents that use symbolic reasoning models and the interaction, cooperation and coordination between these agents. It was not until 1990 that researchers began to investigate other methods and possibilities of agents, resulting in an increasingly wider range of agent types. The growth in Internet use and the advances in artificial intelligence are responsible for the large amount of attention that intelligent agents have received in the last couple of years. Agents are becoming more sophisticated and many people on the Internet are using agents either directly (e.g. programs like BargainFinder or Gossip) or indirectly (e.g. search engines that use agents for indexing). Expectations are that more and more people are going to use agents in the future.

3.2 What exactly is an agent?

An agent is a computerised entity that can act on behalf of someone to carry out a particular task that has been assigned to it. Obviously, we would like to use agents with as little effort as possible and since it is tedious to spell out every detail, we would like our agents to be able to understand what we mean from what we tell them in a natural fashion. Agents can only do this if they ‘know’ something about the context of the request. A good agent would not only need to have this particular expertise, but would also take into account the peculiarities of the user and the specific situation. In addition, we expect agents to be able to communicate with other agents and cooperate with them, and perhaps move from place to place in doing so. Most agents that exist today are sophisticated software programs and only 'live' in cyberspace, but some agents exist in the physical world, for example a robot.

3.2.1 Agent definition

As with many concepts in the field of artificial intelligence the term 'agent' is difficult to define. There are probably just as many definitions as there are researchers working on the subject and the fact that the word 'agent' is used (and misused) to refer to a wide range of computational entities, does not help either. Several definitions of well-respected researchers working on agents are:

"*An intelligent agent is a computer system situated in some environment, and that is capable of flexible autonomous action in this environment in order to meet its design objectives.*"
[Jennings and Wooldridge 1998]

"*An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitment.*" [Shoham 1997]

"*An agent is a program that is, to some degree capable of initiating actions, forming its own goals, constructing plans of action, communicating with other agents, and responding appropriately to events –all without being directly controlled by a human. (…) The second meaning of agent is connected with what is portrayed to the user. Here, 'agent' is used to describe programs which appear to have the characteristics of an animate being, often a human.*"
[Bradshaw 1997]

"*When we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user. Given a choice, we would rather say it is an umbrella term, meta-term or class, which covers a range of other more specific agent types.*" [Nwana 1996]

Most definitions refer to several aspects commonly found in agents: autonomy, adaptivity and the ability to do a certain task. It is very hard to tell what qualities a program or entity must have to be classified as an agent. Shoham [1997] probably phrased it best by stating "*what makes any hardware or software component an agent is precisely the fact that one has chosen to analyse and control it in these mental terms*". With this Shoham means that a program is an agent if one has decided to look at it as an agent.
Not only are there many different definitions of agents, researchers in the field also use several names and synonyms to distinguish different kinds of agents. Some examples are personal agents, personal assistants, autonomous agents, bots, softbots, robots, spiders, and crawlers. Generally speaking, agents that operate in the real world are called robots, and agents operating on the Internet are referred to as bots, spiders or crawlers. Agents that help a user to perform a particular task within a program or act on behalf of the user are called personal agents or personal assistants.
Comparisons have been made between an agent and an object in object-oriented programming languages. The main difference between the two is that an agent is considered to be a higher level concept. The actual implementation details of an agent can be hidden and neglected. They are much more sophisticated than objects in the sense that they are autonomous and are thought of as intelligent.

3.2.2 Agent properties

Another way of defining an agent is by describing its properties. There are several characteristics that the ideal agent must possess in order to be truly intelligent [Jennings and Wooldridge 1998], [Nwana 1996]:

- *Responsive (or reactive)*: an agent must be able to perceive its environment and be capable of performing actions based on its perceptions.
- *Autonomous*: the ability to perform actions without the constant guidance from the user. The agent must determine how a given problem is solved best and act on its own initiative.
- *Adaptive*: although not all agents are capable of adapting their behaviour, adaptation is generally regarded as an important characteristic for intelligent agents. The agent should learn the user's preferences and be able to improve from experience.
- *Social*: agents should be able to interact and cooperate with other agents or humans in order to carry out complex tasks they cannot handle alone and help others with their activities.
- *Communicative*: agents should be capable of communicating with other agents or humans using an expressive communication language. Ideally, this language should resemble human-like speech acts.

Several other but generally regarded as less important properties are [Jennings and Wooldridge 1998], [Nwana 1996], [Russell and Norvig 1995]:

- *Mobility*: the ability of an agent to move from one system to another to access remote resources or meet other agents.
- *Personality*: the capability of showing a 'believable' human-like character such as emotion.
- *Inferential capability*: an agent can act on abstract task specifications using prior knowledge of general goals. This goes beyond the given information and the agent may have explicit models of itself, the user, the situation and/or other agents.
- *Proactive*: agents should not simply act in response to their environment, but be able to perform opportunistic, goal-directed behaviour and take the initiative where appropriate. Instead of waiting to be told what to do next, they should make suggestions to the user.
- *Individual world view*: every agent has its own model of the external world.
- *Reproduction*: if agents are able to reproduce themselves, they can evolve and adapt to changing situations, increasing their survival skills. This is one of the main research topics of artificial life.

Most of today's software agents are designed for special-purpose applications and no agent can do very much of what is outlined above. Although several agents on the Internet might be able to pass the Turing test[2], the general consensus in the artificial intelligence community is that really intelligent agents have not yet been created.

3.3 Agent types

Different types of agents have been created based on the different properties that an agent should possess. Most agent types are constructed with one or several properties in mind.

Deliberative agents

A deliberative or cognitive agent is an agent that possesses an explicitly represented symbolic model of the world and makes decisions via a symbolic reasoning process. The agent knows how to perform actions and can predict their outcome based on its knowledge about the dynamics of the world and how it evolves.

Deliberative agents are said to be intentional, i.e. they have goals and explicit plans to reach those goals. They can memorise situations, analyse them, foresee possible reactions of their actions and use this to decide what actions to perform. They use high-level instructions and

[2] See Appendix B for explanation.

can often negotiate with other agents to resolve conflicts. The disadvantage of deliberative agents is that their reasoning process takes quite some time, so they cannot be used for problems that need a real-time response. On the other hand, their intelligence and reasoning is often superior to that of other agent types.

Reactive agents

Reactive agents, also called reflex or behaviour-based agents, react by stimulus-response rules to the current state of the environment perceived through their sensors. They have no representation or symbolic model of their environment and are incapable of foreseeing what is going to happen. The main advantage of reactive agents is that they are robust and have a fast response time, but the fact that purely reactive agents do not have any memory is a severe limitation. Also, the simplicity and functionality of their behaviour is a sign of the intelligence of its designers, rather than of the agent. On the other hand, very complex patterns of behaviours can emerge from the interaction of multiple reactive agents.

Deliberative and reactive agents are two extremes, but many agents can be classified somewhere in between, using a combination of both approaches (see also Figure 4).

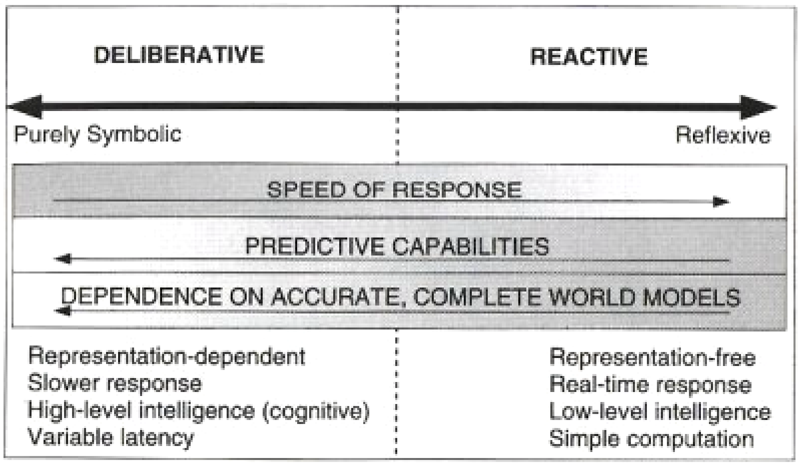


Figure 4: Control system spectrum from [Arkin 1998]

Interface (or personal) agents

Interface agents cooperate with the user to accomplish a task and function as a personal assistant. Despite the name, the interface agent is usually not the interface between the user and a computer program. Instead, it observes the interaction between the user and the program from the sideline, learns from it, and interacts with both the user and the program.

Some interface agents are designed to train or teach the user, others can help multiple users to work together. Most interface agents are capable of learning and adapting to the interests, habits and preferences of the user. Pattie Maes, who has designed an e-mail assistant, has identified four different ways of learning for interface agents [Maes 1994]:

- Observing and imitating the user.
- Receiving positive or negative feedback from the user based on the agent's performed actions.
- Receiving explicit instructions from the user.
- Asking other agents (with the same task) for advice.

An example of a (rather annoying) interface agent is the paperclip or office assistant found in Microsoft Office programs.

Mobile agents

Agents that can migrate from one machine to another are called mobile agents. They are able to work in a platform-independent environment. Mobile agents are very popular for distributed information retrieval and telecommunication network routing since they can reduce the need for constant communication between a central computer and the remote systems.

Information (or Internet) agents

The information agent is probably the most popular type of agent. Many information agents can be found on the Internet. They are designed to manage the explosive growth of information and provide the user with the requested data. They can be mobile, searching the Internet for specific information, or static, for example filtering and sorting e-mail. Other examples of information agents on the Internet are the comparison-shopping agents that can compare the prices of products sold by online stores. Also most search-engines use mobile agents called spiders or crawlers to search for and index web pages.

Viruses

Computer viruses can be regarded as a special type of agent. They work autonomously, are able to migrate from one computer to the other, and can multiply themselves. The goal of most viruses is to spread around and infect as many computer systems as possible, often altering or deleting computer data in the process.

Heterogeneous (or hybrid) agents

Combining at least two different agent types leads to a heterogeneous or hybrid agent system. Heterogeneous agents need a common agent communication language and protocol to communicate and cooperate. More of this will be discussed in the next chapter.

3.4 Agent classifications

The wide range of agents that have been developed makes it very difficult to categorise all agents and as with the definition of an agent there is no classification that researchers have agreed upon. One approach is to look at three dimensions or axes that can be used to measure an agent's capabilities; agency, intelligence, and mobility as is demonstrated in Figure 5 [Gilbert et al 1995].

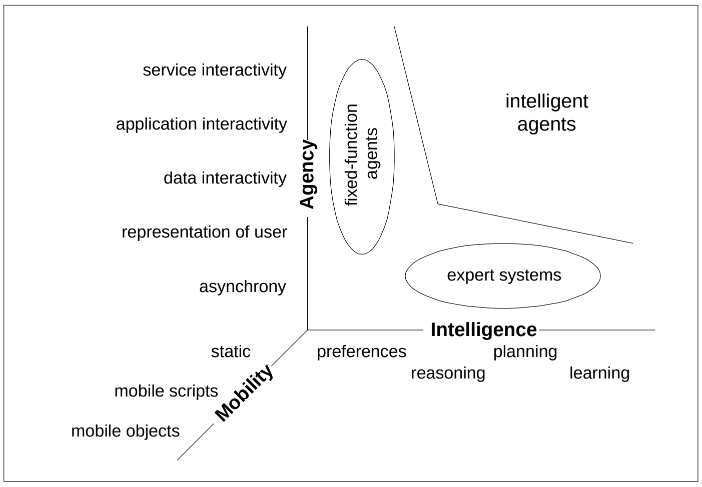


Figure 5: Gilbert's scope of intelligent agents defined by agency, intelligence and mobility

Agency refers to the degree of autonomy the agent has in representing the user to other agents, applications and computer systems. Intelligence deals with the ability of the agent to use domain-specific knowledge and its ability to solve the problem at hand. An agent is

mobile if it can move between systems in a network. Of course, more dimensions can be added but three dimensions are easier to visualise.
Franklin and Graesser [1997] tried a different classification approach. Based on the biological taxonomy of living creatures, they devised the tree structure shown in Figure 6. This three structure can be expanded easily to further categorise agents, for example the different types of viruses.

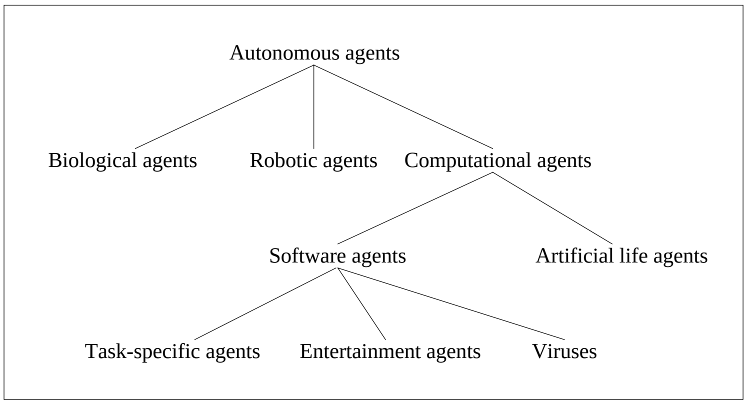


Figure 6: Franklin and Graesser's agent taxonomy

Another classification often found in agent literature is the one devised by Nwana [1996], that uses several dimensions to classify software agents:

- *Ideal and primary attributes*: agents should exhibit several ideal and primary attributes like autonomy, learning and cooperation.
- *Mobility*: agents are classified as either static or mobile.
- *Reasoning model*: either deliberative or reactive (see also section 3.3).
- *Role*: e.g. information searching, management, process control or Internet.
- *Hybrid*: combination of two or more of the above.
- *Secondary attributes*: e.g. versatility, benevolence, veracity, trustworthiness, temporal continuity, ability to fail gracefully, and mentalistic and emotional qualities.

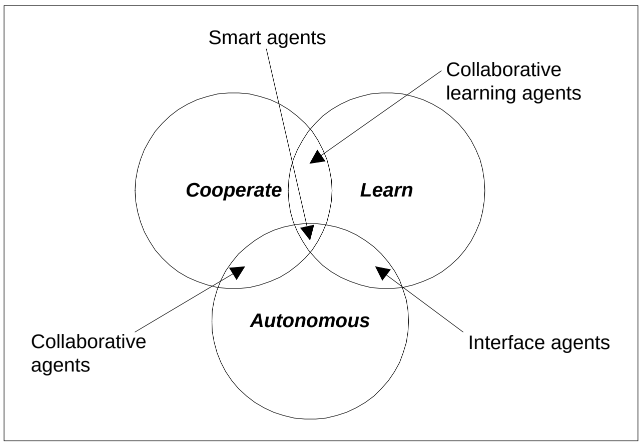


Figure 7: Part view of a Nwana's agent typology based on ideal and primary attributes

Note that the distinctions between the agents in Figure 7 are not strict. For example, with collaborative agents there is more emphasis on cooperation and autonomy than on learning but we do not imply that collaborative agents never learn. Likewise, for interface agents, there is more emphasis on autonomy and learning than on cooperation. Ideally, agents should do all three equally well, but this is the objective rather than the reality.

3.5 Agent applications

Most applications of agents fall in one of two categories: simplifying distributed computing or acting as intelligent resource managers. The first is often used when the problem that needs to be solved is distributed. In this case agents are able to solve local problems and communicate to other agents in other locations or move through a network. In the second case, agents are used to automate certain processes and/or overcome user interface problems. Current direct-manipulation user interfaces are rather static. Agents allow more flexibility by running search or filtering operations in the background, delegating tasks to other agents, 'waking up' at certain scheduled times, adapting to new situations etc. Examples of application domains that use agents are:

- *Industrial applications*
  - Process control (e.g. ARCHON agent)
  - Manufacturing (robots)
  - (Air) Traffic control

- *Commercial applications*
  - Information management (data mining, filtering or gathering information on the WWW)
  - Electronic commerce (automated trading)
  - Business process management (workflow management)
  - Medical applications (patient monitoring or information exchange between doctors)
  - Learning (helping users to learn new computer programs)

- *Entertainment* (playing with or against virtual characters in computer games)

The range of firms and universities actively working on agent technology is quite broad and the list is growing. It includes small companies (e.g. Tryllian and Verity), medium-size organisations (e.g. General Magic, MIT, Delft University of Technology) and the real big multinationals (e.g. IBM, Lotus, Microsoft). Recently, the concept of intelligent agents has received much interest by telecommunication company's like AT&T, British Telecom and KPN Research who are working to use agents in both their computer and mobile phone networks.

3.6 Advantages and disadvantages

It is clear that agents are ideal for helping people to perform a task or by taking over tasks completely. Other advantages of agents are:

- *Increased flexibility and adaptability*: agents are able to learn and adapt to particular situations. Everyone can have their own personal agent that helps as they please. The agents will adapt to the user's preference.

- *Abstraction and modularity*: agents form a good way of abstracting functionality. They correspond to the general idea that many people have of a smart autonomous computer program. Also, parts of an agent can be replaced with new functionality without changing the interface to humans. It is even possible to replace a complete agent by a similar improved agent without the need for the user to adjust.

- *Robustness:* a well-designed agent with a good reasoning system can stabilise or resolve the situation in case of errors or misunderstanding.

- *The possibility for parallelism*: apart from agents working in parallel with humans, taking over their work, agents can work together to speed up the execution of the given task.

Someone once said that "*every advantage also has its disadvantages*" and of course this is also true for intelligent agents. Although agent technology has an important role to play in the development of leading-edge computing applications, it should not be oversold. Most applications that currently use agents could be built using non-agent techniques. A number of drawbacks common to all agent-based applications are:

- *No overall system controller or global perspective*: the actions chosen by an agent are determined by that agent's local state. In many realistic agent systems complete global knowledge is not possible. This means that agents make sub-optimal decisions. An agent-based solution may not be appropriate for domains in which global constraints have to be maintained.

- *Time constraint:* the more sophisticated an agent is and the more functionality it has, the longer it takes for the agent to reach a decision. This makes it difficult for complex agents to function in environments where real-time response must be guaranteed. Also, learning agents might need a substantial amount of time to learn the appropriate behaviour, if they are able to learn their task at all.

- *User's knowledge*: people need to know how they can use agents to do certain tasks for them. This means that they need to know what the agent can and cannot do. The functionality of the agent should be clear but at the same time the agent's complexity should be hidden from the user in order not to confuse him or her too much.

- *Trust and delegation*: for individuals to be comfortable with the idea of delegating tasks to agents, they must first trust them. Both individuals and organisations need to become more accustomed and confident with the notion of autonomous software components, if they are to become widely used. Users have to gain confidence in the agents that work on their behalf.

- *Legislation and liability:* should a user be held (partly) responsible for his or her agent's actions? Who is liable for malfunctioning agents? These are matters that are not resolved yet.

Chapter 4: Multi-agent systems

Multi-agent systems are formed when multiple agents are combined to perform a certain task. Section 4.1 shows why this approach can be helpful. The next two sections, section 4.2 and 4.3, describe how agents can work together to solve their task. Usually, this requires some kind of inter-agent communication and this is the subject of section 4.4.

4.1 Introduction

Multi-Agent Systems (MASs) are systems in which two or more agents interact with each other to accomplish a task, usually through cooperation and coordination of their actions. The MASs approach is part of the field of Distributed Artificial Intelligence, but is largely influenced by other fields like robotics, psychology, ethology, biology and artificial life. Research on MASs focuses on mechanisms of self-organisation and the creation of agents that can accomplish complex tasks by working together.

Why use multi-agent systems?

The concept of intelligent agents is not as revolutionary as it might seem at first glance. All the used techniques, like information filtering, planning, autonomous behaviour and adaptive control have already been used successfully in various applications. The difference with agents is that they combine all these techniques and form a new and more intuitive way of creating smart applications. The idea is that one and one is not two, but could be three or even more. This is called synergy; the sum of the parts is larger than the whole. As Marvin Minsky described in his book 'The society of the Mind' [Minsky 1986] intelligence is not found in some central processor, but in the collective behaviour of a large group of specialised and interconnected entities. Even with a group of very simple agents, highly complex intelligence can emerge as the result of interaction between agents, coordination, and careful selected behaviours.
There are many possible applications of MASs but the most important ones fall into one of four categories:

1. *Simulation*: agents can be used to directly model the actions and behaviours of individuals in a simulated environment. These simulations are often called microscopic simulations, micro-simulations, or micro-worlds to denote that the used representations are at the level of the individual.

2. *Collective robotics*: a group of robots can cooperate in accomplishing their task and individual robots need to coordinate their actions in order not to bump into each other. An example of a collective-robot MAS is a team of robots playing soccer and working together to score goals against another team of robots. Many universities around the world have created such a team and play against each other in 'Robocup' games, an attempt to promote AI and intelligent robotics research [Robocup 2000].

3. *Program design*: MASs can be used in computer programs, which are capable of interaction, adaptation, and possibly even reproduction, in order to provide more flexibility to their users. An example is cooperating interface agents, negotiating with other agents on behalf of the user.

4. *Distributed problem solving*: this involves all situations in which agents are used to solve a problem other than those mentioned above. Both the problem and the expertise of the agents can be distributed. Homogeneous agents can be applied to solve a distributed problem (e.g. sending e-mail) or specialised heterogeneous agents can solve a central problem (e.g. robots creating an industrial product).

4.2 Interaction

Interaction occurs when two or more agents form a dynamic relationship through influential actions [Ferber 1999]. Agents can interact through a series of events, during which they are in contact with each other in some way. This contact can be direct, via another agent or indirect through a mutual environment. Direct interaction and interaction via other agents usually occurs by sending messages. When interaction takes place through changes in a mutual environment no communication is necessary.
Interaction situations can be classified in relation to three criteria; the objectives or intentions of the agents (compatible or incompatible goals), the relationships of these agents to the available resources (e.g. space, time, energy) and the skills available to the agents to achieve their goals. The eight possible situations are shown in Table 3 below [Ferber 1999].

Table 3: Types of interaction

Goals	Resources	Skills	Types of possible situations
Compatible	Sufficient	Sufficient	Independence
Compatible	Sufficient	Insufficient	Simple collaboration (e.g. communication)
Compatible	Insufficient	Sufficient	Obstruction (e.g. traffic jams)
Compatible	Insufficient	Insufficient	Coordinated collaboration (e.g. industrial activities)
Incompatible	Sufficient	Sufficient	Individual competition (racing)
Incompatible	Sufficient	Insufficient	Collective competition (grouping, team competitions)
Incompatible	Insufficient	Sufficient	Individual conflicts over resources
Incompatible	Insufficient	Insufficient	Collective conflicts over resources (companies, war)

4.3 Cooperation and coordination

If we look at Table 3, the most promising and useful situation would be a MAS in which the agents are able to cooperate. Cooperation can improve agent performance, resolve conflicts and increase survival capacity. One can look at cooperation from the agent's own perspective. In this case we can say that an agent is cooperative if it has a compatible goal and wants to help out other agents. The agent has the 'intention' of cooperating. But what if the agent, despite its good intentions, only gets in the way of the other agents? Another way of looking at cooperation is from an external observer's point of view. Then agents are cooperative if we can observe cooperative behaviour.
Once a group of agents decide to cooperate, coordination problems may arise. The question is which agent has to do what, using what resources? Agents may need information and results, which only other agents can supply, and resources can be limited. Coordination is a form of cooperative behaviour and consists of all the extra actions that need to be done in order to manage a group of agents. It is not directly productive but serves to improve the way in which activities are carried out. The easiest way to resolve coordination problems is to break down the task at hand into subtasks and distribute the subtasks over the agents. This task allocation technique can be performed in several ways:

- ***Centralised allocation:*** specifically designed for coordination with one central 'leader'. The disadvantage is that cooperating agents have to know each other, but it is fairly easy to implement. Centralised allocation can be separated into two sorts:

  - *Imposed allocation via hierarchical structure:* one agent gives orders to its subordinates.
  - *Allocation by trader or broker:* a special agent knows the capabilities of all other agents and mediates between them.

- ***Distributed allocation:*** in this case, given an open set of agents, the possibility to coordinate and cooperate exists but it is not obligatory. Each agent individually tries to obtain the services it needs from other agents. As with centralised allocation, distributed allocation can be separated into two types:

  - *Allocation by acquaintances:* agents know all other agents and their capabilities, and send requests to other agents to perform the task.
  - *Allocation by contract net:* bidding requests are sent to all agents and the agent that makes the best offer is allowed to perform the task.

- ***Emergent:*** emergent cooperation is characteristic for reactive systems. Each agent is designed to perform a particular task and usually no negotiation during the job is necessary. An example is reactive agents that use potential fields, imaginary attracting or repelling fields, to avoid other agents [Arkin 1999].

Not all tasks or subtasks can be done by one agent alone, so task allocation is not always sufficient. Some tasks require more sophisticated methods of coordination, such as planning, synchronisation (sequencing of actions) and agent-regulations (set of rules of behaviour for all agents).

4.4 Communication

One of the most important aspects of multi-agent systems is communication. If agents do not communicate they cannot tell other agents about their intentions. A good communication system allows agents to communicate effectively and exchange information despite possible differences in hardware platforms, operating systems, programming languages, and reasoning systems. Communication can be either synchronous or asynchronous. In the first case the agent sends a message and waits until it receives a reply. With the latter the agent sends a message and continues his work without waiting for an answer. Communication can be implemented either as message passing or using shared variables. Message passing is considered a better approach because it is simpler to implement and no synchronisation methods like semaphores are needed.
There is a trade-off between computation (agents maintaining models of the situation) and communication costs. The more an agent stores in its memory, the more local computation is required to maintain a world model and reason about it, but the lower the communication overheads.
Many different agent communication protocols exist, but the most important ones are KQML, FIPA's ACL and KIF. The relationship between them is shown in Figure 8.

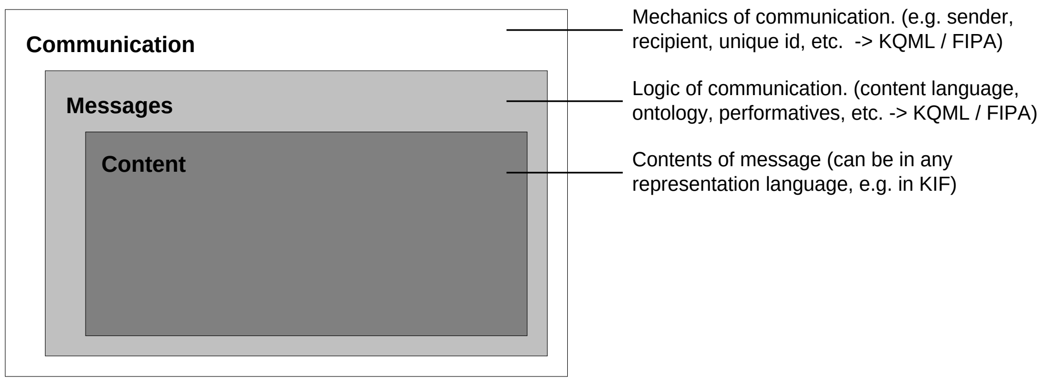

Communication
Messages
Content
Mechanics of communication. (e.g. sender, recipient, unique id, etc. -> KQML / FIPA)
Logic of communication. (content language, ontology, performatives, etc. -> KQML / FIPA)
Contents of message (can be in any representation language, e.g. in KIF)

Figure 8: Composition of an agent message

4.4.1 Knowledge Query and Manipulation Language (KQML)

KQML, which stands for Knowledge Query and Manipulation Language, is a language and protocol for exchanging information and knowledge between software agents. KQML has been developed as part of the DARPA Knowledge Sharing Effort[3], a consortium that is "*aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable*" [Finin et al. 1994].

KQML defines both the message format and the message-handling protocol to support run-time knowledge sharing. It is not an interpreted or compiled language but a set of rules that can be used in communicating. KQML messages, which are called performatives, express attitudes regarding the content of the information exchange and allow agents to find other agents that are able to process their requests. Performatives are combined to form KQML exchanges (conversations). The content field of each performative is not dictated by KQML and can contain any kind of data. All KQML implementations ignore the content field of the message, except that they need to recognise where it begins and where it ends.

Although there is a predefined set of KQML performatives, agents may choose to process only a few performatives or use other performatives of their own. Reserved performatives must be implemented in the standard way. Specifications for the full set of reserved KQML performatives can be found in [Labrou and Finin 1997a]. The following is an example of a KQML message:

```
(ask-if   :sender           A
          :receiver         B
          :language         prolog
          :ontology         traffic-rules
          :reply-with       id1
          :content          "trafficlight(red)" )
```

The first word of the KQML message is the name of the performative. In this '`ask-if`' performative agent A is querying agent B in the Prolog language about the truth value of '`trafficlight(red)`'. It is up to agent B to interpret the content of the message and evaluate the query. The word '`:ontology`' is used to provide additional information about the domain related to the content of the message. Any responses to this KQML message will be identified with '`id1`'.

Early experiments with KQML started in 1990 and current KQML implementations use standard communication and messaging protocols as a transport layer, for example TCP/IP, SMTP, HTTP and CORBA. Although there have been attempts to draw up the semantics of KQML [Labrou and Finin 1997b], they are not well-defined and it is quite common that different people use the same performative with a different meaning [Skarmeas 1999].

4.4.2 FIPA's Agent Communication Language (ACL)

The FIPA Agent Communication Language (ACL) is the European counterpart of KQML. The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit association that "*develops specifications of generic agent technologies to provide a high level of interoperability across applications*". The target of FIPA is to create 'Intelligent Physical Agents', devices intended for the mass market, capable of executing actions to accomplish goals set by or in collaboration with human beings or other intelligent agents with a high degree of intelligence [FIPA 1997].

The FIPA organisation has produced a specification of an agent communication language that looks similar to KQML but with much better defined semantics. The exact specifications can be found on the FIPA homepage on the Internet [FIPA 2000].

[3] The terms 'ARPA' and 'DARPA' (Defense Advanced Research Projects Agency) are both used in literature and denote to the same organization. Also the terms 'Knowledge Sharing Initiative' and 'Knowledge Sharing Effort' refer to the same consortium.

4.4.3 Knowledge Interchange Format (KIF)

The Knowledge Interchange Format (KIF) is a formal language for the interchange of knowledge between different computer programs. Based on first order logic, KIF was specifically designed to act as a mediator or interlingua in the translation of other languages. Translation occurs by transforming statements from language A to KIF and from KIF to language B. KIF can also be used as a common language between two agents that use different native languages or to describe programs or scripts for agents to follow. It is not meant to be a programming language or internal knowledge representation of an agent, but can be used as such. Given the prefix syntax of KIF, it looks similar to LISP, Scheme or CLIPS. An example of a KIF sentence is:

```
(defrelation bachelor (?x) :=
      (and (man ?x) (not (married ?x))))
```

The sentence above says that variable x is a bachelor, if x is a man and x is not married.
As with KQML, KIF is part of the DARPA Knowledge Sharing Effort, but unlike KQML, the KIF language description includes a specification for both the syntax and the semantics. The complete specification of KIF can be found in [Gensereth and Fikes 1992].

Chapter 5: Building agents

The first section briefly outlines the general agent construction process. Section 5.2 and 5.3 cover two important aspects of agents, namely the knowledge representation language and the knowledge base. Section 5.4 shortly addresses agent programming. Finally, the last section describes several existing agent architectures that can be used as a guideline or template for constructing agents.

5.1 The construction process

Much literature has been written about the construction of agents, but little has been said about the general design process. Most papers and books deal with the construction of specific agents or agent types. Naturally, the followed course of design depends very much on the type of agent and its application, but in any case some guidelines can be given. Generally speaking, every agent can be constructed by going through the following phases:

1. *Analysing the problem domain*: identify the significant objects and features of the application domain. What are the percepts that the agent can encounter and what kind of actions can be done to influence the environment?
2. *Defining the agent's requirements:* what goals or performance measures is the agent supposed to achieve? How should it respond in certain situations?
3. *Choice of knowledge representation language and reasoning model:* based on the domain knowledge that needs to be modelled a choice is made for a knowledge representation language and reasoning model.
4. *Building the (initial) knowledge base:* by storing facts and rules into a knowledge base the agent's behaviour and inner-workings are defined.
5. *Testing the agent*: just like in any other design process, the agent needs to be tested to verify that the agent works correctly and its requirements are met.

Other design or construction phases might be necessary depending on the type of agent, for instance creating a learning mechanism, designing a graphical user interface, or adding specific communication protocols. With some agents it might be necessary to 'train' them first in order to fill their knowledge base with useful information.

5.2 Knowledge representation and reasoning

Humans store knowledge in the various parts of their brain (scientists are still trying to figure out how it works exactly), but since computers or agents do not have a brain, they must use an alternative way. With agents, a computer memory is filled with certain symbols, numbers and characters that represent a certain fact or belief. Each symbol represents an object or idea and this is called 'knowledge representation'. A knowledge representation language defines the syntax and semantics for expressing knowledge. It should be capable of expressing all the necessary descriptions in an appropriate and effective way. What our agent is expected to do and in what domain will have a significant impact on the type of knowledge representation. One has to consider four important features [Tecuci 1998]:

1. *Representational adequacy:* the ability to represent all of the kinds of knowledge that are needed in a certain application.
2. *Inferential adequacy:* the ability to use the available knowledge to deduce new information (reasoning).
3. *Problem-solving efficiency:* the ability to represent and create the knowledge and procedures that are needed to solve the problem.
4. *Learning efficiency:* the ability to easily acquire and learn new information and integrate it within existing knowledge structures.

Very much related to the choice of the knowledge representation language is the reasoning system, also called inference system. An agent's reasoning system consists of data structures for storing knowledge and, more importantly, procedures for manipulating these structures and deducing new knowledge. The choice of knowledge representation dictates the type of reasoning.
The communication protocols mentioned earlier in section 4.4 are forms of knowledge representation languages. Other examples of often-used knowledge representations and reasoning systems are predicate calculus, semantic networks, frames, production systems, Bayesian networks and fuzzy systems. Many other knowledge representations exist and the interested reader is referred to [Russell and Norvig 1995] for more detailed information.

Predicate calculus

Predicate calculus is a formal logic that uses predicates on objects to define attributes and relations between objects. Two techniques, called resolution and unification, are used to process predicate statements and check whether a particular statement is true or not. Resolution is an algorithm for proving that facts are true or false by showing that the negation of the fact is not true. Unification is a technique that takes two sentences in predicate logic and tries to match and replace their variables. Together these algorithms form the basis of the programming language Prolog.
Predicate calculus scores high on representational and inferential adequacy, but low on problem-solving efficiency. It is very difficult to implement learning methods due to the complexity of predicate calculus, but the integration of new knowledge is fairly easy because of the modularity of the representation.

Semantic networks

Semantic networks are graphs in which the nodes represent objects, situations or events, and arcs between the nodes represent the relation between them. Semantic networks are well suited for representing knowledge about objects, but difficult to represent processes. Their inferential capacity is fairly high, but their learning capacity is rather low due to the difficulty of adding new nodes to existing networks.

Frames

A frame is a collection of attributes that define the state of an object and its relationship to other frames (objects). A frame contains slots that represent attributes and fillers or scripts, which are attached procedures that are called when the value of a slot changes. Frames are often linked into a hierarchy to represent the hierarchical structure of objects. In this case, frames at a certain level can inherit knowledge from higher level frames.
Frames and semantic nets are closely related, both represent objects and their relations with other objects, and both have the same advantages and disadvantages. The main difference between the two representations is their syntax.

Production systems

Production systems represent what to do in certain predetermined situations, but are less adequate for representing knowledge about objects. The IF-THEN rules used by production systems are very easy to understand and new knowledge can be added easily. This is the reason that production systems have become one of the most popular forms of declarative knowledge representations in AI. A typical production system consists of the following parts:

- *Knowledge base:* permanent memory containing IF-THEN rules.
- *Working memory:* temporary memory containing new or derived facts.
- *Inference engine or matching mechanism:* reasoning logic used to produce new data.
- *Conflict resolution mechanism:* procedure that decides which rules should be executed.

A production system works by matching the facts stored in the working memory to the IF-part of the rules stored in the knowledge base. The conflict resolution mechanism chooses the best rule of all matching rules. The THEN-part of the best rule is executed.
An example of a production system is the '**C** **L**anguage **I**ntegrated **P**roduction **S**ystem' or CLIPS for short [CLIPS 2000].

Bayesian networks

Bayes' theorem says that the conditional probability that event Y will occur given that event X already occurred can be computed, providing we know the prior probabilities that X and Y could happen, and the conditional probability that X will occur when we know that Y has already occurred. In mathematics this is described as:

$$P(Y \mid X) = P(X \mid Y) * P(Y) / P(X) \qquad \textbf{\textit{Equation 5-1}}$$

A Bayesian network, also called a belief network, causal network or probabilistic network, is a data structure that uses this theorem and represents dependence between variables. Each variable has a corresponding node with a conditional probability table defining the relationships between its parent nodes. The primary use of Bayesian networks is to use probability theory to reason with uncertainty.

Fuzzy systems

Fuzzy control systems produce actions using a set of fuzzy rules based on fuzzy logic. In conventional logic assertions about the world are either true or false, there is nothing in between. Values such as true and false (1 and 0) are referred to as crisp, that is, they have one exact meaning. Fuzzy logic allows variables to take on other values (between 0 and 1), determined by how much they belong to a particular set. In fuzzy logic these variables are referred to as linguistic variables, which have non-crisp meanings (e.g. fast, slow, far, big, etc.). Membership functions measure the degree of similarity an instance of a variable has in its associated fuzzy set. A fuzzy logic control system consists of the following:

- *Fuzzifier:* maps a set of crisp inputs to a collection of fuzzy input sets.
- *Fuzzy rule base:* contains a collection of IF-THEN rules.
- *Fuzzy inference engine:* maps fuzzy sets onto other fuzzy sets according to the rulebase and membership functions.
- *Defuzzifier:* maps fuzzy output sets onto a set of crisp output commands.

The main advantage of fuzzy systems is that they are more flexible than conventional rule-based methods and are often better understandable by humans due to the use of linguistic variables.

5.3 The knowledge base

The most difficult problem of developing an intelligent agent is the encoding of knowledge in the agent's knowledge base and the modification of this knowledge through learning or reasoning. A knowledge base is a set of representations of facts about the world. Each individual representation or fact, also called sentence, is expressed in the used knowledge representation language. Certain conclusion might be drawn from the stored knowledge by using inference. For example, if it is known that "*IF a THEN b*" and fact a is known to be true, then fact b must also be true and can be stored in the knowledge base. The rules for drawing these conclusions are also part of the knowledge base.
The process of building a knowledge base is called knowledge engineering. A good knowledge base is clear and correct. The relations and facts that matter should be defined and the irrelevant details should be ignored. The construction of an intelligent agent's knowledge base can be divided into three phases [Tecuci 1998]:

- *Systematic elicitation of expert knowledge:* the basic terminology (ontology) is developed and the conceptual structure of the knowledge base and knowledge representation language are chosen.
- *Knowledge base refinement:* the knowledge base is extended and debugged. After this phase the knowledge base should be complete and correct enough to provide solutions to most problems.
- *Knowledge base reformulation:* the knowledge base is reorganised to improve its efficiency.

The knowledge base can be created after a choice has been made which knowledge representation to use. In the early days of expert systems the knowledge engineer, someone who knows how to build a knowledge base, worked together with the domain expert and transformed the knowledge of the domain expert into a form usable by the reasoning system. The disadvantage of this approach is that the domain experts can not always formulate their knowledge exactly. In addition, many domain experts are reluctant to work with a knowledge engineer in fear of their job being taken over by the developed expert system.
A different approach can be taken with machine learning techniques. Using historical data from databases or examples generated by experts the agent is trained to perform classification and prediction tasks without going through the expensive knowledge acquisition process. Verification of the learned knowledge can be done through experimental testing on data sets different from those on which learning was performed.
Building and maintaining the knowledge base is easier if the agent can learn, but building a good learning engine is not an easy task. It is very difficult to start with a blank knowledge base and fill it with the appropriate knowledge using machine learning techniques (tabula rasa learning). A combination of hard encoded knowledge and machine learning techniques seems to be more successful.

5.4 Programming languages

Eventually, all agents have to be programmed using some sort of programming language. The most used agent-programming languages today are object-oriented languages that became popular in the 1990s. Object-oriented programming is a style of programming in which the basic units of construction are objects that are usually modelled after their real-world counterpart. An object consists of attributes and procedures that are called methods. Attributes model the specific properties of an object and methods implement its behaviour. A method is initiated by ‘sending a message’ to the object, i.e. calling the procedure. The objects are instances of classes that function as object templates.
The main advantages of object-oriented languages over more traditional imperative languages is the reusability mechanisms they provide mainly through the use of classes, inheritance of object properties and methods, and the modular style of programming and design.
There are many different sorts of programming languages and language extensions that support the creation of agent applications. However, most of these language extensions focus on the creation of mobile agents. Two examples of agent-programming languages are:

5.4.1 Telescript

Developed by General Magic, Telescript was one of the first commercial languages that could support mobile agents. A Telescript ‘place’ can accept and accommodate Telescript agents. The agent can issue a ‘go’ command that suspends its execution and resumes the next instruction after the go-statement from the new place. Agents can communicate and interact with other agents by issuing a ‘meet’ command.

5.4.2 Java

Java is an object-oriented language that has become very popular in a short time. Java is one of the most used languages for agent programming [Bigus and Bigus 1998]. It is designed as a simpler version of C++ with no pointers. It supports multi-threading, single inheritance and a very limited form of multiple inheritance via so called ‘interfaces’.
An advantage of Java is that it is portable and ‘architecture neutral’. This means that Java code is compiled into ‘bytecodes’ that can be sent over a network and executed on any target machine as long as it uses the Java Virtual Machine. An example of a Java extension that can be used for agent programming is JESS, the Java Expert System Shell, a production system in Java.

5.5 Architectures

An agent architecture can be defined as a particular methodology for building agents. It specifies how the various parts of an agent can be assembled in such a way that it can accomplish the necessary actions and how these parts should interact with each other.

5.5.1 Blackboard-based architectures

Originally designed for the HEARSAY-II speech-understanding system [Hayes-Roth 1985], the blackboard architecture is one of the most widely used architectures in symbolic multi-agent systems. The blackboard architecture is not very specific and can be seen as a ‘meta-architecture’, an architecture for implementing other architectures.
The blackboard model is based on a division into independent modules that do not communicate any data directly, but interact by sharing data in a common storage as is shown in Figure 9.

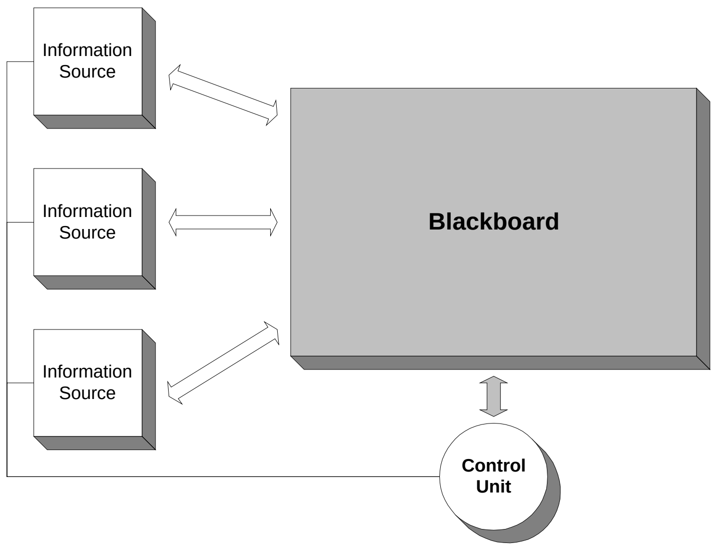


Figure 9: The blackboard model

A number of asynchronous information sources communicate with each other by placing and reading information from a shared resource called the blackboard. Only one resource at a time has access to the blackboard and a special control unit regulates the access. When an information unit wants access to the blackboard it sends a request to the control unit that schedules the request.
Blackboards form a good way of communication between heterogeneous information sources. The sources do not need to know about each other, the only thing they see is the blackboard.

5.5.2 The subsumption architecture

In the mid-1980s, Rodney Brooks developed the subsumption architecture [Brooks 1986]. Brooks argued that the traditional ‘sense-plan-act paradigm’ used by the artificial intelligence community so far, was not good enough. He believed that complex behaviour does not necessarily need a complex control system but can be the result of multiple simple rules working on the same task, a rather new idea at the time.
The subsumption architecture is a layered architecture that uses arbitration strategies and augmented finite state machines as its basis. Behaviours in the subsumption architecture are represented as separate layers with individual layers working in parallel on individual goals. The lower layers have no awareness of higher layers. Higher layers can be added on an already working control system without further modifications to the lower layers. The first behaviour of an agent might simply be to avoid objects (see also Figure 10). Another higher-level behaviour might be to move in a given direction. This behaviour will dominate the obstacle-avoidance behaviour by suppressing its output to the actuators unless an object is too close. The higher levels subsume the lower levels, hence the name of the architecture.

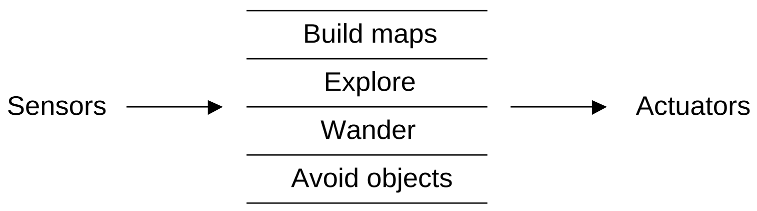


Figure 10: Subsumption behaviour decomposition of an autonomous robot

The subsumption architecture has been the foundation for the reflex agents and the behaviour-based robotics approach [Arkin 1998], [Ehlert 1999]. The largest influence of the subsumption architecture on both is the direct coupling of perception and action using simple behavioural schemes.

5.5.3 ARCHON

ARCHON is part of the European ESPRIT programme and stands for **AR**chitecture for **C**ooperative **H**eterogeneous **ON**-line systems. ARCHON is the name of the consortium of 14 organisations as well as the name of the architecture they devised. The general goal of ARCHON is to create a heterogeneous multi-agent system for industrial applications, but it focuses on a specific application, which is electricity management. Each ARCHON agent supervises and controls a part of the total system and cooperates with other ARCHON agents. The ARCHON agent consists of five components:

1. *High-level communication module*: handles the communication with other agents.
2. *Acquaintance model:* contains information about other agents (e.g. skills, interests, status).
3. *Planning and coordination:* initiates cooperation requests and deals with requests from other agents.
4. *Self-model:* contains the agent’s own state, an abstract model of the system that the agent controls, and plans on how to control it.
5. *Monitor:* provides the interface between the other components and the controlled system, checks the state, and starts or stops certain tasks within the system.

The relation between the five components is shown in Figure 11.

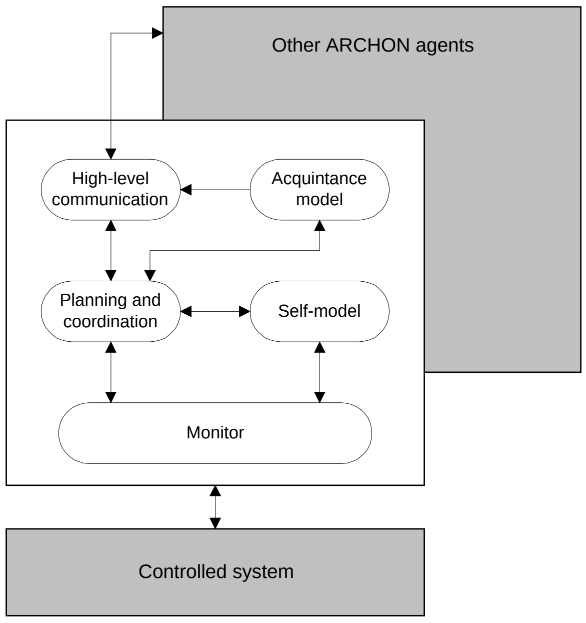


Figure 11: An ARCHON agent

5.5.4 The Belief-Desire-Intention architecture

Developed by Anand Rao and Michael Georgeff, Belief-Desire-Intention agents (BDI-agents) are characterised by a mental state with three components [Rao and Georgeff 1995]. The beliefs of an agent are its model of the world and its desires contain the goals it wants to reach. The intentions are the actions that the agent has decided to do in order to fulfil its desires, a commitment to perform a plan. If an agent receives sensor input, it will update its world model with the use of a belief revision function. The agent then generates options for actions (desires) with an option generation function. A filter function updates the agent's intentions according to its current beliefs, desires and past intentions. At the end of the cycle the action selection function chooses the action that is expected to perform the best according to the agent's current intentions.

PART II:

DESIGN AND IMPLEMENTATION

Chapter 6: The tactical-level driving agent

This chapter describes the design of an intelligent tactical-level driving agent that can be used to control an intelligent autonomous vehicle. After a short introduction in section 6.1, section 6.2 states the requirements of the agent. Section 6.3 describes the design choices that have been made and section 6.4 gives a brief overview of the design. The next six sections, section 6.5 to 6.10, describe several parts of the agent in more detail. Section 6.11 closes the chapter with some considerations about the driving agent and its design.

6.1 Introduction

As we already mentioned in chapter 1, the goal of this project is to study the use of intelligent agents that are able to control autonomous vehicles. In this chapter we present a general model of a tactical-level driving agent that can be used to control different types of vehicles. The agent makes its decisions based on the received input from its sensors and its instructions are translated to control operations that are sent to the vehicle it controls. The discussed agent model is general in the sense that we do not specify the precise driving task of the agent. We only give general rules that the agent follows while driving in an urban environment. Attention will be given to the possibility of altering the agent’s functionality without the need for redesign. The agent should be capable of controlling different types of vehicles in different environments without any design changes.

6.2 Requirements

The driving agent should be designed to perform tactical-level driving and thus needs to be able to decide in real-time what manoeuvres to perform in every situation. This real-time constraint is very important. A lot of commonly used agent techniques and architectures cannot be used in real-time applications so this restriction limits our possibilities. Several requirements of the driving agent can be stated:

- *Responsiveness:* the agent has to watch the environment constantly and react appropriately. With appropriately we mean that the agent should respond just like a normal human driver would (or preferably better).
- *Real-time constraint:* the agent should be able to respond to all situations in real-time.
- *Safety:* the safety of both the vehicle driven by the agent and the other road users should be guaranteed. Further, it should be easy to test and validate the correctness of the agent’s functionality.
- *Expandability:* it should be easy to make modifications or add new functionality to the agent without the need for redesigning the agent. This makes the application of new and future sensor technology possible. In addition, this requirement simplifies testing, since the agent can start with basic functionality and, after testing, can be expanded in an iterative manner until the agent is found sophisticated enough to perform its task.
- *Understanding of the knowledge representation:* the expandability requirement implies that the used knowledge representation language must be relatively easy to understand so the agent’s knowledge base can be altered without much effort and the effects of changes to the agent’s functionality can be studied.

Besides these five main requirements, other non-essential requirements can be stated that can improve the agent’s functionality:

- *Communication:* if the agent uses (one of the) universal agent communication languages such as KQML or the FIPA ACL, then it can communicate with other (driving) agents and respond to situations like traffic jams beforehand.
- *Learning:* the agent would be more flexible if it could learn from past experiences. In addition, programming the agent’s knowledge base will be easier after the implementation of a learning engine.

6.3 Design choices

The main difficulty of tactical driving is that good decisions need to be made in real-time. Traditional agent architectures applied in artificial intelligence use sensor information to create a world model. The world model is processed by standard artificial intelligence techniques such as search-based planning, and a plan is constructed for the agent to achieve its goal. This plan is then executed as a series of actions. Most search-based planning methods use well defined initial states and goal states, and given sufficient time, these methods can often find optimal solutions to reach the goal states.
This traditional approach has several drawbacks. First of all, sensing constraints and uncertainties cause the world model to be incomplete or possibly even incorrect, and most traditional planning methods cannot function under noisy and uncertain conditions. Second, in complex domains like tactical driving it is infeasible to plan a complete path from the initial state to the goal state, due to the large amount of searchable states and the inability to perfectly predict the outcome of all possible actions. The amount of possible states ‘explodes’ if realistic manoeuvres such as aborted lane changes and emergency braking are taken into account. As a result a real-time response cannot be guaranteed, making the traditional planning methods unsuitable for tactical driving.

Responsiveness and real-time constraint

Since the real-time constraint is so important, existing agent architectures or techniques that do not guarantee real-time response cannot be used. The approach taken in this project is a combination of traditional world modelling, the subsumption architecture [Brooks 1986] and behaviour-based robotics [Arkin 1998], [Ehlert 1999] but the emphasis is on the latter since fast response times are important. Sensor information is stored in a memory and forms a temporary world model. Reactive procedures called behaviour rules or behaviours use the available information in the memory to quickly generate multiple proposals to perform a particular action. These behaviours can work in parallel, speeding up the calculation process.
Planning in the traditional sense is not applied. A module called the short-term planner only uses simple linear extrapolation to calculate the expected positions of moving objects and an arbiter determines the best action based on the priority ratings of the action proposals included by the behaviour rules.

Safety

Unfortunately, safety can never be guaranteed completely, even by the most conservative drivers. There is always the chance of accidents happening due to some unforeseen circumstance outside the driver’s control. We limit the safety requirement by stating that the driving agent should make sure that it does not cause any accidents itself. A special Collision-detection-and-emergency-braking behaviour should prevent the agent from colliding with other objects. The Collision-detection-and-emergency-braking module is activated if the agent is about to crash into an object. The module halts the vehicle using maximum deceleration, overriding all other behaviour rules.
We have decided not equip the agent with a learning mechanism, although it should be possible to create such a mechanism encapsulated within a set of behaviour rules. The problem with learning is that it makes the implementation much more difficult and can be contradictory to the safety requirement. By giving the agent learning capabilities the agent

can exhibit unexpected newly learned behaviour, making it more difficult to guarantee correct responses in every situation.

Expandability and understanding of knowledge representation

We have tried to make the agent's design as modular as possible in order to meet the expandability and testing requirements. All steps in the agent's reasoning process are performed in separate parts of the agent that can work independent of each other. The agent's knowledge base is also separated in sets of behaviour rules that can function and be tested individually. Every set of rules can be implemented in any way the designer wants, as long as it generates its output within a certain time limit. For the driving agent we propose a fuzzy rule base or production system. A fuzzy production system is more robust than a normal production system and the linguistic variables in the fuzzy system can simplify the creation of the behaviour rules and make the created rules easier for humans to interpret and adapt.

6.4 Global layout

The design choices discussed in the previous section lead to the structure of a tactical driving agent as shown in Figure 12. The driving agent gets its orders from a planning agent or a human supervisor. Both can give instructions like move faster, turn the next road left, stop etc. Other agents might be able to provide additional information, for example local speed limits. The most important sources of information of the agent are its sensors. There are two types of sensor information available. The first gives information about the agent's surroundings, the environment, and the second gives information about the object under the agent's control, the vehicle. The communication module receives all information sent to the driving agent and all incoming messages are transferred to the agent's memory. Access to the memory is regulated by a controller that also activates the different components when necessary. Once the communication module has sent its messages to the memory, the short-term planner makes a fast prediction of the position of all moving objects in the environment.

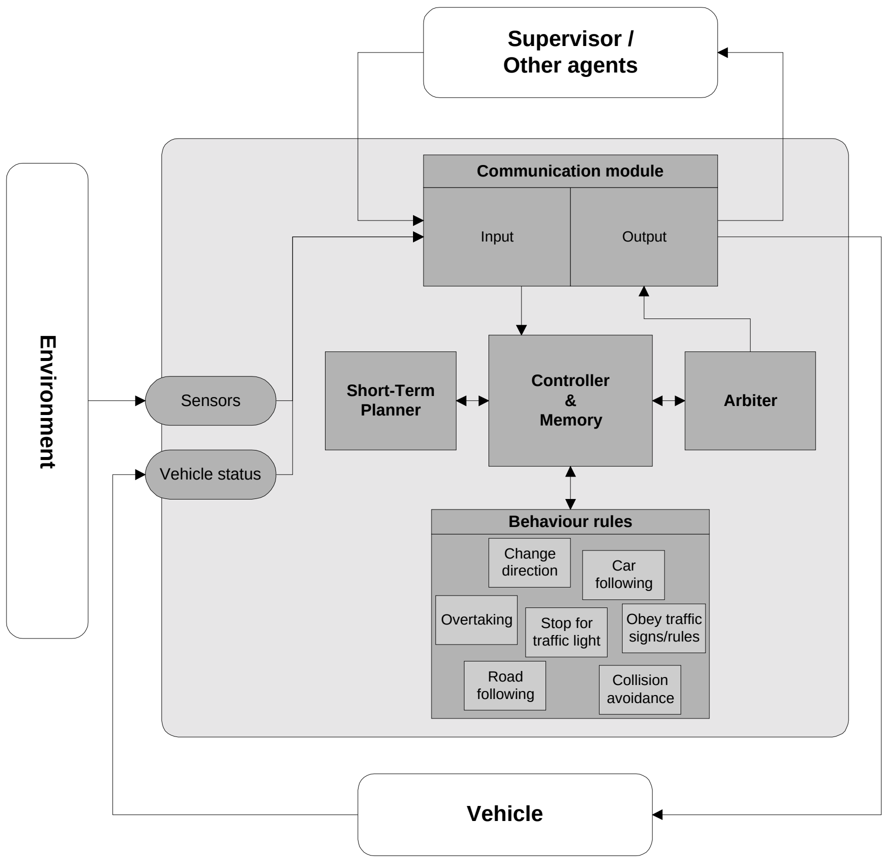


Figure 12: Layout of the tactical-level driving agent

The agent's knowledge base is divided into separate parts called behaviour rules or behaviours, specifying what to do in different situations. Each behaviour proposes an action based on the available sensor data and the prediction made by the short-term planner. All action proposals have a tally or priority rating. An arbiter selects the best behaviours based on this priority rating and sends it to the communication module. The communication module translates the commands to messages that can be understood by the appropriate receiver. Examples of commands that can be sent to the vehicle are accelerate, brake, turn left, or turn right.
The complete loop from receiving sensor messages to sending an output message can be seen as one reasoning cycle, which is timed by the controller. Since the driving agent needs to react in real-time, the agent should be able to complete several reasoning cycles per second.

6.5 Sensors

Sampling and processing sensor information is a difficult process that does not have to do much with artificial intelligence, so we will not go into details here. We will assume that all the actions needed to get the necessary sensor information for our agent are done in the sensor itself. In a sense we can regard the sensors as specialised agents that can only send messages and not receive them. We will further assume that at least the following objects and object properties can be detected or measured:

- **Roads**
  - *Lane alignment* (position and heading of the vehicle relative to the current lane)
  - *Other lanes* (existence and position of other available lanes besides the lane the vehicle is currently driving on)
  - *Curves*
    - Distance to
    - Sharpness
  - *Sideroads or intersections*
    - Distance to
    - Possible directions
  - *Distance to the end of the road*

- **Vehicles** (other than the vehicle controller by the driving agent)
  - *Distance*
  - *Relative angle*
  - *Velocity and acceleration*
  - *Relative heading*
  - *Occupied lane*

- **Traffic lights**
  - *Distance to the light and the stop line*
  - *Relative angle*
  - *Colour of the light*
  - *Lane that is regulated by the light*

- **Traffic signs**
  - *Distance*
  - *Relative angle*
  - *Type of sign*

- **Other moving objects** (all objects that can change their position have to be detected since it is possible that they might be on a collision course with the agent)
  - *Distance*
  - *Relative angle*
  - *Relative heading*

- *Velocity and acceleration*
- *Occupied lane (if available)*

- ❑ **Other objects nearby** (all objects that are close by so the vehicle might bump into them)
  - *Distance*
  - *Relative angle*

Separate sensors report the state of the vehicle controlled by the agent:

- *Speed*
- *Acceleration*
- *Wheel angle*
- *Heading*
- *Fuel level* (amount of fuel left)
- *Status* (ok, crashed, inactive, etc.)

A variety of sensors can be used in order to gather the necessary information, for example lasers, ultrasonic or microwave radar, infrared sensors and cameras. Nevertheless, the physical implementation of sensors that can detect the objects and measure the object properties that are mentioned above remains an extremely difficult task that has not been adequately solved yet. This is the main problem of building a complete functional intelligent driving agent or autonomous vehicle operating in the real world.

6.6 Communication module

The communication module contains knowledge of the used communication protocols and deals with all incoming messages from the sensors, the supervisor or other agents. The communication module is separated into two parts that can work in parallel: an input and an output part. Since the agent cannot know in advance when a message is scheduled to arrive the input part of the communication module should always be ready to accept new messages.
When a message is received, the input module tries to recognise the message format, the sender and its content. If the message cannot be understood it is ignored or a warning message is sent back to the original sender if possible. When the message is found ok, the input section temporarily stores it until it is allowed to write all the received messages to the memory. Temporary storage is necessary since one does not want data in the memory to be read and written at the same time. Outgoing messages can be sent immediately since no conflicts can arise there.
The advantage of using a specialised communication module is that the number of available communication protocols can be expanded easily. In essence, there are no limits to the amount of communication protocols that can be used by the communication module. We propose to equip the communication module at least with KQML or the FIPA ACL and with KIF (see also section 4.4) since they allow the agent to communicate with other agents in a ‘standard’ way. Of course, the agent also needs a protocol to communicate with its sensors and vehicle.

6.7 Controller and memory

The agent’s controller and memory are closely related. The controller initiates the agent’s reasoning cycles and regulates access to the memory. The combination of the controller and memory can be seen as a kind of active blackboard that requests actions or information from other processes rather than passively wait for access requests.
The controller starts a reasoning cycle by allowing the input section of the communication module to write its temporarily stored data to the memory. It makes sure that all data is written in the correct places of the memory for easy reference. Next, the controller gives the short-term planner access to this data so it can start planning the expected positions of the moving objects detected by the sensors. Then the behaviour rules are allowed to read the data

and write back their proposals for new actions. Finally, the controller activates the arbiter that selects the best action proposal and sends this to the vehicle, the human supervisor, or other agents. The length of a reasoning cycle determines the reaction time of the agent. Since on average humans need approximately three tenths of a second to react, we propose to use a similar reasoning cycle length.
For easy reference the agent's memory is separated into the following parts:

- Agenda with orders from a planning agent and/or supervisor.
- The perceived objects, divided into the following:
    - ❑ Latest sensor information.
    - ❑ Data from previous reasoning cycles (if available).
    - ❑ Expected positions of moving objects as calculated by the short-term planner.
- List of actions proposed by the behaviour rules.

The agenda is rather static and probably does not change much over the different time cycles. Data about the sensed objects is changed each cycle. New data is added and obsolete data is removed in order to reduce the amount of used memory space.
The total time needed to complete one cycle can be reduced by separating the memory into two parts that can be accessed simultaneously; the list of behaviour proposals, and the perceived objects with the agenda. After the behaviour rules are finished and have written their action proposals in the memory, the arbiter starts selecting the best proposal. At the same time the controller can give access to the communication module to write new data to the memory. This process is demonstrated in Figure 13.

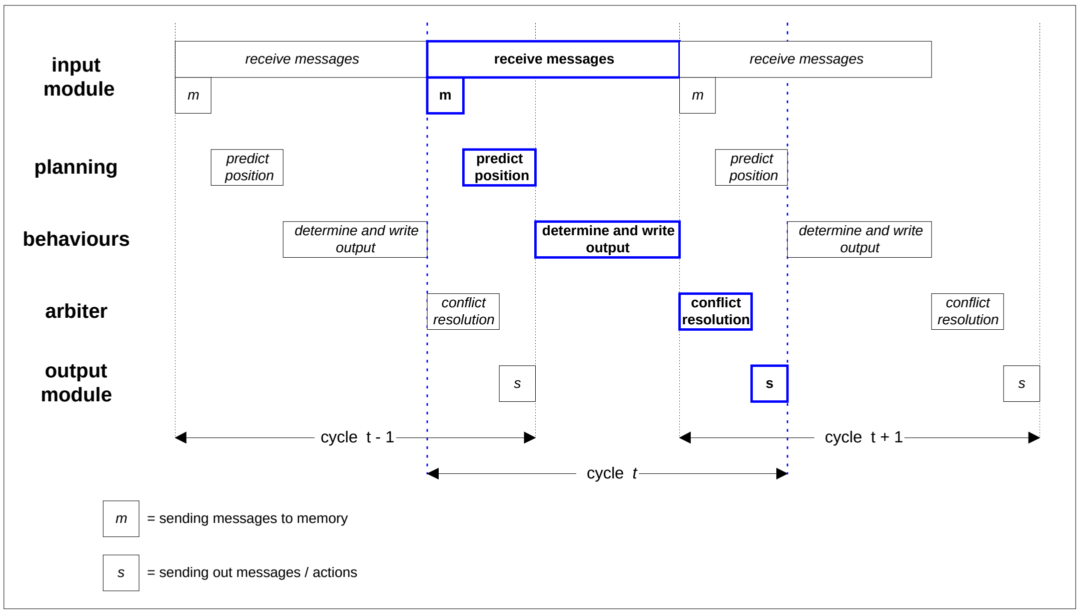


Figure 13: Reasoning cycle speed-up by overlapping cycles

6.8 The short-term planner

The short-term planner is not a complete planning system in the traditional AI sense, but tries to predict the position of the agent and other moving objects based on the current sensor data. All moving objects move with a certain speed, heading and acceleration. Based on these three factors, the position of each object can be predicted through extrapolation. For example, if a vehicle is moving in front of the agent with 10 m/s, and the agent has the same heading but a speed of 12 m/s then the agent will be 2 metres closer to the other vehicle after one second. This example is rather trivial but if the heading of the other vehicle relative to the agent is not 0 or 180 degrees then the calculation becomes a bit more complex.

6.9 Behaviour rules and actions

The driving task given to our agent is divided into relatively independent subtasks that function in a somewhat similar manner as the layers in the subsumption architecture. Each subtask is covered by a set of behaviour rules. A set of behaviour rules, also called behaviour for short, can be seen as a plan or script for different types of traffic situations. Each behaviour specifies where to look for and what to do. The strength of this approach is that behaviours can be added or changed later on without any modifications to existing behaviours. Also, parallellising behaviours will speed up the computation process.

Essentially, each specific behaviour can be implemented using whichever AI technique is natural for the task; rule-based algorithms, neural networks or potential-fields approaches. The only restriction is that each behaviour returns its output in a form that can be understood and used within the agent's system.

The precise choice and implementation of the behaviour rules depends on the task of the agent and the environment in which the agent and vehicle are placed. In this project we have chosen one of most difficult situations for a tactical driving agent to be working in, which is an urban environment. For our driving agent we propose (at least) the following behaviours:

Road following

The Road-following behaviour is responsible for keeping the vehicle driving on the road. Besides controlling the lateral position of the vehicle, based on the distance to the road and lane edges, the Road-following behaviour also influences the vehicle's speed. It makes sure that the car slows down for curves and on straight roads accelerates until the desired speed is reached.

Intersection / Changing directions

The Intersection or Changing-directions behaviour forms the link between the tactical and strategic level of driving. It looks at the orders of the supervisor or planning agent that are stored at the agent's memory. If the vehicle approaches a crossing, its speed is reduced, precedence rules are applied, and the vehicle will choose one of the sideroads according to its instructions. If no orders are present, the agent can request a direction or choose one as it sees fit.

The Changing-directions behaviour can be split up into several different (sub-)behaviours, one for each type of intersection. This is consistent with the fact that humans use different strategies to handle different types of intersection. Examples of intersection types are intersections with or without sorting lanes, T-junctions, X-junctions, and roundabouts.

Traffic lights

This behaviour searches for traffic lights, the lane to which the traffic lights belong and makes sure that the vehicle stops for red or yellow lights if possible.

Car following

The Car-following behaviour ensures that the vehicle does not bump into any other vehicles. If another car is driving in front of the agent at close range, speed is reduced to match that car's speed. The exact amount of braking depends on the speed difference between the agent's vehicle and the other vehicle, and the distance between them. The agent should make sure that this amount never crosses a certain threshold.

Overtaking and switching lanes

Related to the Car-following behaviour is the Overtaking or Switching-lanes behaviour. If a slower vehicle is in front of the agent, it may decide to overtake this vehicle. This decision depends on the difference in velocity between two vehicles and the available space to overtake the vehicle, both in front and to the left of the vehicle.

Overtaking is probably one of the most difficult behaviours to implement since the agent needs to deal with many different types of sensor information to determine if the current situation allows safe overtaking. Also, overtaking may need to be aborted if sensor information turns out to be incomplete or wrong. In this case the agent should make sure that it returns to its previous lane safely.

Applying other traffic rules

Besides traffic lights and precedence rules at junctions, other traffic rules need to be followed. Examples are, not driving at speeds above the local maximum, driving on the right side of the road as much as possible (at least in the Netherlands), no turning on one way streets.
For this behaviour it is necessary to keep track of the traffic signs and restrictions encountered on the current road. Because the memory of the agent will clear data on a regular basis to save space, the Traffic-rules behaviour needs to keep track of these signs itself, in its own private memory space. This memory space is embedded within the behaviour. Note that the behaviour also needs to keep track when the signs and other rules apply. Usually, turning onto a new road will reset most of the current restrictions.

Collision detection and emergency braking

The Collision-detection and Emergency-braking behaviour is a special kind of safety measure that is activated when the agent is bound to crash into an object. These objects can be either stationary or moving. At all times the behaviour needs to ensure that the vehicle can be halted before it hits the object. Actions from the Emergency-braking behaviour have the highest priority and overrule all other behaviours.

6.10 The arbiter

The arbiter examines all the action proposals generated by each behaviour to determine the best action. This action is then sent to the communication module and translated into operational-level commands for execution by the intelligent vehicle.
The arbiter starts examining the list of available action proposals in the agent’s memory once it is allowed to do so by the controller. Each action proposal consists of priority rating or tally, a certain lateral command, for example steer left 10 degrees, and a longitudinal command, for example accelerate with 1 m/s. It is also possible that the lateral or longitudinal command is set to ‘no action’. In this case the arbiter will ignore that particular command.
The arbiter selects the best lateral and longitudinal command based on the highest priority rating of each action proposal. The best lateral and longitudinal command do not necessarily have to be part of one action proposal. For example, suppose that the best lateral command is proposed by the Lane-switching behaviour that suggests steering right and has set its longitudinal command to ‘no action’. The best longitudinal command will then be chosen from another action proposal that is not set to ‘no action’, provided there is one.
Note that the arbiter does not have any knowledge about the task of the agent whatsoever and functions independent of the agent’s behaviour rules.

6.11 Some considerations about the design

So far, we have not addressed one important problem of designing and implementing behavioural rules, which is behavioural parameter optimisation. Behaviours depend upon a number of internal settings such as thresholds or gains. For example, at what distance from a traffic light should the agent start braking, or what speeds are acceptable when driving through a curve and how does this depend on the sharpness of the curve? Different methodologies can be used to determine correct and efficient parameters, ranging from trail-and-error parameter testing to experiments with genetic algorithms.
Since creating good and correct behaviours is already hard enough as it is, we propose an iterative manner of design, starting with one simple behaviour, refine it with trail-and-error testing, and then add more behaviours, one by one.

6.11.1 Avoiding information overload

A potential problem for driving agents used to control a real-life vehicle is information overload. A computer needs time to process all the sensor data and incoming data can pile up. Humans know instinctively what the irrelevant details are, simply close themselves off for it, and only deal with the important matters. For agents controlling real-life vehicles, extracting only the relevant information from the available sensor data is a challenging task and time constraints prevent processing all of this information. The agent must intelligently select the information most critical to the immediate task. One solution might be to make assumptions about the objects in the environment. For example, upon first observing an oncoming vehicle, the agent could note its position and velocity and then ‘forget’ about the vehicle for some time interval, knowing that it would not be able to close the distance in that time. This resembles the visual orientation strategy used by humans [Aasman 1995].

Another solution is to use active perception and activate the sensors only if a behaviour thinks it is necessary. In this case one extra step is needed in the agent’s reasoning cycle, and the controller must start by checking which sensors have to be activated.

In simulation, information overload usually is not a problem since a simulation is already an abstraction of the real world, leaving out many details and providing the agent only with relevant data.

6.11.2 Other agents

When responding to traffic control systems or other stimuli such as the presence of surrounding vehicles, a human driver’s behaviour is basically reactive. However, when planning a trip, choosing a route, time to depart, or deciding whether or not to divert in the presence of a traffic jam, drivers must possess a more complex reasoning capability. Luckily, this type of reasoning can be done in advance, before the actual trip is taken, and does not have very strict time limitations. Since drivers make these decisions based mostly on a trade-off between cost and utility, an utility-based agent [Russell and Norvig 1995], or an BDI-agent can be used as the basic structure for a driver planning agent. It is possible for multiple planning agents to work together to improve performance. Planning agents can send updates of the current road situation to each other so reactions to traffic jams can be made instantly. Also, the driving agent can use other types of agents to acquire knowledge, for example static local agents placed along the roads that track the current traffic situation.

Chapter 7: Dynamic driving behaviour

In this chapter we will discuss different types of driving behaviour and the way these behaviours can be integrated into our driving agent. Section 7.1 starts with a short overview of agents in traffic simulations. In section 7.2 different types of drivers are distinguished and in section 7.3 we will show how the existing design of our driving agent can be altered to accommodate these different driver types. Finally, in section 7.4 some other influences that can have an effect on driving behaviour are discussed.

7.1 Agents in traffic simulation

New methods produced for Intelligent Transportation Systems (ITS) are intended to improve the use of existing road capacity. These methods often try to improve this capacity by directly influencing human behavioural patterns. Therefore, modelling the uncertainty of human behaviour is essential to traffic analysis and simulation. Agent-based techniques seem to be a very suitable approach to represent human behaviour in traffic simulations. Both humans and agents are autonomous and rational beings, can interact with others, and are (usually) capable of learning. A Multi-Agent System (MAS) can be used to form the basis of a microscopic traffic simulator. Microscopic simulators or micro-simulators use interacting and autonomous entities to model a system. Drivers will definitively play a central role in the simulator and are regarded as the main components (agents) of the MAS. Other important elements are the road network and traffic control systems that dictate the movement rules of the drivers (e.g. traffic lights and signs). These environment components can also be modelled as (reactive) agents, as their tasks are often very specific and well defined.

7.1.1 Existing micro-simulators

Many different traffic-simulation programs exist, but most programs do not model vehicles as separate entities, or are not publicly available. However we were able to find a number of good micro-simulations.

A nameless multi-agent simulator

In 1992 Frank Bomarius published a report about the “multi-agent approach towards modelling traffic scenarios” [Bomarius 1992]. His idea was simply to model all the important objects as agents that could communicate the relevant data. Four years later two MSc students at the University of Edinburgh implemented this idea for their final MSc project [Chong 1996], [Chan 1996]. Their nameless text-based simulator uses Shoham’s AGENT-0 architecture [Shoham 1993] to create multiple agents that function as vehicles or traffic lights, but also as roads and crossings. As the emphasis of their project was on creating a MAS-simulation and not necessarily creating smart vehicles, all their vehicle agents use very simple rules based on gap acceptance and speed. More advanced behaviours like overtaking cannot be modelled due to the simplicity of both their agent and simulation environment.

SOAR

In his PhD. thesis Jannes Aasman describes the use of Allen Newell’s cognitive modelling tool and production system SOAR. Aasman has used SOAR to model the behaviour of a human driver that approaches an unordered intersection [Aasman 1995]. His goal is to study the cognitive load of humans while driving. A multi-tasking model is used to control a simulated car in a simulated traffic environment that contains other vehicles and cyclists. The model deals with visual orientation, speed control, lane keeping, steering curves, navigation and the necessary human limb movement to perform these tasks.

SIMONE

The **SI**mulation model for **MO**torway traffic with **NE**xt generation vehicles (SIMONE) is a microscopic traffic simulator developed to study the impact of the adaptive cruise control

(ACC) driver support system on traffic flow [Minderhoud 1999]. It simulates a motorway with vehicles equipped with and without an ACC system. SIMONE is capable of simulating individual human driving behaviour based on an individual driver's preferred speed and reaction time.

Sapient and Shiva

One of the few existing driving-simulation systems that uses tactical-level reasoning is the SAPIENT reasoning system of Rahul Sukthankar [1997]. One implementation of the SAPIENT system uses explicitly encoded rules and algorithms to perform the driving task. This MonoSAPIENT system was motivated by material developed to improve human driver performance and uses a decision tree to derive the correct action. A second and improved implementation called PolySAPIENT uses reasoning objects that specialise in a single aspect of the driving domain, similar to our driving agent's behaviours proposed in the previous chapter. Each object is associated with an observed traffic entity, such as a nearby vehicle or an upcoming exit, and examines the projected interactions of that entity on the agent's proposed actions. Both the SAPIENT systems deal with driving on highways and freeways, and were tested in a simulator called SHIVA, which stands for Simulated Highways for Intelligent Vehicle Algorithms. The SHIVA simulator was especially designed for testing different tactical-driving algorithms and allows fast creation of different test scenarios. Unfortunately the SHIVA simulator only runs on a very fast and specialised SGI machine and is not publicly available.

7.2 Driver characteristics

The objective of a driving agent controlling a vehicle in the real world is to drive as safe and efficient as possible, but the goal of an agent in a simulated world is usually somewhat different. Generally, agents in simulations are designed to imitate human driving behaviour in order to provide realistic behaviour in the simulation and thus a more accurate simulation model. However, the more realistic an agent's behaviour, the more complicated the agent will be and the more processing speed is needed. The ultimate agent, that imitates human driving behaviour as realistically as possible, uses the complete model of human reasoning and thinking with all its different aspects. Obviously this is not feasible. A balance must be found between the size and complexity of the agent and the realism of its behaviour.

7.2.1 Driver types

The primary goal of every driver is to reach his destination and avoid accidents at the same time. This goal is independent of the situation and the personal motives of the driver. The execution of the tactical-driving task however, highly depends on individual habits and motives. Each driver requires a certain time, speed, safety, economy and comfort to reach his destination. If we look at the road users we encounter on the street every day, we can roughly distinguish between three different types of drivers; the conservative driver, the aggressive driver and the average driver that can be classified in between.

The conservative driver is very careful and usually drives at speeds lower than the maximum speed limit. He prefers large gaps between other cars, pulls away slowly at traffic lights and starts braking a bit early with low deceleration. This kind of driving can often be observed from older people or people searching for the correct road.

Aggressive drivers are people that are in a hurry and quickly become agitated by other slower road users in front of them. The aggressive driver often follows closely behind other cars (tailgating), brakes forcefully and relatively late, and quickly pulls up at traffic lights. Aggressive drivers tend to take more risks and change lanes a lot more than other drivers. The more aggressive the driver, the more chance that he will break several traffic rules like running through red lights, going faster than the maximum speed limit, or overtaking where it is not allowed. This type of behaviour can be often viewed from people who are late or commuters that travel the same route day in, day out.

The average driver is a sensible driver that (almost always) follows all traffic rules, drives with a speed correlating to the legal speed limit or sometimes slightly above, and does not take many risks.

7.2.2 Important parameters

When we look at the description of the different driver types mentioned above, we can extract several parameters that have a large influence on a driver's style of driving. One of the most important (visible) factors is the driver's choice of speed. This choice has a large effect on the different driving subtasks, for example overtaking. Drivers that prefer high speeds are more likely to overtake a car than slower drivers. Another factor is the driver's used rate of acceleration or deceleration. Aggressive drivers tend to accelerate faster than slower drivers. A third factor is the distance the driver keeps to other cars, also called gap acceptance. Again aggressive drivers use smaller gaps than slower drivers.
The three parameters mentioned above, choice of speed, acceleration and deceleration rate and gap acceptance, are often related. Aggressive drivers like to drive fast, accelerate quickly and use small headways. Conservative drivers drive relatively slow and use a smaller rate of acceleration and deceleration. The aggressiveness of a driver is also related to the amount of risks a driver is willing to take and his tendency to brake certain traffic rules, for example running through a red light. A driver's aggressiveness is not constant. Normally, people are not very aggressive, but traffic jams or being cut off by other drivers can increase aggression.

7.3 Changes in the driving agent model

Creating different types of agents with different driving styles, can be done in several ways. If we take the existing tactical-level driving agent model discussed in chapter 6, we could equip each agent with its own set of (slightly different) rules, depending on the type of driver we want it to be. Unfortunately, building several different drivers this way would require a lot of work. A better solution is to give each agent the same set of rules and store a set of parameters that influence these rules (see Figure 14).

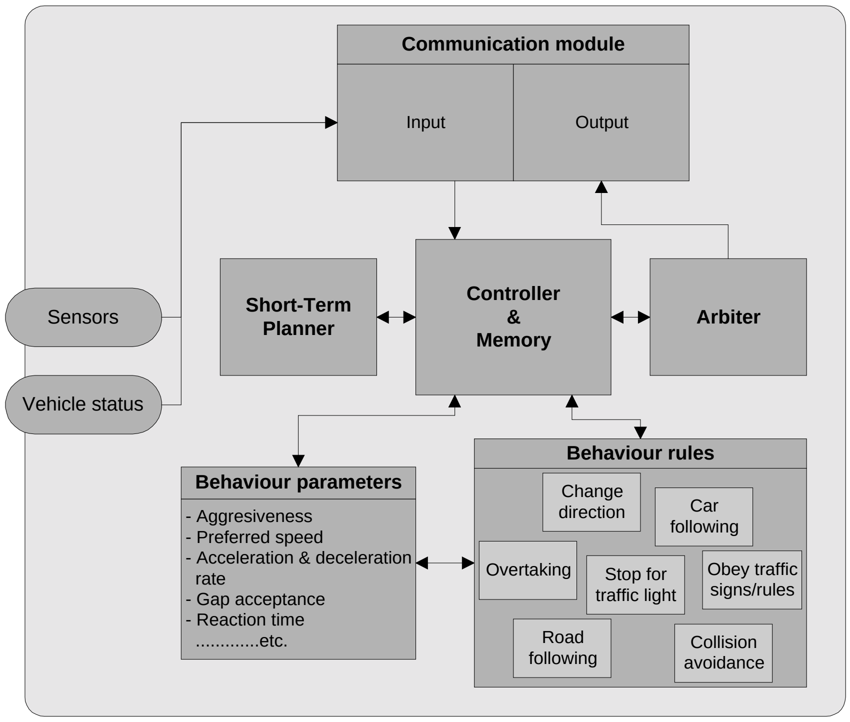


Figure 14: Tactical-level driving agent design including behaviour parameters for dynamic driving behaviour

The behaviour rules will be activated depending on a certain parameter, for example the distance to a red traffic light that triggers a braking action, or the precise amount of the response is determined based on the parameters, for example the amount of braking.
The used behavioural parameters can be either static or dynamic. The latter requires one extra step in the agent's reasoning cycle. After the new sensor data is stored in the agent's memory, the parameters module needs to determine if some event has happened that has an impact on certain parameters such as the driver's aggressiveness. Aggressive drivers can become less agitated if they have a clear road in front of them and can drive at the speed they prefer and other drivers can become more aggressive if they end up in traffic or get cut off by other vehicle's.

7.4 Other influences on the driving task

Besides the behaviour factors mentioned in the previous sections, other aspects can influence an agent's driving behaviour, for example the reaction time of the agent. An agent's reaction time can be altered by changing the length of its reasoning cycle.
Also, weather conditions can have an influence on driving. Fog or heavy rain can reduce the visibility of the driver and force him to drive more carefully. Bad visibility can be simulated by changing the sensor range of the agent, but the decrease of vehicle traction as a result of wet roads is much harder to implement.
Vehicle properties can be modelled to add more realism to a traffic simulator. The vehicle controlled by the agent can be one of several different types: a normal car, a bus, a truck etc. The type of car determines the vehicle's maximum speed, how fast it can brake and accelerate, and how fast it can make a turn. Obviously cars can drive faster, brake harder, and turn more quickly than a bus or a truck. Therefore, it is important that the agent has knowledge of capabilities of its vehicle.

Chapter 8: Driving agent simulator

This chapter describes the implementation of a prototype simulation program that was created to test the design of the driving agent discussed in chapter 6 and 7. In section 8.1 we discuss our reasons for building a new simulator instead of using an existing one. Section 8.2 describes some of the choices we made during the design of the simulator. The implementation of the simulation program is the topic of section 8.3 and section 8.4 describes the implementation of the driving agent that was embedded in the simulator. Section 8.5 and 8.6 go into more detail and describe the behaviour rules and parameter functions of the driving agent.

8.1 Why build another simulator?

The hardest part of developing an intelligent agent is identifying the rules that determine how the agent behaves. Exploring new tactical algorithms in the real world in close proximity to other vehicles is obviously unsafe and coordinating multiple moving entities to create testing scenarios is difficult and expensive. The cheapest and easiest way to create a testing environment is by using computer simulation. In a simulator the intelligent vehicle can drive around without the designer worrying about possible accidents and damage to equipment. Also simulation scenarios can be set up and altered more easily than in the real world. By creating different test scenarios the developer can first design and test simple behavioural rules and progressively add more complex behaviours to the agent. If the simulator is realistic enough, the results found with the simulator can be used to control a vehicle in the real world.

Although traffic simulators have been in use for over thirty years, hardly any simulator has all the capabilities desirable for designing intelligent vehicle algorithms. A review of existing micro-simulation models has shown that most simulators are used as a tool for the analysis of traffic flow and traffic management [SMARTEST project 1997]. Only a small percentage of all traffic simulators allow control of a vehicle at the tactical level [Sukthankar et al 1996]. Another problem with existing simulators is that most simulators concentrate on freeway or highway traffic and thus only deal with phenomena associated with these environments, such as traffic jams, stop-and-go traffic and weaving. A different situation arises in cities, where a large number of cars are headed in many different directions and more unexpected situations can arise.

The simulator we want to use must be able to simulate an urban environment with different types of vehicles, intersections, traffic lights, and possibly other objects commonly found in a city. In addition at least one vehicle, but preferably all vehicles, should be controlled by a driving agent, so there has to be an interface between the simulation program and our driving agent (or agents). Since our study on traffic simulators did not result in finding a suitable simulator (see also section 7.1.1), we decided to create a new prototype simulation program. The advantage of creating a new program is that we can create a simulator and simulator-agent interface that satisfies our needs perfectly. Also, we know exactly how the simulator and interface works. The disadvantage is that we have to start from scratch and need to spend a considerable amount of time to create the simulation program.

8.2 Implementation choices

The aim of our prototype simulation program is to provide a way for analysing and evaluating the correctness of the agent model described in the previous two chapters. This is the reason why the program focuses on monitoring the agent’s reasoning process and is not optimised for speed.

The programming language we used to build the simulation program is Borland Delphi 5 Professional for NT. We have chosen this language in part since we were already familiar with it, but mainly because Delphi is an easy language, very suitable for quick prototyping.

8.2.1 Motion model

A very important choice in the design of a traffic simulator is the choice of the motion model. The motion model determines the way simulated objects move. Inaccuracies in modelling vehicles in a simulation lead to unrealistic behavioural rules of the driving agent. Roughly, four different types of motion models can be distinguished [Sukthankar 1997].

The simplest motion model is the slot car model that divides roads in little pieces called cells or slots. Each vehicle clearly occupies (part of) a slot at all times. The result is that lane changes are instantaneous and vehicle orientations are always tangential to the local road curvature.

Kinematic motion models are a bit more advanced and deal with all aspects of motion apart from considerations of mass and force. Kinematic models implement smooth motion, even during lane changes. Furthermore, the vehicles can move along realistic trajectories. However, since forces are not modelled, the vehicles will perform manoeuvres without ever slipping.

Dynamic motion models are used in simulators where vehicles must respond realistically to forces. In 2-D simulations these forces may include tire-surface interactions and may also involve engine and transmission models. In 3D dynamic simulations the effects of gravity (e.g. driving downhill) and cornering stability is added.

Table 4: Summery of the trade-offs involved with different choices of motion models

Model	**Benefits**	**Drawbacks**
Slot car	Executes very quickly, simple to code	Unrealistic lane changes
Kinematic	Smooth motion, realistic lane changes, reasonably fast execution	Unrealistic cornering
Dynamic 2D	Allows tire models, engine models, skidding	Unrealistic collisions, slow with many vehicles
Dynamic 3D	Realistic collisions, cornering, braking and suspension	Compute-intensive, very complicated

Of course, the more realistic a simulation the more computing time is required, especially when large numbers of vehicles are involved. Weighing all the benefits and drawbacks of the four motion models in Table 4, we have chosen to use a kinematic motion model. The slot car model is too simple since a large part of the tactical driving task, such as lane and road following, is ignored. Dynamic 2D models are more realistic but are rather difficult to implement and we feel that the loss of computational speed and more difficult programming outweigh the advantages.

8.2.2 Sensors

Also for perception, one has to choose a certain level of detail. Simulators that use unrealistic perceptual assumptions are easier to program, but make the results less usable in practice. Since the simulation program is meant as a prototype we have decided to use relatively simple to code but high-level sensors in order to simplify the implementation. With high-level we mean that the sensors return information that does not have to be processed further to be usable. We have implemented sensors that use object-to-lane mapping (position of vehicle relative to lane), and road trackers (position of vehicle relative to road). Also, velocity information about moving objects is provided directly by the sensors. In reality, one would need to compare two sensor sweeps and calculate the other vehicle's speed based on the agent's own speed and the difference in position of the other vehicle between the two sweeps.

8.3 Implementation of the simulator

The simulator program roughly consists of four elements: a user interface to provide visual feedback, a simulation controller, an environment containing simulated objects, and the driving agent. In the next sections we will discuss the basic operation of these four parts but for more detailed information about the Delphi implementation, see Appendix A.

8.3.1 User interface

The interface of the simulator consists of a standard Delphi application window, shown in Figure 15. The larger part of the window shows a picture of the environment in which the agent will be driving during a simulation run. Different environments can be loaded with the ‘Select Map’-option in the Simulation menu. Environments are defined in text files with the extension .MDF, which stands for Map Data File. MDF-files contain the properties and parameters of the objects that make up the environment. Possible objects are roads, intersections, traffic lights and traffic light controllers. Vehicles can be added by clicking on a road in the environment map.

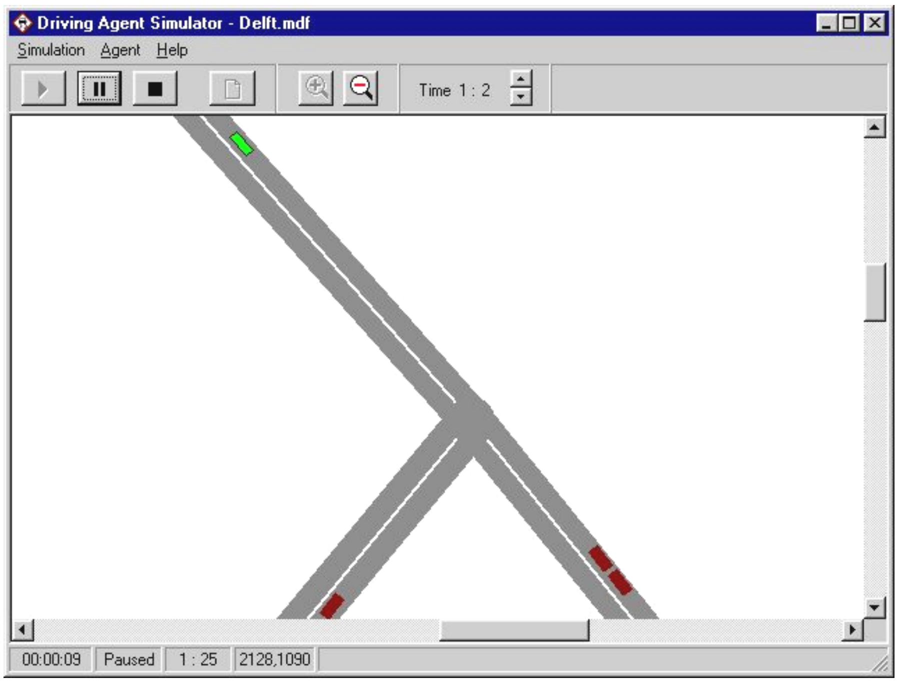
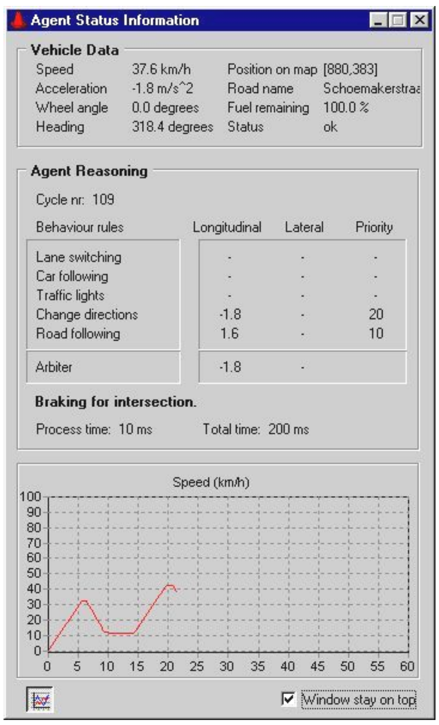


Figure 15: Main window of the Driving Agent Simulator program

Additional information about one particular agent that is controlled by the user can be shown in the agent’s Status Information window that can be activated through the Agent menu (see Figure 16). This window shows the speed, acceleration, heading, etc. of the intelligent vehicle, as well as the agent’s currently active behaviours and their action proposals.
An agent completes one reasoning cycle between every 50 to 500 milliseconds, depending on the simulation settings. Obviously this is to fast for humans to monitor, therefore the simulator can be slowed down by changing the time factor on the top of the main window or by setting the ‘Pause each cycle’ option in the menu. The latter pauses the simulation each time the agent has completed one reasoning cycle.


Agent Status Information

Vehicle Data

Speed	37.6 km/h	Position on map	[880,383]
Acceleration	-1.8 m/s^2	Road name	Schoemakerstra
Wheel angle	0.0 degrees	Fuel remaining	100.0 %
Heading	318.4 degrees	Status	ok

Agent Reasoning

Cycle nr: 109

Behaviour rules	Longitudinal	Lateral	Priority
Lane switching	-	-	-
Car following	-	-	-
Traffic lights	-	-	-
Change directions	-1.8	-	20
Road following	1.6	-	10
Arbiter	-1.8	-	

Braking for intersection.

Process time: 10 ms Total time: 200 ms

Figure 16: Agent's Status Information window

Another way to monitor the agent's reasoning process is by saving the actions of the agent in a log file. Unfortunately, one reasoning cycle adds approximately 1300 bytes to the agent's log, so running the simulator for one minute produces a file of 380 KB, given that the agent completes 5 reasoning cycles per second. To save memory and hard disk space, the agent's log file has a maximum size and data is stored on a first-in-first-out basis.

8.3.2 Simulation controller

The task of the simulation controller is to start, pause, or stop a simulation run, keep track of the elapsed time and give orders to the environment to update its data. If the start button in the main window is pressed then the simulation controller will start up all driving agents. During the simulation, the controller regularly sends an 'update' order to the environment. The environment then calculates the new values for all its objects and sends relevant visual feedback to the screen. This update process is shown in Figure 17.

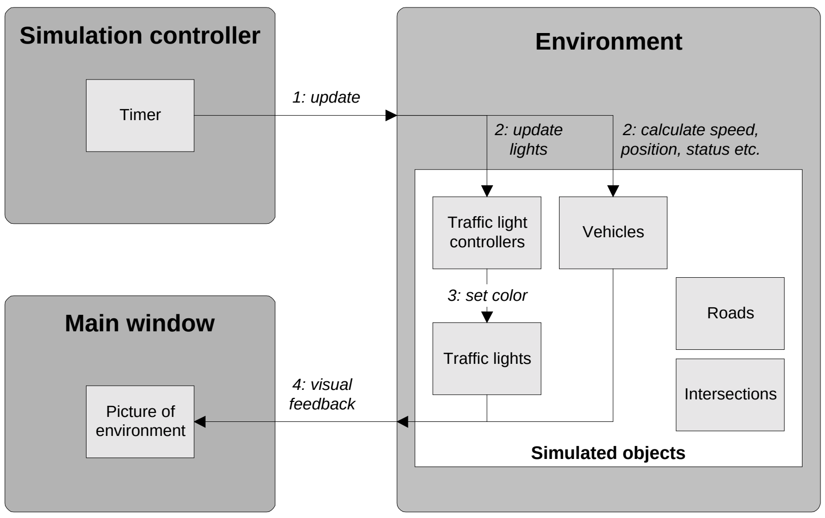


Figure 17: Simulation update loop

The simulation controller uses a timer that activates the update loop after a certain time period. By default the update frequency is about 20 times per second, but this rate can be adjusted so that the program can be run on slow computers. If the time factor of the simulation is changed, the controller frequently pauses the simulation without the user being aware of this. The result is that the simulation appears to be running slower.

8.3.3 Environment

All simulated objects together, such as vehicles, roads and traffic lights, form the simulation environment. Loading a map data file initialises the environment and the objects described in the file are stored in the environment. The environment keeps track of the position and status of all objects during simulation and updates the objects if it is ordered to do so by the simulation controller. The current implementation of the environment contains roads, intersections, traffic lights, traffic light controllers and vehicles.

Roads

Roads are static 2D objects, represented in the environment as rectangles. They have a start coordinate and an end coordinate that determine the line between the right side and the left side of a road. Each side has a certain width and contains one or more lanes. An exception is one-way streets that have no lanes on one side. Roads can connect to each other at intersections.

Intersections and curves

An intersection is a connection between two or more roads. Intersections are represented by a circle with a certain radius. The middle of the circle is the position coordinate of the intersection. Each intersection contains a list of connecting roads. Either the start coordinate or the end coordinate of each road must be within the intersection boundaries.
A curve is a special type of intersection. Curves connect exactly two roads to each other.

Traffic lights and traffic light controllers

Traffic lights are defined by their position, status, road, roadside and lane number. The status of a traffic light is simply the colour of its light. The road, roadside and lane number refer to a certain lane of a road that is controlled by the light.
A traffic light controller is a virtual object, not shown in the environment map, that regulates the colour of a group of traffic lights, for example all lights positioned around an intersection. Based on a certain cycle time the traffic light controller regularly changes the colour of the lights under its supervision. In the current implementation only one light at a time can be set to green.

Vehicles

Contrary to all previously discussed environment objects, vehicles are added at run-time. Once the simulation program is started and an environment is loaded, the user can create a vehicle by clicking on a road that is shown in the main window of the simulation program.
Vehicles are represented as rectangles with a certain random length and width. Specific properties of vehicles are their position, speed, acceleration, heading, wheel angle, status and type. If a vehicle hits another object, for example a traffic light, the vehicle's status is set to 'crashed' and the vehicle is not allowed to move again during the rest of the simulation run. If a vehicle hits another vehicle both vehicles will crash. Each vehicle is controlled by an individual driving agent.

8.4 Implementation of the driving agent

The implementation of the driving agent is done based on the design discussed in chapter 6 and 7. Each agent is made up of the following components: sensors, the communication module, a memory, behaviours and behavioural parameters, and an arbiter. Note that the planner proposed in chapter 6 is not implemented. We did not use much of the information

provided by the planner in our implementation, and the information that was needed is calculated within the behaviour rules.
Each vehicle in the environment has its own driving agent. However, one agent in the simulator has the focus of attention and can be 'controlled' by the user. This means that the user can change the settings of the agent's behaviour parameters and can follow its reasoning process in the Status Information window.

8.4.1 Threads

Each agent is implemented as a separate thread in the simulation program, running with a priority a bit lower than the simulation's main thread. The scheduling and execution of the threads depend on the number of processors available as well as the used operating system. Not all operating systems support true multithreaded-processing, even if multiple processors are available. For example, Windows 95 and NT only simulate multithreaded-processing using a scheduling routine called time slicing. Since the simulator program was constructed to run on Windows NT4 this means that all actions done by the simulation controller, running in the program's main thread, will be scheduled in front of the other agent threads because its priority is higher.
The advantages of using threads is that the program can be faster, running threads in parallel if the operating system allows it, and that the agents can run autonomous of the program's main thread. The disadvantage is that there's a limit to the number of threads one can use, because the overhead in managing multiple threads can impact the program's performance. The Delphi 5 help-file recommends a limit of 16 threads, assuming that most threads are waiting for external events. However, running the simulator on a Pentium III 450 MHz with 64 MB RAM, we were able to run up to 30 agents without any problem, so this number might be a bit low.
The first step in the execution loop of an agent is to request data from its sensors. After the agent has finished processing its data, it sends the new orders to the vehicle (see also Figure 18). If the agent finishes his reasoning cycle within a certain time limit, denoted as the agent's 'cycle time' its thread is put asleep for the rest of the cycle time. This is done to prevent agents from using all available CPU time. As a beneficial side effect, it also reduces the amount of information the agent receives. By default the agent's cycle time is 200 ms, so the agents will perform 5 reasoning cycles per second. If an agent does not finish it's reasoning cycle within the set cycle time, it will still complete the reasoning process, but will not sleep afterwards. In this case, either the agent's cycle time is too short, or the screen update rate is to high resulting in too little processing time for the agents.

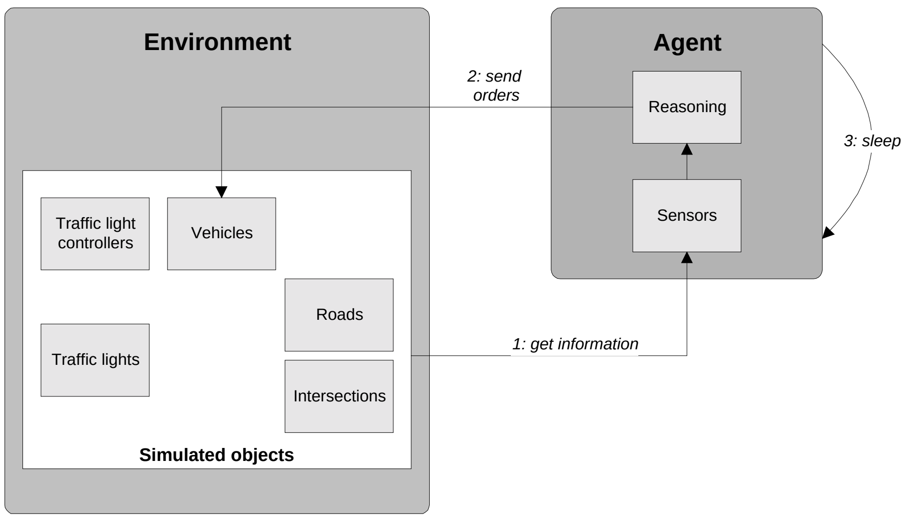


Figure 18: Agent update loop

8.4.2 Simulator-Agent interface

We have chosen to integrate the driving agent into the simulation. The reason is that this very much simplifies the interface between the agent and the simulator. The interface consists of several messages (procedure calls) that can be sent by the agent. If the agent wants to send new orders to its vehicle, it can choose between two different messages: one for changing the vehicle’s acceleration and one for changing the vehicle’s wheel angle. The agent is not allowed to make any other changes to the environment. Receiving sensor data is done by sending a request-sensor-data message. Once this message is sent, the agent’s sensors start polling the environment for the appropriate data and store it in new messages. All sensor messages are then sent to the agent.

8.5 Behaviours

In this section we will address the driving agent’s implemented behaviours. Initially we wanted to run the behaviours in parallel and implement them as threads. However, since we also wanted to reduce the number of threads in order to run more agents (threads) we decided not to do this and execute the behaviours in sequence. The precise execution order does not matter because the arbiter has to wait until all behaviours are finished determining their output.
The actual implementation of all behaviours was done one at a time, testing a new behaviour thoroughly before adding another. The behaviours that were implemented are, ordered from low to high priority: Road following, Change directions, Traffic lights, Car following and Switching lanes. A short description of the implemented behaviours and tasks is given below, but more details can be found in Appendix A.2.

Road following

The Road-following behaviour is divided into the following tasks:

- *Drive at preferred speed*: adjusts the agent’s speed based on its preference.
- *Stay in lane:* makes course adjustments if the agent’s vehicle is about to cross its lane boundaries.
- *Adjust speed for curve:* lowers the agent’s speed based on its preferred speed and the radius (sharpness) of a curve.
- *Take curve:* changes the agent’s course so that it makes the correct turn.
- *Brake for end of road*: brakes if there’s a dead end.
- *Off road:* if the agent happens to accidentally drive off the road, this task will change the vehicle’s course so that the agent will drive back to the nearest road.

When the Road-following behaviour is activated, a new action proposal is created containing a lateral and a longitudinal command. Initially both commands are set to ‘no action’, but each task is allowed to change this. Running the tasks in the order mentioned above, the result is that a task can override the changes made by another task. For example, if the Drive-at-preferred-speed task proposes to accelerate, the Adjust-speed-for-curve task may override this by changing the proposal to a deceleration command.

Traffic lights

The Traffic-lights behaviour first searches through the sensor data if any traffic lights are detected. If this is the case the next two tasks are executed:

- *Deal with yellow light*: determines if the agent can brake in time for a yellow traffic light and changes its speed based on the outcome.
- *Stop for red light*: stops the agent, unless for some reason it is impossible to stand still in time. In this case the agent will run through the red light.

Change directions

The Change-directions behaviour consists of the following three tasks:

- *Adjust speed for intersection:* reduces the agent's speed if necessary so it does not approach an intersection too fast.
- *Cross intersection*: before each intersection the agent chooses a random direction and this task changes the course of the agent's vehicle to this direction at the right time.
- *Turn around at dead end*: if the agent is close to a dead end, this task will turn its vehicle around so the agent can be on it's way again.

Car following

The first step of the Car-following behaviour is to find the closest vehicle in front of the agent. If one is found, the following two tasks are executed:

- *Match speed:* adjusts the agent's speed so that it matches that of the other vehicle, but only if the agent is driving faster.
- *Keep distance:* makes sure that a certain distance remains between the agent and the other vehicle. The exact distance depends on the speed of both vehicles.

Switching lanes

The Switching-lanes behaviour is separated into two tasks:

- *Switch to right lane:* if the agent is not driving on the most right lane of the road and there is enough space available on that lane, the agent will switch lanes.
- *Overtake vehicle:* if a vehicle is in front of the agent and that vehicle is driving much slower than the agent's preferred speed, this task will override the car-following behaviour and overtake the vehicle if possible.

We have tried to keep the behaviours and behaviour tasks as independent of each other as possible, but unfortunately in some cases dependency cannot be avoided. For example, if the agent decides to brake for a traffic light, you do not want to switch lanes while doing so. This means, that the Switching-lanes behaviour needs to know the output of the Traffic-Lights behaviour.

8.6 Dynamic behaviours

Most behaviours use certain parameters in order to reach a decision. For example, the precise amount of acceleration and deceleration is determined according to a specific function that is dependent on the agent's preferences (type of driver) and the linguistically desired amount. Linguistic amounts are: none, very little, little, normal, much, very much and maximum. The implemented acceleration and deceleration function for three different driver types is shown in Figure 19.

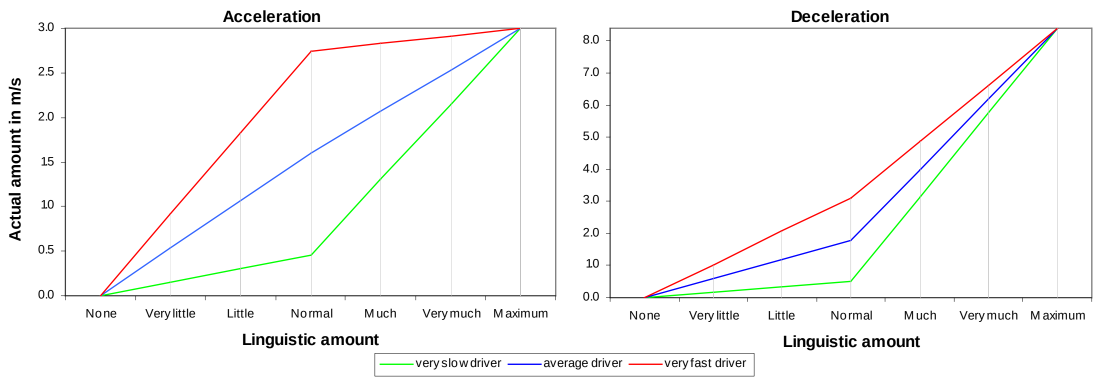


Figure 19: Acceleration and deceleration function of a vehicle

Besides an acceleration and deceleration parameter, two other behavioural parameters determine the agent's behaviour, which are the gap-acceptance-parameter and the preferred-speed-parameter. The gap-acceptance-parameter determines the desired gap between vehicles as well as the margin that is used for braking. For example, the Stop-for-red-light task will halt the vehicle at a certain distance in front of the light. The precise distance is determined by the gap-acceptance parameter.
The preferred-speed parameter determines the agent's desired speed on a straight road and in a curve. The desired speed at a curve is calculated according to the lineair interpolation of two functions designed by Emmanual Felipe and Francis Navin, who did research on the responses of drivers to horizontal curves [US Roads 2000]. They found that drivers adjust their speed according to their comfortable lateral acceleration tolerance, which is approximately between 0.35 and 0.40 g. The functions they devised are:

$$S_{comf} = 11.64 * \ln(r) - 7.32 \quad \text{with } 17 < r < 100 \qquad \textbf{\textit{Equation 8-1}}$$

$$S_{dif} = 15.56 * \ln(r) - 11.68 \quad \text{with } 17 < r < 100 \qquad \textbf{\textit{Equation 8-2}}$$

S_{comf} denotes the speed in km/h, driving at a comfortable speed and S_{dif} is the speed driving at a 'very difficult speed'. Variable r is the radius of a curve in metres. Based on these two equations we used lineair interpolation to draw up a function that determines the desired speed in a curve. Taking Equation 8-2 as an upper limit for very fast drivers and Equation 8-1 as the function for average drivers, the lower limit for very slow drivers is determined by

$$\begin{aligned} S_{low} &= (11.64 - (15.56 - 11.64)) * \ln(r) - (7.32 - (11.68 - 7.32)) \\ &= 7.72 * \ln(r) - 2.96 \qquad \textbf{\textit{Equation 8-3}} \end{aligned}$$

Thus the equation becomes

$$\begin{aligned} S &= (7.72 + f * (15.56 - 7.72)) * \ln(r) - (2.96 + f * (11.68 - 2.96)) \\ &= (7.72 + 7.84 * f) * \ln(r) - (2.96 + 8.72 * f) \qquad \textbf{\textit{Equation 8-4}} \end{aligned}$$

with f = relative preferred speed parameter between 0 an 1
and with S >= 10 and S <= 100

PART III:

THE RESULTS

Chapter 9: Evaluating the program

In this chapter we will demonstrate the reasoning process of the implemented tactical-level driving agent and make some critical remarks about our prototype program. We start with an example situation in section 9.1 and explain how our agent deals with it. In section 9.2 we discuss one of our experiments and show that the agent exhibits human-like driving behaviour. In section 9.3 we will make some comments about the implementation of our simulator program and in section 9.4 we close this chapter with the discussion if our driving agent is really worth the name 'agent'.

9.1 The agent's reasoning process: an example

We will demonstrate the reasoning process of the implemented driving agent with the use of an example situation. The example situation, which is shown in Figure 20, consists of four vehicles near an intersection with traffic lights. The agent we will be looking at is the one controlling the vehicle marked with the red dot. At the moment the agent is approaching the intersection while driving behind another vehicle.

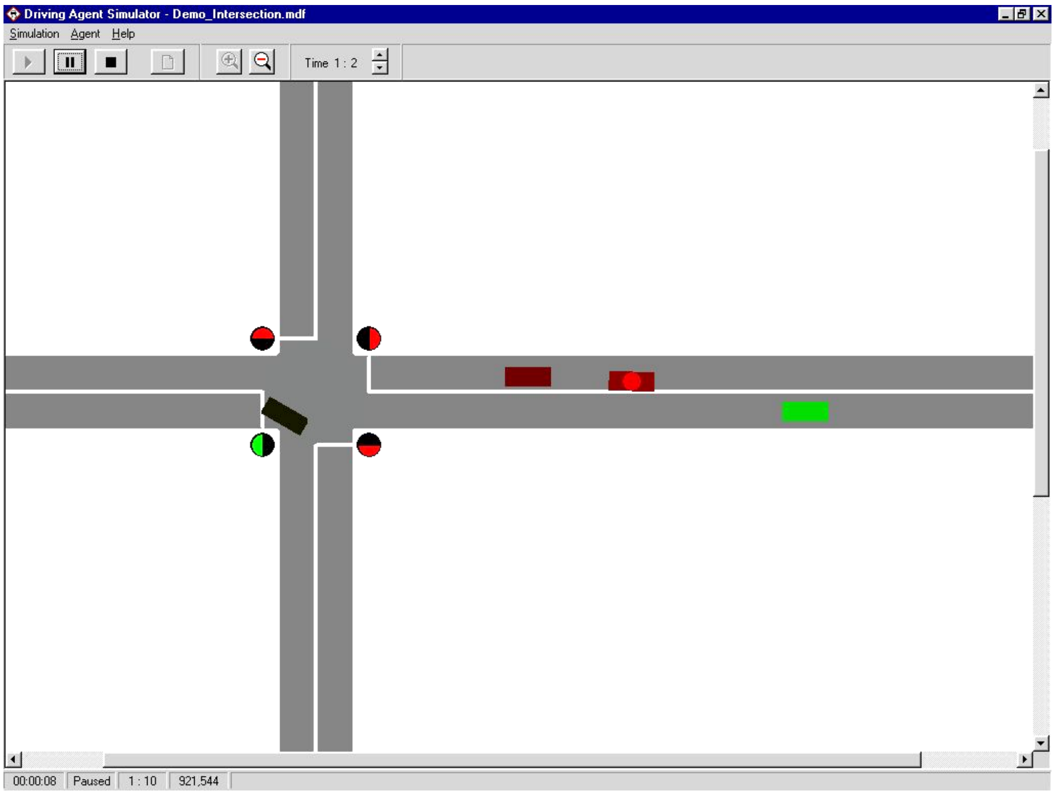


Figure 20: Example situation consisting of four vehicles, an intersection and a group of traffic lights

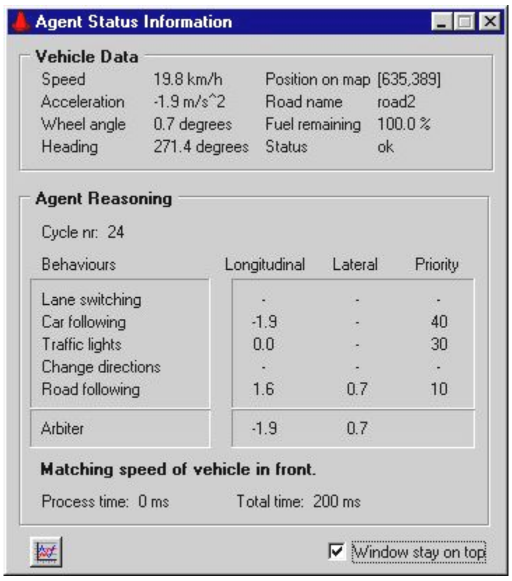
The colour of all four vehicles represents their current rate of acceleration. Green vehicles are accelerating and the greener they are, the faster they are accelerating. The same goes for braking, but in this case a vehicle is coloured red. Vehicles that are not accelerating or braking are coloured black, for example the vehicle in Figure 20 turning right at the intersection.

9.1.1 Sensor information

In our example situation the agent, has just received 7 different sensor messages[4]. One message contains information about the vehicle controlled by the agent, for example its speed, acceleration, the angle of its wheels, fuel level etc. This vehicle-status message is sent to the agent every reasoning cycle. Another sensor message contains information about the road the agent is currently driving on. This message includes the distance to the left and the right edge of the road, the distance to the left and right edge of the vehicle's lane, and the heading of the agent relative to the road. A third message contains information about the intersection the agent is approaching, including the distance of the agent to the intersection and the relative angles of the sideroads. The latter are the possible directions the agent can choose from. Message four is a message from the traffic-lights sensor that has detected one visible traffic light, which is the red light in the upper right corner of the intersection. The other lights face another direction and are not clearly visible to the agent, so the traffic-lights sensor ignores them. Finally, the last three messages are received from the vehicles sensor that has detected three different vehicles in the area around the agent. Messages from the vehicles sensor contain information about a vehicle's speed, acceleration, and heading relative to the agent's own heading.

9.1.2 Activated behaviour rules

In our example situation, three of the five implemented behaviour rules (see also section 8.5), are active, as is shown in Figure 21. At the lowest level the Road-following behaviour has proposed to accelerate. The reason for this is that the behaviour has noticed that the vehicle is not driving at its preferred speed, which is currently set to approximately 50 km/h. The Road-following behaviour also noticed that the vehicle is very close to the left edge of its current lane, so it also proposed to turn the steering wheel 0.7 degrees to the right.


Agent Status Information

Vehicle Data

Speed	19.8 km/h	Position on map	[635,389]
Acceleration	-1.9 m/s^2	Road name	road2
Wheel angle	0.7 degrees	Fuel remaining	100.0 %
Heading	271.4 degrees	Status	ok

Agent Reasoning

Cycle nr: 24

Behaviours	Longitudinal	Lateral	Priority
Lane switching	-	-	-
Car following	-1.9	-	40
Traffic lights	0.0	-	30
Change directions	-	-	-
Road following	1.6	0.7	10
Arbiter	-1.9	0.7	

Matching speed of vehicle in front.

Process time: 0 ms Total time: 200 ms

☑ Window stay on top

Figure 21: Agent Status Information window belonging to our agent in the example situation

The second active behaviour is the Traffic-lights behaviour. This behaviour has proposed not to accelerate since it has noticed a red light nearby. Since the light is still reasonably far compared to the agent's current speed and the agent is driving rather slow (approximately 20 km/h), the Traffic-lights behaviour does not (yet) propose to start braking.
The third active behaviour, which is the most important one in this case, is the Car-following behaviour. The Car-following behaviour has noticed a slower vehicle in front of the agent.

[4] The precise content of each sensor messages is described in Appendix A.1, Figure 29.

This vehicle is currently braking for the red traffic light and the agent has decided to match the vehicle's speed, so it proposed to brake (negative acceleration) with 1.9 m/s.
In this situation the Change-directions behaviour is not activated because the vehicle is still reasonably far away from the intersection and taking the agent's slow speed into account it does not see any reason to alter this. Normally, the Lane-switching behaviour might propose to overtake the slower driving vehicle in front of the agent, but because there's an intersection nearby it decides not to do so.
In our example only the Road-following behaviour has proposed to change the vehicle's direction so the arbiter selects this suggestion. Of all the active behaviours that have suggested changing the agent's speed, the Car-following behaviour has the highest priority so its longitudinal proposal is combined with the lateral proposal of the Road-following behaviour and both are sent to the vehicle.

9.2 Human-like driving behaviour

Although we have not validated the used simulation and driving behaviour parameters yet, preliminary experiments have shown that our implemented agent exhibits human-like driving behaviour ranging from slow and careful to fast and aggressive driving behaviour. In this section we will present the results of one of our experiments.
The experiment consists of two different drivers approaching an intersection and stopping in front of a red traffic light. Both drivers perform this task without any other traffic present. The first driver is a careful driver with a low preferred speed, reasonably large gap acceptance and a low preferred rate of deceleration. We call this driver the 'grandpa' driver. The second driver is a young and aggressive driver, with a high preferred speed, small gap acceptance and a high preferred rate of deceleration. The drivers start at the same distance from the intersection. The speed of both vehicles during the experiment is shown in Figure 22.

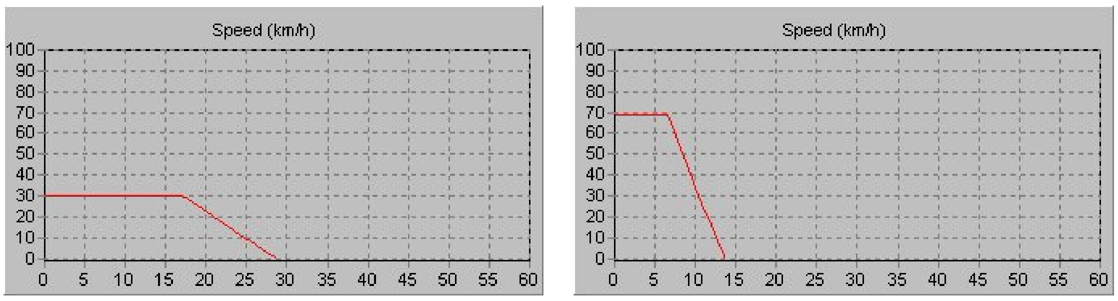


Figure 22: Speed of the grandpa driver (left) and young aggressive driver (right) during the experiment

Since the grandpa driver is driving at a lower speed, it takes a while before he starts braking, but his braking start-point (50m) is closer to the intersection than that of the young aggressive driver (65m), due to his lower speed. The difference between the used braking pressures is clearly visible. Both drivers brake with a relatively stable deceleration (approximately 0.7 m/s^2 and 2.7 m/s^2), which is consistent with human braking behaviour [Aasman 1995].
The experiment was done several times, but in almost all cases the shown graphs were roughly the same. Also, the precise stopping positions of both vehicles were approximately the same in all experiments. The young aggressive driver had a tendency to brake relatively late and often came to a stop just in front of or on the stopping line. The grandpa driver on the other hand always came to a full stop well ahead of the stopping line. The stopping positions of both vehicles during one of the experiments is compared in Figure 23.

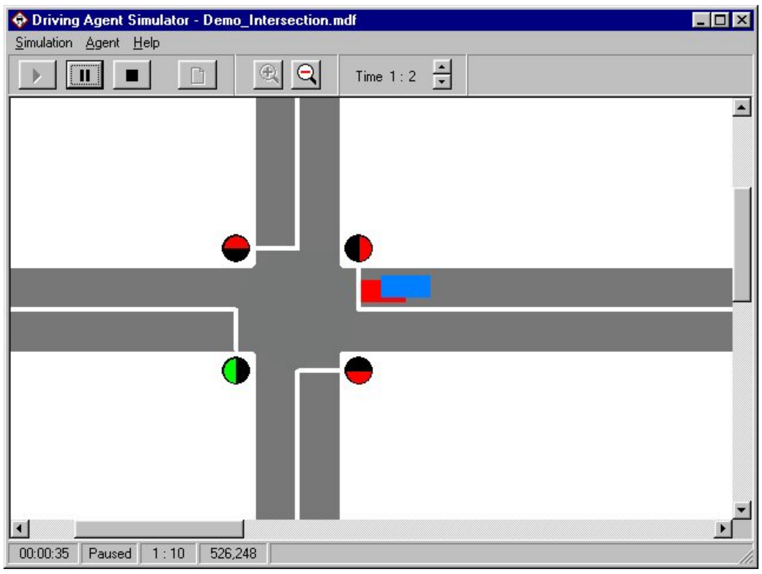


Figure 23: Compared stopping positions of the young aggressive driver (red vehicle) and grandpa driver (blue vehicle)

9.3 Critical remarks and possible improvements

The aim of our simulation program is to test the design and functionality of our driving agent, but as a result its current implementation is rather inefficient since we did not optimise it for speed. We have implemented most of the suggested agent parts described in chapter 6 but some of them are not necessary in our program. For example, if the agent sends a request for sensor data, all the data is sent immediately, so the extra storage of messages in the communication input part done in our program is unnecessary. All messages can be transferred to the agent’s memory directly since the agent will not continue its work until all sensor information for that cycle is received. Also, our main focus was on the correctness of the agent’s driving behaviour and reasoning process, and not on creating a fast executing agent. The computer used to implement and test the program is an Intel Pentium III, 450 MHz with 64 MB of RAM, running the Microsoft NT 4.00 operating system. The help-file of our programming language recommends a limit of 16 threads (agents), assuming that most threads are waiting for external events. However, our experiments have demonstrated that at least 30 agents can run in our simulation simultaneously without any problem, so this number might be a bit low.
A drawback of our simulator is that some unrealistic assumptions were made. Agent perception is perfect. All agents have a field of view of 360 degrees and objects are not obscured or covered by other objects. Further, vehicle actions are atomic. For example, braking at a certain rate occurs instantly after the action is sent to the vehicle. In real life this would occur more gradually. Also, pedestrians, cyclists and crosswalks are not yet modelled so the agent’s ability to react to unexpected events was not yet accurately tested. We also did not test our design for deadlock. Deadlock is a situation where further movement of one or more vehicles is inhibited due to the current status of the vehicle or its close environment. In our program deadlock might occur if due to some unforeseen reason two vehicles crash and block the road. Since there is no planning agent implemented yet that can replan an agent’s route, other agents will stop behind the crashed vehicles and will wait until the road is cleared.
Although we had good experiences with the Borland Delphi 3 programming environment, we were not too enthusiastic about the newer Delphi Professional 5.0 version for NT that we used in this project. It has several bugs that slowed down the implementation process. One of the most time-consuming problems is that the Delphi 5 debugger has a problem dealing with threads. On average one out of five times the debugger would generate an error message while debugging and lock up the Delphi programming environment.

9.4 Is our driving agent really an agent?

If we look back at our discussion about the definition of intelligent agents in chapter 3, one might wonder if the driving agent we constructed is really an intelligent agent. If we follow the statement of Shoham that any piece of hardware or software is an agent if one has chosen to analyse and control it as an agent, this question becomes trivial. We have chosen to design and implement a ‘driving agent’ and therefore it is an agent. However, we feel that this conclusion is too simplistic. It is more realistic to look at the properties that an agent should have and check if our driving agent also possess them.

- *Responsive:* our driving agent certainly is that. It can perceive its environment through its sensors and will act based on these perceptions.
- *Autonomous:* our agent is also autonomous. The user does not have to do anything and the agent will drive around on its own (until it runs out of fuel).
- *Adaptive:* the implemented version of our agent is not really adaptive. It cannot alter its behaviour rules by itself or learn from experience. Note however, that in the agent design discussed in chapter 7 we showed that the agent could change its parameters on the occurrence of certain events, so the agent’s design allows (a small amount of) adaptivity.

We can conclude that the current implemented agent has two out of the three main trades associated with intelligent agents. In chapter 3, we stated that not all agents are capable of learning and most programs that are called agents do not possess all of the three trades. Therefore, we feel that we can safely call our driving agent a real agent, at least at the moment. Probably, a few years from now our agent will be outdated and new agent’s technologies and insights will prove themselves, making the driving agent an old-fashioned program.

9.4.1 What type of agent is it?

The driving agent is not strictly an agent in the sense of the subsumption architecture. Although its behaviour rules operate according to the subsumption architecture, some behaviours use a temporary memory storage, something that is not allowed in a strictly reactive or reflexive architecture. Also, the agent’s processing is done in a cyclic manner, as is done in the traditional ‘sense-plan-act’ AI architectures. In purely reactive architectures behaviours are activated at the same moment the input stimuli is received. It is clear that the designed agent has trades of both the traditional AI approach and the reactive and reflexive agent approach. However, the main functionality of our agent lies in its behaviour rules. As a result we feel that our agent is more of a reactive agent than a deliberative agent.

9.4.2 Behaviours as agents?

Some might argue that each behaviour of our driving agent is in fact a simple reactive agent and that our agent is a multi-agent system with competitive agents trying to control the vehicle. We disagree. It is true that most of our behaviours are constructed according to the ideas of the subsumption architecture, using reactive layers (we have called them tasks) that can dominate lower layers. However, each behaviour can only function with the help of other entities such as the sensors, memory and the communication module. If any these parts fail the behaviour cannot do its work and therefore they are not autonomous. Since autonomy is one of the main properties of an agent we feel that the behaviours are not agents.

Chapter 10: Conclusions

In this last chapter we will discus the results of our study and draw some conclusions. In section 10.1 we give a short overview of the work that we have done and mention some of the advantages and disadvantages of our approach. In section 10.2 we present some suggestions for future research.

10.1 Results

In this report we have investigated the use of intelligent agents to control autonomous vehicles. More specifically, we have looked at agents that perform tactical-level driving, making short-term decisions about their driving task in real-time. After our survey of existing techniques and applications, we designed a general model of a driving agent that could be applied to control a vehicle, regardless of the agent's environment, vehicle type and precise task. Next, we expanded our design so that the agent could be used to model different types of drivers. This way an agent could act as an individual driver with its own set of characteristics. A prototype simulation program was constructed to test our improved agent design. Preliminary experiments with the simulation program have shown that our driving agent exhibits human-like driving behaviour and is capable of modelling different driving styles, ranging from slow and careful to fast and aggressive driving behaviour.

The main advantage of agent-based microscopic traffic simulation over the more traditional macroscopic simulation is that it is more realistic. Instead of using general traffic-flow models, traffic becomes an emergent property of the interaction between agents. Another advantage is that agent-based simulation is more flexible. Changes to traffic scenarios can be made quickly by altering the position of individual vehicles and changing an agent's parameters. A disadvantage is the increase of computational resources and the higher number of parameters that need to be set and validated.

The use of agents in microscopic traffic simulators is not new. Several simulators have used agents to model vehicles and some of them even use some sort of aggression factor to model different driver types. However, our approach differs from all the simulators that we studied. First of all, our simulator differs in the used environment. Most microscopic traffic simulators model highway or freeway traffic. Our simulator deals with driving in an urban environment, which is more complicated. Second, with our agent multiple behaviour parameters can be set to produce to desired driving behaviour. Using just one parameter severely limits the possibilities to model different behaviours. Third, by using relatively independent behaviour rules our agent's functionality can be expanded or altered easily and can be used in completely different environments.

Besides improving the performance of traffic simulations, our driving agent can be seen as the first step towards controlling real-life autonomous vehicles. These vehicles are not necessarily cars. One can also think about robots or autonomous guided vehicles. Although we cannot prove it (yet), we are confident that our agent can be equipped with the right behaviour rules to control a real-life vehicle.

10.2 Future work

At the moment we are experimenting with different types of agents in several scenarios. The goal is to study the current possibilities of our simulation program and driving agent in order to improve them further. From our current position, we can go into two different directions. The first is improving the realism of the simulation program and driving agent's behaviour in order to create more realistic simulations. The second is improving our agents to control real-life autonomous vehicles instead of simulated ones.

10.2.1 Simulation

In its current state our traffic simulator and driving agent implementation are limited, mainly because they were meant as a prototype. The simulation environment can be made more realistic by adding new objects such as busses, trucks, emergency vehicles, pedestrian crossings, cyclists, traffic signs, trees and buildings. Once the simulator is improved with the new objects the agent's functionality must be extended to deal with these objects. In addition, the simulation environment needs to be validated. Although we have tried to use realistic values for vehicle acceleration, turn radius, road size etc, the used settings might prove to be inaccurate. In addition, we need to study human driving behaviour more extensively in order to validate our different driving style models.

Other possible improvements are the random or stochastic appearance of new vehicles driving in from outside the environment map and creating pre-determined driving styles, such as a typical commuter, a drunk driver, or a 'grandpa' driver. Another improvement is the enhancement of the used visualisation techniques. At the moment the simulation uses a 'bird's-eye' viewpoint. We think that it is very interesting to also include an in-car viewpoint and look at the environment from the driver's perspective. If we add a human control interface (e.g. steering wheel and pedals) then our simulation program can also be used to study real human driving behaviour and their reaction to other drivers.

The drawback of the proposed improvements will be that both the simulation environment and the agent will need more computation time and will run more slowly. Therefore, we recommend using a distributed approach in the future so that the driving agents can run on different computers. The environment and simulation controller discussed in section 8.3 can act as a server and the agents can be the clients communicating to the server.

10.2.2 Real-life autonomous vehicles

The ultimate driving agent is one that can control a real-life vehicle, instead of a simulated one, making real intelligent vehicles possible. The ultimate agent should be able to drive different types of vehicles under all possible circumstances and in all possible situations. Also, in the perfect case, it can communicate with other intelligent agents, providing or receiving information about road conditions, traffic jams or other unexpected events.

We believe the agent design presented in chapter 6 to be good enough to control real vehicles, provided that the necessary sensor technology will be developed. However, there is still a long way to go before we can actually try to implement such a complex agent. The next step from our current project would be to create more realistic sensors for the simulated agent. At the moment the implemented sensors are 'perfect', The sensor readings do not contain any noise and the possibility of objects that obscure other objects is neglected. The current implementation of the driving agent should be adapted to be able to deal with uncertainty and noisy sensor readings. In addition, a planning and navigation agent is necessary to give directions to the tactical-level driving agent.

If this is successful, we can try to use the agent on a real moving object, for example a robot. The first steps towards this end have already been taken. Recently, we have experimented with a very simple driving agent to control a LEGO Mindstorms robot [LEGO 2000] shown in Figure 24 and the first results are positive. Using two simple behaviours that are called 'collision detection' and 'random wandering', the LEGO robot is able to explore its environment. We must note however that the LEGO robot's sensors, and thus its capabilities, are somewhat limited. We intend to expand the robot with more advanced sensors and do further research in this area in the future.

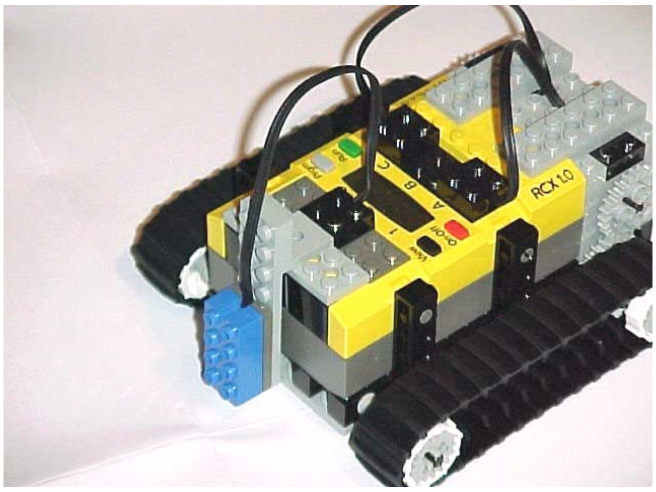


Figure 24: LEGO Mindstorms robot

Bibliography

Note: all Internet addresses were checked and operational on December 22nd, 2000.

Aasman, J. (1995) *Modelling driver behaviour in Soar.* Dissertation Rijkuniversiteit Groningen, KPN Research, Leidschendam, The Netherlands.

Arkin, R. C. (1998) *Behavior-based robotics.* The MIT Press, Cambridge, Massachusetts.

Bigus, J.P. and Bigus, J. (1998) *Constructing intelligent agents with Java: a programmer's guide to smarter applications.* Wiley & Sons, Inc.

Bomarius, F. (1992) *A Multi-Agent Approach towards Modelling Urban Traffic Scenarios.* Research Report RR-92-47, Deutches Forschungszentrum für Künstliche Intelligenz, September 1992.

Bradshaw (1997) *Software agents.* AAAI Press / The MIT Press, Cambridge, Massachusetts.

Brooks, R.A. (1986) *A robust layered control system for a mobile robot.* MIT AI Lab Memo 864, September 1985. Also in *IEEE Journal of Robotics and Automation Vol. 2*, No. 1, March 1986, pages 14-23. Also http://www.ai.mit.edu/people/brooks/papers/AIM-864.ps.Z

Carnegie Mellon University (CMU) Robotics Institute (2000) http://www.ri.cmu.edu/projects/project_178.html

Chan, S. (1996) *Multi-Agent Traffic Simulation - Vehicle,* MSc dissertation, Department of Artificial Intelligence, University of Edinburgh. http://www.dai.ed.ac.uk/daidb/staff/personal_pages/mingshu/dissertations/stanleyc-dissertation.ps.gz

Chong, K.W. (1996) *Multi-Agent Traffic Simulation - Street, Junction and Traffic light,* MSc dissertation, Department of Artificial Intelligence, University of Edinburgh. http://www.dai.ed.ac.uk/daidb/staff/personal_pages/mingshu/dissertations/kenc-dissertation.ps.gz

CLIPS (2000) http://www.ghg.net/clips/CLIPS.html

Connekt (2000) http://www.connekt.nl

Ehlert, P.A.M. (1999) *The use of artificial intelligence in autonomous mobile robots.* Research report, Delft University of Technology. http://elektron.its.tudelft.nl/~s218303/docs/OTreport_ps.zip

Ferber, J. (1999) *Multi-agent systems: an introduction to distributed artificial intelligence.* Addison Wesley Longman Inc., New York.

Finin, T., Labrou, Y. and Mayfield, J. (1994) *KQML as an agent communication language*. In *Proceedings of the Third International Conference on Information and Knowledge Management* (CIKM'94), ACM Press, November 1994.
Also http://www.cs.umbc.edu/agents/introduction/kqmlacl.ps

FIPA (1997) *Agent communication language.* Technical report of the foundation for intelligent physical agents. http://www.cselt.it/fipa/spec/fipa97/f8a22.zip

FIPA (2000) http://www.cselt.it/fipa or http://www.fipa.org

Franklin, S. and Graesser, A. (1996*) Is it an agent, or just a program?: a taxonomy for autonomous agents.* In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Intelligent Agents III*, Springer-Verlag New York, pages 21-35.
Also http://www.msci.memphis.edu/~franklin/AgentProg.html

FROG (2000) http://www.frog.nl

Gensereth, M. and Fikes, R. (1992) *Knowledge Interchange Format, Version 3.0. Reference Manual*. Logic Group Report Logic-92-1, Stanford University. http://logic.stanford.edu/papers/kif.ps

Gilbert, D., Aparicio, M., Atkinson, B., Brady, S., Ciccarino, J., Grosof, B., O'Conner, P., Osisek, D., Pritko, S., Spagna, R. and Wilson, L. (1995) *IBM Intelligent Agent Strategy*, IBM corporation.

Hayes-Roth, B. (1985) *A blackboard architecture for control.* In *Artificial Intelligence Vol. 26*, pages 251-321.

Hewitt, C. (1977) *Viewing control structures as patterns of passing messages*. In *Artificial Intelligence, Vol. 9*, No.3, pages 323-374.

Hoedemaeker, M. (1999) *Driving with intelligent vehicles: driving behaviour with adaptive cruise control and the acceptance by individual drivers*. Trail thesis series T99/6, November 1999, Delft University Press, The Netherlands.

Ioannou, P.A. (1997) *Automated Highway Systems.* Plenum Press, New York.

Jennings, N. R. and Wooldridge, M.J. (1998) *Agent technology: foundations, applications, and markets*. Springer-Verlag, Berlin, Germany.

Labrou, Y. and Finin, T. (1997a) *A proposal for a new KQML specification.* Technical report CS-97-03, University of Maryland, February 1997. http://www.csee.umbc.edu/~jklabrou/publications/tr9703.ps

Labrou, Y. and Finin, T. (1997b) *Semantics and conversations for an agent communication language*. In *Readings in Agents*, Huhns, M. N and Singh, M.P. editors, Morgan Kaufmann Publishers, Inc. Also in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (IJCAI-97), August 1997. http://www.cs.umbc.edu/~finin/papers/ijcai97.ps

Lego Mindstorms (2000) http://mindstorms.lego.com

Maes, P. (1994) *Agents that reduce work and information overload.* In *Communications of the ACM, Vol. 37*, No. 7, pages 31-40, ACM Press, July 1994.
Also http://pattie.www.media.mit.edu/people/pattie/CACM-94/CACM-94.p1.html

Minderhoud, M.M. (1999) *Supported driving: impacts on motorway traffic flow.* Trail thesis series T99/4, TRAIL Research school, Delft University Press, The Netherlands.

Minsky, M. (1986) *Society of the mind*. Simon & Schuster, New York.

Nwana, H.S. (1996) *Software agents: an overview*. In *Knowledge Engineering Review Journal, Vol. 11*, No. 3, November 1996. Also http://www.labs.bt.com/projects/agents/publish/papers/ao4.ps.gz or http://www.sce.carleton.ca/netmanage/docs/AgentsOverview/ao.html

Pfeifer, R. and Scheier, C. (1999) *Understanding intelligence*. MIT Press, Cambridge, Massachusetts.

Pomerleau, D. (1993) *Neural network perception for mobile robot guidance*. Kluwer Academic Publishers, Boston.

Rao, A. S., and Georgeff, M. P. (1995) *BDI Agents: From Theory to Practice*. In *Proceedings of the 1st International Conference on Multi-Agent Systems* (ICMAS-95), San Francisco, USA, June, pages 312-319.

Robocup (2000) http://www.robocup.org

RUG-COV Driving simulator (2000) http://www.ppsw.rug.nl/cov/cov_sim.htm and http://www.rug.nl/rc/pictogram/199905-2/rijsimul.htm (in Dutch).

Russell, S. and Norvig, P. (1995) *Artificial Intelligence: a modern approach*. Prenctice-Hall International, Inc.

SAVE project (2000) http://www.iao.fhg.de/Projects/SAVE/save/saveinit.htm

SMARTEST project; Algers, A., Bernauer, E., Boero, M., Breheret, L., Di Taranto, C., Dougherty, M., Fox, K., Gabard, J. (1997) *Review of micro-simulation tools*. Institute of transport studies, University of Leeds, August 1997. http://www.its.leeds.ac.uk/projects/smartest/deliv3f.html

Shoham, Y. (1993) *Agent-oriented progamming.* In Artificial Intelligence 60, pages 51-92.

Shoham, Y. (1997) *An overview of agent-oriented programming*. In *Software agents.* Bradshaw, J.M editor, AAAI Press / The MIT Press, Cambridge, Massachusetts, 1997.

Skarmeas, N. (1999) *Agents as objects with knowledge base state.* Imperial College Press, London.

Smiley, A. and Brookhuis, K.A. (1987) *Alcohol, drugs and traffic safety*. In *Road users and traffic safety*, Rothengatter, J.A. and De Bruin, R.A. editors, Assen: Van Gorcum, pages 83-105.

Sukthankar, R. (1997) *Situation awareness for tactical driving.* Technical report CMU-RI-TR-97-08, Robotics institute, Carnegie Mellon University, January 1997. http://www.cs.cmu.edu/afs/cs/usr/rahuls/www/pub/thesis.ps.gz

Sukthankar, R., Hancock, J., Pomerleau, D. Thorpe, C. (1996) *A Simulation and Design System for Tactical Driving Algorithms*. In *Proceedings of artificial intelligence, simulation and planning in high autonomy systems.* Also http://www.cs.cmu.edu/afs/cs/usr/rahuls/www/pub/shiva_ais.ps.gz

Tecuci, G. (1998) *Building intelligent agents: an apprenticeship multistrategy learning theory, methodology, tool and case studies.* Academic Press, San Diego, California.

TRAIL (1999) *Automation of car driving: exploring societal impacts and conditions.* Heijden, van der, R.E.C.M., Wiethoff, M. editors, December 1999, TRAIL studies in transportation science, S99/4, Delft University Press, The Netherlands.

Tzomakas, C. (1999) *Contributions to the visual object detection and classification for driver assistance systems.* Dissertation April 1999, Ruhr-Universität Bochum, Shaker Verlag, Aachen Germany.

US Roads (2000) *Canadian researchers test driver response to horizontal curves.* Road Management & Engineering journal. http://www.usroads.com/journals/rmej/9809/rm980904.htm

Wittig, T. (1992) *ARCHON: an architecture for multi-agent systems.* Ellis Horwood Limited, England.

Appendix A: Implementation details

The object model in section A.1 contains the important objects that are implemented in the prototype Driving Agent Simulator program. In order not to confuse the reader with too much detail, the less important objects, attributes, and operations are not described here.

A.1 Object model

Note that, unlike standard OMT notation, some objects in this model contain other objects. Also note that shaded objects are complex objects that are shown in more detail in another figure.

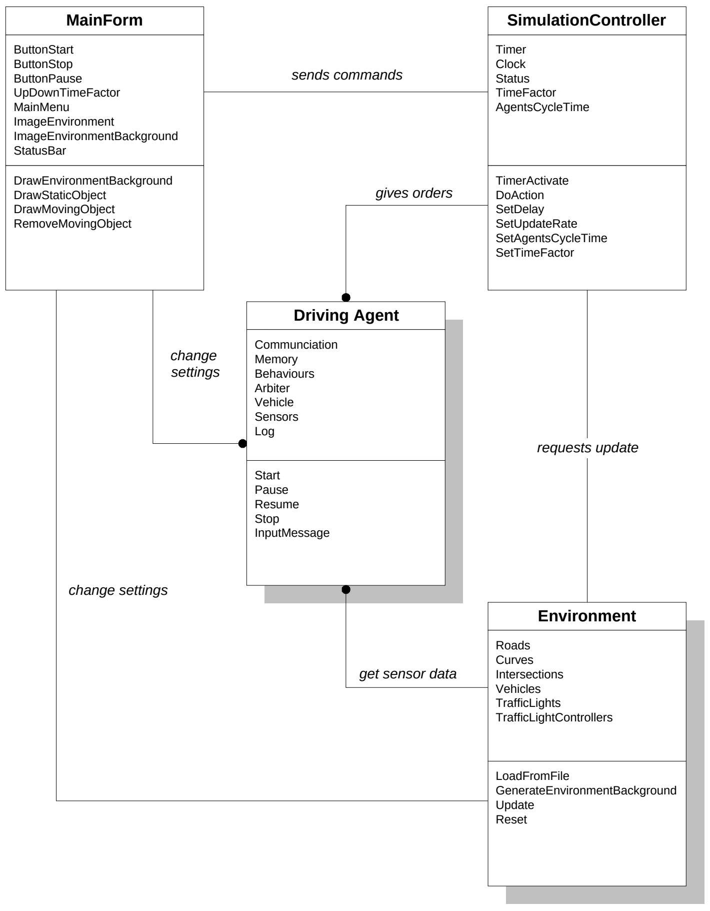


Figure 25: The main objects of the simulator program

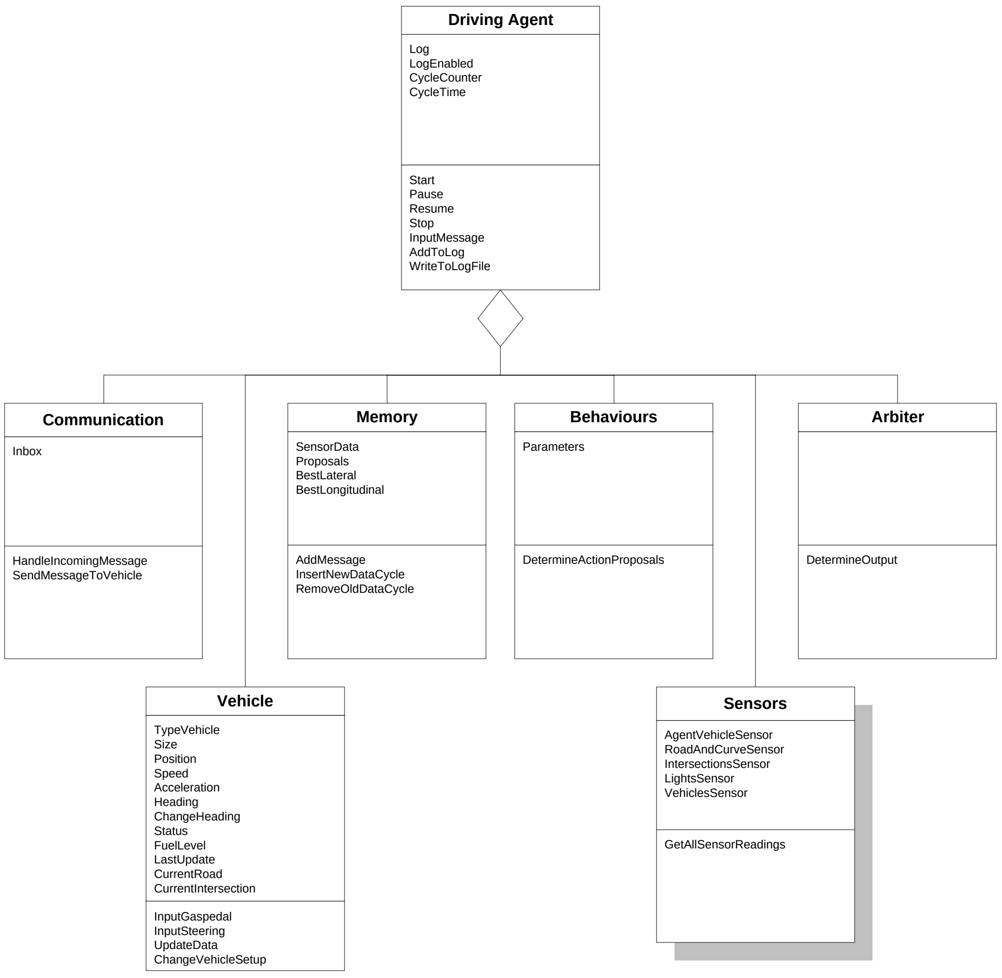


Figure 26: Driving agent object

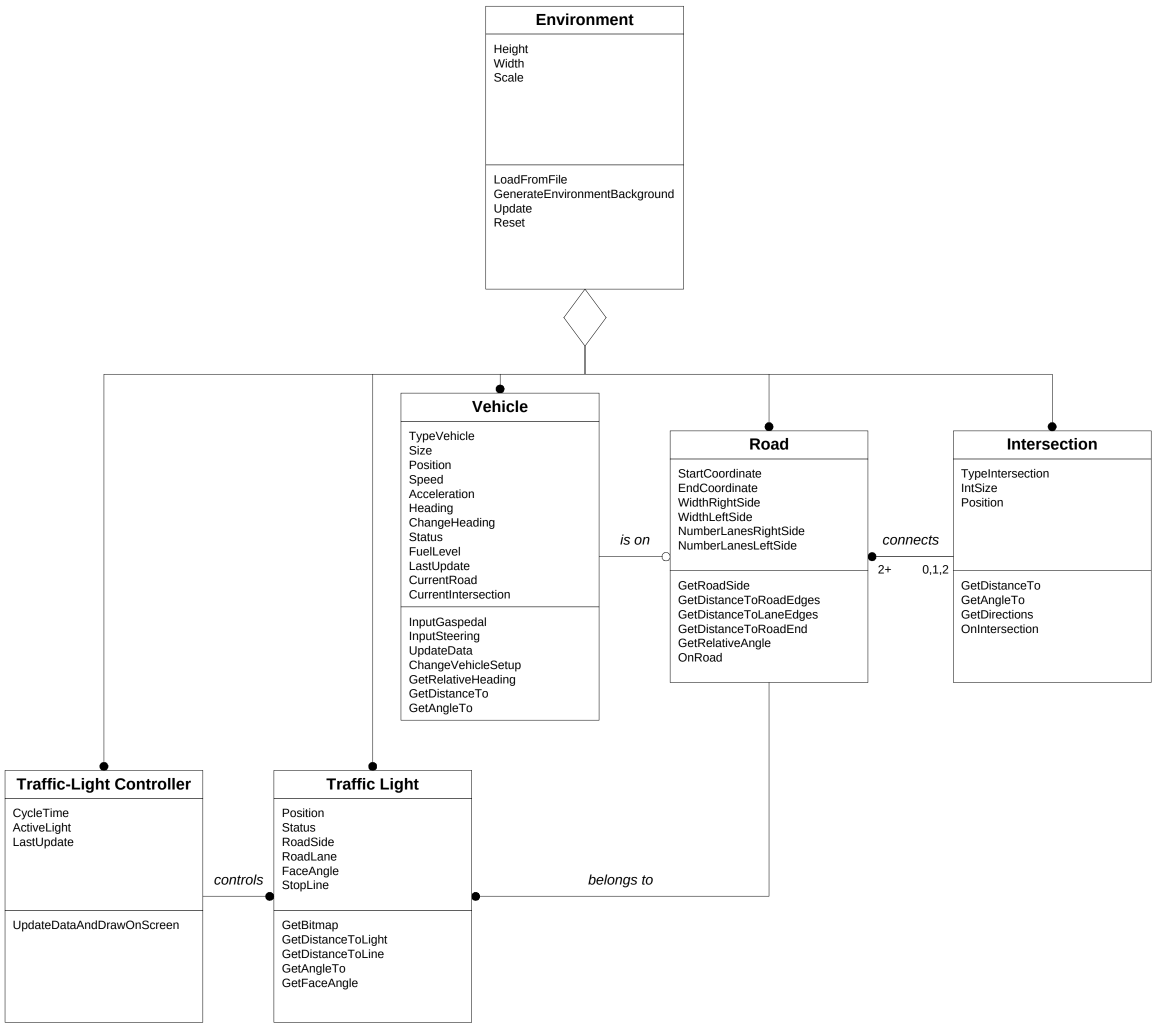


Figure 27: Environment object

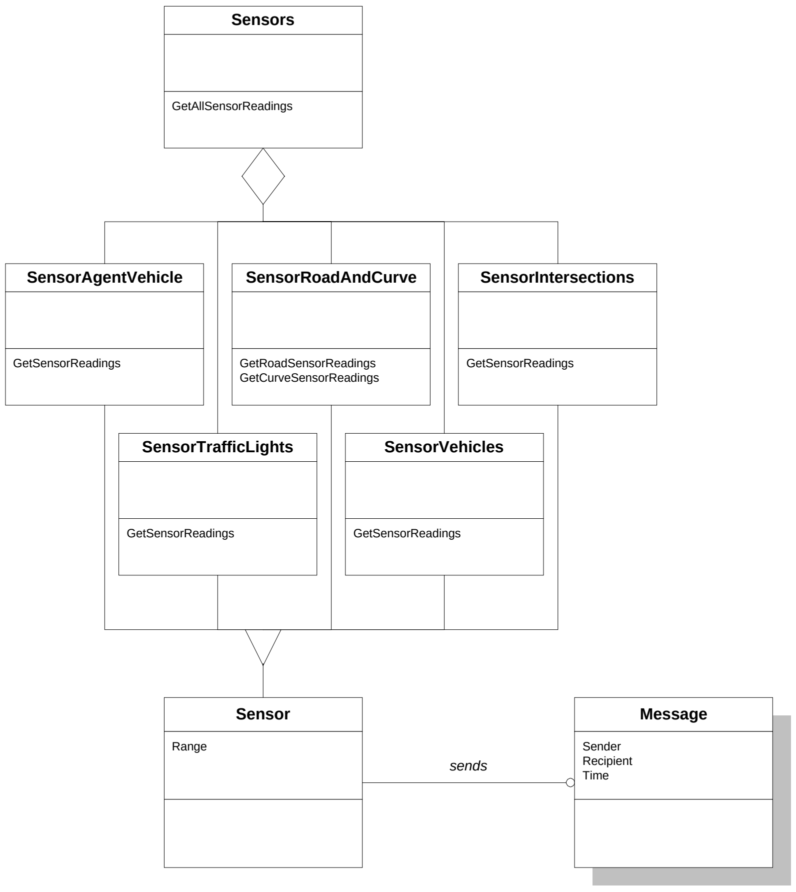


Figure 28: Sensors object

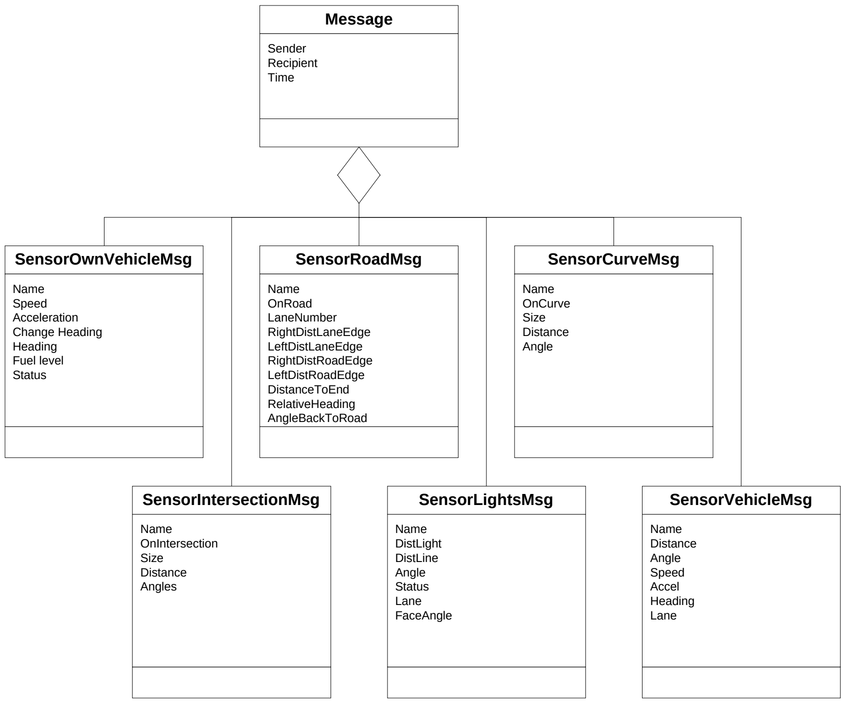


Figure 29: Sensor messages

A.2 Agent behavioural rules

This section contains a simplified version of the used agent behavioural rules and tasks in pseudo-code.

```
function RoadFollowing returns Action
   if (data available) and (Agent on a road) then
      Task_DriveAtPreferredSpeed(Action)
      Task_StayInLane(Action)
      Task_BrakeForEndOfRoad(Action)
   else if not ((Agent on a road) or (Agent on a intersection)) then
      Task_OffRoad(Action)
   if (Agent near curve) then
      Task_AdjustSpeedForCurve(Action)
      Task_TakeCurve(Action)


procedure Task_DriveAtPreferredSpeed
   if (Agent in curve) then
      PrefSpeed := Parameters.GetPrefSpeedCurve
   else if (Agent on intersection) then
      PrefSpeed := Parameters.GetPrefSpeedIntersection
   else
      PrefSpeed := Parameters.GetPrefSpeedStraightRoad
   if not (Agent speed = PrefSpeed) then
      if (Agent speed < PrefSpeed) then
         Action := Accelerate(Normal)
      else
         Action := Brake(VeryLittle)
   else
      Action := Accelerate(0)


procedure Task_StayInLane
   if (Agent heading <> Road direction) then
      Action := Steer road direction
   else
      Action := Steer straight
   if (Agent distance to right lane edge < Parameters.Margin) then
      Action := Steer left
   else if (Agent distance to left lane edge < Parameters.Margin) then
      Action := Steer right


procedure Task_AdjustSpeedForCurve
   Direction := Curve.Angle
   Distance := CalculateBrakeDist(Parameters.PrefSpeedCurve, Brake(normal))
   if (Curve.Distance < Distance) then
     Action := CalculateDeceleration(PrefSpeedCurve, Curve.Distance)


procedure Task_TakeCurve
   if (Agent in Curve) then
      if not (Agent heading = Curve.Angle) then
         Action := Steer Direction       // Direction set in AdjustSpeedForCurve
      else
         Action := Steer straight


procedure Task_BrakeForEndOfRoad
   if (Agent is near end of road) then
      Distance := CalculateBrakeDist(Speed = 0, Brake(normal))
      if (Distance < distance to road end) then
         Action := CalculateDeceleration(Speed = 0, distance road end)


procedure Task_OffRoad
   Action := Steer back to nearest road
```

```
function ChangeDirections returns Action
   if (data available) then
      if (Agent is on a road) then
         Task_TurnAroundAtDeadEnd(Action)
      if (Agent is near an intersection) then
         Task_AdjustSpeedForIntersection(Action)
         Task_CrossIntersection(Action)

procedure Task_TurnAroundAtDeadEnd
   if (Agent is very near end of road) then
      Action := Turn 180 degrees

procedure Task_AdjustSpeedForIntersection
   ChosenDirection := Choose random direction
   Distance := CalculateBrakeDist(Parameters.PrefSpeedInt, Brake(normal))
   if (Intersection.Distance < Distance) then
      Action := CalculateDeceleration(PrefSpeedInt, Intersection.Distance)

procedure Task_CrossIntersection
   if (Agent on Intersection) then
      if not (Agent heading = ChosenDirection) then
         Action := Steer ChosenDirection   // set in AdjustSpeedForIntersection
      else
         Action := Steer straight



function TrafficLights returns Action
   if (data available) and (lights detected) and (agent is on a road)
      Task_DealWithYellowLight(Action)
      Task_StopForRedLight(Action)

procedure Task_DealWithYellowLight
   Light := FindCorrectTrafficLight
   Distance := CalculateBrakeDist(Speed = 0, Brake(normal))
   if (possible to stand still in time) then
      if (Light.DistanceStopLine < Distance)
         Action := CalculateDeceleration(Speed = 0, Light.DistanceStopLine)
   else
     Action := do nothing

procedure Task_StopForRedLight
   Light := FindCorrectTrafficLight
   Distance := CalculateBrakeDist(Speed = 0, Brake(VeryMuch))
   if (possible to stand still in time) then
      if (Light.DistanceStopLine < Distance)
         Action := CalculateDeceleration(Speed = 0, Light.DistanceStopLine)
   else
     Action := do nothing



function CarFollowing returns Action
   if (data available) and (agent is on a road) then
      Vehicle := FindVehicleInFront
      if Vehicle exists then
         Task_MatchSpeed(Action, Vehicle)
         Task_KeepDistance(Action, Vehicle)

procedure Task_MatchSpeed
   if (Agent.Speed > Vehicle.Speed) then
      Distance := CalculateBrakeDist(Speed = Vehicle.Speed, Brake(Little))
      if (Vehicle.Distance < Distance) then
         Action:= CalculateDeceleration(Speed=Vehicle.Speed, Brake(Little))
```

```
procedure **Task_KeepDistance**
   if (Vehicle.Speed > 0) and (Agent.Speed almost same Vehicle.Speed) then
      if (Vehicle.Distance > Parameters.PrefDist) then
         Action := Accelerate(little)
      if (Vehicle.Distance < Parameters.PrefDist) then
         Action := Brake(little)



function **LaneSwitching** returns Action
   if (data available) and (agent is on a road) then
      Task_SwitchToRightLane
      if (vehicles in area) then
         Vehicle := FindVehicleInFront
         if Vehicle exists then
            Task_OverTakeVehicle(Action, Vehicle)

procedure **Task_SwitchToRightLane**
   if (Agent not in most right lane) and (no vehicle close in right lane)
      and (no intersection or traffic light is near by) then
      Action := Steer right

procedure **Task_OverTakeVehicle**
   if (Agent not on left of road) and (no vehicle ahead in left lane)
      and (no intersection or traffic light is near by) and
      (Parameters.PrefSpeed > Vehicle.Speed) then
      Action := Steer left and Accelerate(normal)
```

Appendix B: Glossary

ACC: Adaptive Cruise Control, functions like a normal car cruise control but also maintains a certain distance between the car and vehicles ahead.

ACL: Agent Communication Language, the name of a specific agent communication language designed by the FIPA consortium.

AGV: Autonomous Guided Vehicle, vehicle that can drive a certain route on its own, usually with the help of a guidance system embedded in the vehicle or its environment.

AHS: Automated Highway System, system for highways on which (a convoy of) intelligent vehicles can drive autonomously, intended to improve highway capacity and reduce accidents.

ALVINN: Autonomous Land Vehicle In a Neural Network, system that uses artificial neural networks to track road edges and perform autonomous robot steering.

Artificial Life: part of the field of artificial intelligence that studies natural living systems and evolutionary computing techniques. Each entity in an artificial life system is given a ‘fitness’ rating that denotes its success. The most successful entities are allowed to procreate. This ‘survival of the fittest’ should lead to a population of increasingly better entities.

Artificial Neural Network: see also ‘neural network’.

Backward chaining: deduction that the IF-part of a IF-THEN rule *can* be true, if the THEN-part is true. Note that the IF-part is not necessarily true.

BDI-agent: Belief-Desire-Intention-agent, see also section 5.5.4.

Behaviour-based robotics: all approaches to robotics that use variations of the subsumption architecture. The total behaviour of a robot is divided into separated behaviours consisting of stimulus-response relationships.

CCD Camera: charge-coupled-device camera that uses light sensitive diodes to create a picture.

DAI: Distributed Artificial Intelligence, study of distribution and coordination of knowledge and actions in environments that involve multiple entities.

Deadlock: situation in which two or more entities are stuck because they are waiting for each other to initiate the next action.

Decision tree: an object or situation described by a set of properties is modelled into a tree-graph that is used to make a decision. Each node of the tree corresponds to a test of the value of one of the properties and the branches are the possible outcomes of the test.

Dedicate lane: special lane of a road on which only limited types of vehicles are allowed.

Expert system: problem-solving system based on classical AI techniques. Expert systems are based on the idea that knowledge can be extracted from an expert human and stored in a computer.

FIPA: Foundation for Intelligent Physical Agents, international non-profit association that develops specifications of generic agent technologies to provide a high level of interoperability across applications. The target of FIPA is “Intelligent Physical Agents” – devices intended for the mass market, capable of executing actions to accomplish goals set by or in collaboration with human beings or other intelligent agents with a high degree of intelligence.

Forward chaining: deduction that the THEN-part of a IF-THEN rule must be true, if the IF-part is true.

Fuzzy logic: logic that determines the truth value of a statement as a function of the truth values of the components of the statement. Other than in ‘regular’ logic, statements can be partly true (value between 0 and 1) depending on fitting particular membership functions.

Global Positioning System (GPS): system to determine one’s position anywhere on earth with an accuracy of approximately three meters.

Inference: the deduction of new knowledge based on given rules and facts.

ISA: Intelligent Speed Adapter, system that reduces the speed of a vehicle if something goes wrong or the vehicle is driving above the local speed limit.

KIF: Knowledge Interchange Format, formal language for the interchange of knowledge between different computer programs. Based on first order logic, KIF was specifically designed to act as a mediator or interlingua in the translation of other languages.

Knowledge base: set of representations of facts and rules for deducing new facts about the world.

Knowledge representation language: syntax and semantics of a language used to store facts and rules.

KQML: Knowledge Query and Manipulation Language, language and protocol for exchanging information and knowledge between software agents.

Machine learning: study of computer algorithms that improve automatically through experience.

Macro-simulators: simulators that use a general model to abstract the total behaviour of a system.

MAS: Multi-Agent System, system in which agents interact with each other to accomplish a task, usually through cooperation and coordination of their actions. Subfield of Distributed Artificial Intelligence (DAI).

Micro-simulators: simulators that use separate entities with their own functionality (e.g. agents) to model a system.

Neural network: simplified and computerised model of a brain. A neural network consists of a number of processing units called neurons connected by links. Each link has a numeric weight associated with it and each neuron has an activation level. The weights can be modified in such a way that the network gives a specific output when given certain input. Often a neural network is organised by placing neurons in different layers. Input is received through neurons in the input-layer, output is given via output-layer neurons. The neurons between the input and output layers form the hidden layers.

Ontology: specification of an abstract and simplified view of an application domain. An ontology contains the important objects, concepts and other entities that exist in the area of interest as well as the relationships that hold them together.

Potential fields method: method used in behaviour-based robotics that generates an imaginary field representing a navigational space based on a chosen potential function. Goals are usually presented as attractors and obstacles are treated as repulsors.

Real-time system: system with well-defined fixed time constraints. Processing must be done within the time constraints or the system will fail.

SAVE: System for effective Assessment of the driver state and Vehicle control in Emergency situations, designed to intervene in situations in which the driver gets unwell, falls asleep or is otherwise incapable of safely controlling the vehicle.

Subsumption architecture: a particular type of architecture for autonomous (robotic) agents, created by Rodney Brooks, that uses several independent and parallel processes (sometimes called behaviours) that connect sensors to actuators with relatively little internal processing.

Tactical driving: that part of driving that deals with short-term reasoning (see also section 1.2).

Thread: lightweight process that shares code, data, and other resources with peer threads and/or the main program that created them. Threads usually run in parallel with other threads or the main program if the operating system allows it.

Turing test: experiment created by Alan Turing in 1950 to determine if a computer program is intelligent. A human is placed in front of a computer terminal and is allowed to ask questions via this terminal. If the human cannot distinguish between the answers given by the computer or by a human, the computer is said to be intelligent.

Variable message sign: signs beside or above roads that provide up-to-date traffic information.

Appendix C: Paper

A reactive driving agent for microscopic traffic simulations

P.A.M. Ehlert *and* ***L.J.M. Rothkrantz***
Knowledge Based Systems Group
Department of Information Technology and Systems
Delft University of Technology
Zuidplantsoen 4, 2628 BZ Delft, the Netherlands

P.A.M.Ehlert@twi.tudelft.nl
L.J.M.Rothkrantz@cs.tudelft.nl

Abstract

Computer traffic simulation is important for making new traffic-control strategies. Microscopic traffic simulators can model traffic flow in a realistic manner and are ideal for agent-based vehicle control. In this paper we describe a model of a reactive agent that is used to control a simulated vehicle. The agent is capable of tactical-level driving and has different driving styles. To ensure fast reaction times, the agent's driving task is divided in several competing and reactive behaviour rules. The agent is implemented and tested in a prototype traffic simulator program. The simulator consists of an urban environment with multi-lane roads, intersections, traffic lights, light controllers and vehicles. Every vehicle is controlled by a driving agent and all agents have individual behaviour settings. Preliminary experiments have shown that the agents exhibit human-like behaviour ranging from slow and careful to fast and aggressive driving behaviour.

Keywords: reactive agents, microscopic traffic simulation, multi-agent systems, driving behaviour

1 Introduction

In the last two decades, traffic congestion has been a problem in many countries. To reduce congestion, most governments have invested in improving their infrastructure and are exploring new traffic-control strategies. A problem is that infrastructure improvements are very costly and each modification must be carefully evaluated for its impact on the traffic flow. Computer traffic simulations form a cost-effective method for making those evaluations. In addition, traffic simulations can evaluate the improvements not only under normal circumstances, but also in hypothetical situations that would be difficult to create in the real world. Obviously, the used simulation model needs to be accurate in modelling the circumstances and in predicting the results.
Intelligent agents, which are smart autonomous computer programs, can be used to simulate the driving behaviour of individual drivers. The adaptability and flexibility of an intelligent agent make it possible to control various types of vehicles with different driving styles. Each agent is equipped with its own behaviour settings to simulate personalised driving behaviour. This way, the simulated vehicles will behave realistically and the interaction between multiple drivers can be studied.
This paper describes a model of a reactive agent that can perform tactical-level driving. Tactical-level driving consists of all driving manoeuvres that are selected to achieve short-term objectives. Based on the current situation and certain pre-determined goals, the agent continuously makes control decisions in order to keep its vehicle on the road and reach its desired destination safely.

2 Microscopic traffic simulators

Many traffic simulators that are used today are macroscopic simulators. Macroscopic simulators use mathematical models that describe the flow of all vehicles. These models are often derived from fluid dynamics and treat every vehicle the same. Only the more advanced models can differentiate between vehicle types (e.g. cars, trucks, and busses) and even then all vehicles are treated equally within one vehicle type.

In real life many different types of vehicles are driven by different kind of people, each with their own driving style, thus making traffic flow rather unpredictable. In microscopic simulations, also called micro-simulations, each element is modelled separately, allowing it to interact locally with other elements. For example, every simulated vehicle can be seen as an individual with the resulting traffic flow being the emergent behaviour of the simulation. Microscopic simulators are able to model the traffic flow more realistically than macroscopic simulators.

A Multi-Agent System (MAS) [1] can be used to form the basis of a microscopic traffic simulator. The main components (agents) of a MAS traffic simulator will be the vehicles. Every vehicle is controlled by an individual agent. Other important elements of the simulated environment can also be modelled as agents, for example a traffic light agent that controls a group of traffic lights. In 1992 Frank Bomarius published a report on such a MAS [2]. His idea was simply to model all the used objects as agents that could communicate the relevant data. Four years later two MSc students at the University of Edinburgh implemented this idea for their final MSc project [3],[4]. Their nameless text-based simulator uses Shoham's AGENT-0 architecture [5] to create multiple agents that function as vehicles or traffic lights, but also as roads and intersections. As the emphasis of their project was on creating a MAS-simulation and not necessarily creating realistic driving behaviour, all their vehicle agents use very simple rules based on gap acceptance and speed. More advanced behaviours like overtaking cannot be modelled due to the simplicity of both their agent and simulation environment.

A more advanced simulation environment is the SHIVA simulator, which stands for Simulated Highways for Intelligent Vehicle Algorithms [6]. The SHIVA simulator was especially designed to test tactical-level driving algorithms and allows fast creation of different test scenarios. In his PhD thesis Rahul Sukthankar described two different reasoning systems for tactical-level driving, called MONOSAPIENT and POLYSAPIENT, and his use of SHIVA to test the driving algorithms [7]. A drawback of the SHIVA simulator is that it needs a special SGI machine to run and is not publicly available.

At first glance, the approach we used with our driving agent resembles the POLYSAPIENT reasoning system used by Sukthankar, but its implementation is quite different. First of all, our simulator implements an urban environment. SHIVA and most other traffic simulators model highway or freeway traffic. Second, with our agent multiple behaviour parameters can be set to produce the desired driving behaviour. Most other simulators only use one or two driving-behaviour parameters (aggression or gap acceptance and preferred speed) or none at all. Third, by using relatively independent behaviour rules our agent's functionality can be expanded or altered easily and the agent can be used in completely different environments.

3 Traditional versus reactive agents

An intelligent agent is an autonomous computerised entity that is capable of sensing its environment and act intelligently based on its perception. Traditional agent architectures applied in artificial intelligence use sensor information to create a world model [8],[9]. The world model is processed by standard search-based techniques, and a plan is constructed for the agent to achieve its goal. This plan is then executed as a series of actions. This traditional approach has several drawbacks. Sensor constraints and uncertainties cause the world model to be incomplete or possibly even incorrect, and most traditional planning methods cannot function under noisy and uncertain conditions. Furthermore, in complex domains like tactical driving it is infeasible to plan a complete path from the initial state to the goal state, due to the large amount of searchable states and the inability to perfectly predict the outcome of all possible actions. The amount of possible states 'explodes' if realistic

manoeuvres such as aborted lane changes and emergency braking are taken into account. As a result a real-time response cannot be guaranteed, making the traditional planning methods unsuitable for tactical driving.
Reactive agents, also called reflex or behaviour-based agents, are inspired by the research done in robotic control. Their primary inspiration sources are Rodney Brooks' subsumption architecture [10] and behaviour-based robotics [11]. Reactive agents use stimulus-response rules to react to the current state of the environment that is perceived through their sensors. Pure reactive agents have no representation or symbolic model of their environment and are incapable of foreseeing what is going to happen. The main advantage of reactive agents is that they are robust and have a fast response time, but the fact that pure reactive agents do not have any memory is a severe limitation. This is the reason that most reactive agents use non-reactive enhancements.

4 Driving agent model

We have designed a model of a reactive driving agent that can control a simulated vehicle. The agent is designed to perform tactical-level driving and needs to decide in real-time what manoeuvres to perform in every situation. These decisions are based on the received input from the agent's sensors. After the agent reaches a decision, the instructions are translated into control operations that are sent to the vehicle.
The driving agent is modular in design. Every part can be adapted, replaced or otherwise improved without directly affecting other modules. The used parts are: several sensors to perceive the agent's environment, a communication module, a memory and controller for storing data and regulating access to the memory, a short-term planner, multiple behaviour rules and behaviour parameters, and an arbiter for selecting the best action proposed by the behaviour rules. A picture of the agent's layout is shown in Figure 30.

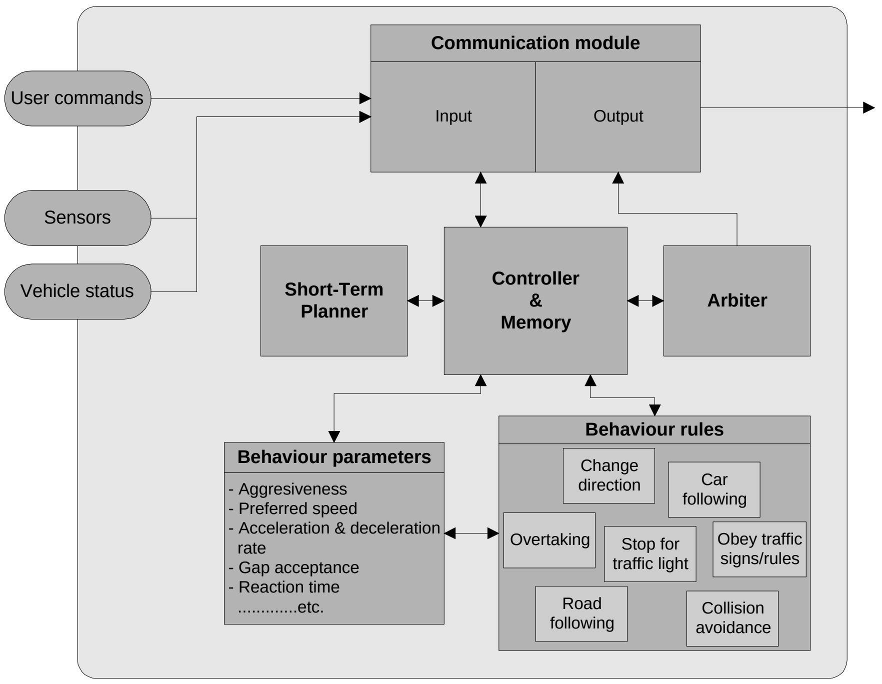


Figure 30: Driving agent layout

Our agent uses both traditional and reactive methods to perform its task, but the emphasis is on the latter since fast response times are important. Sensor information is stored in the memory and forms a temporary world model. Reactive procedures called behaviour rules or behaviours use the available information in the memory to quickly generate multiple proposals to perform a particular action. Planning in the traditional sense is not applied. The short-term planner only uses simple linear

extrapolation to calculate the expected positions of moving objects and the arbiter determines the best action based on the priority ratings of the action proposals included by the behaviour rules.
We will discuss the agent's reasoning process, behaviour rules and behaviour parameters in more detail in the next subsections.

4.1 Reasoning

The complete loop from receiving sensor messages to sending an output message to the vehicle can be seen as one reasoning cycle. The timing of a reasoning cycle and the activation of the agent's parts are done by the controller that also regulates the access to the memory. Since we want the driving agent to react in at least real-time, the agent is able to complete several reasoning cycles per second. The activation of the agent's parts is shown in Figure 31.

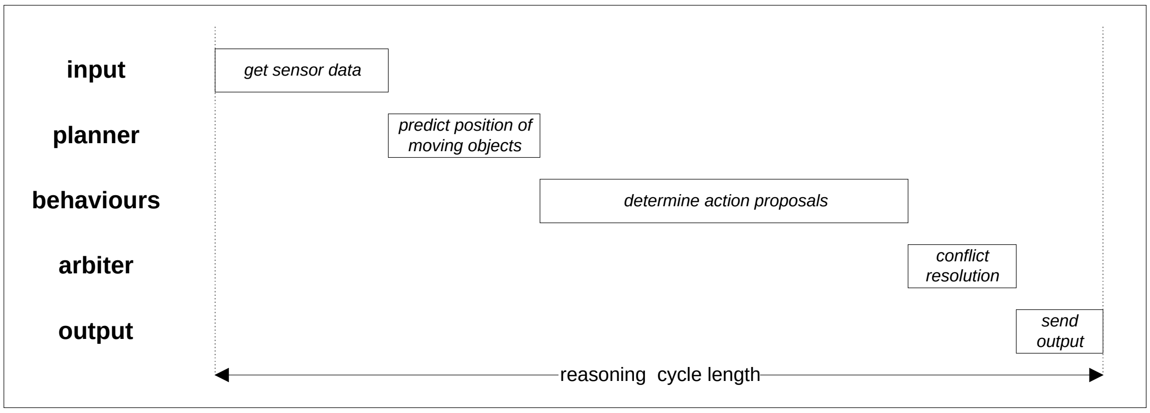


Figure 31: The reasoning cycle regulated by the agent's controller

The agent uses two types of sensor information. The first type gives information about the agent's environment, for example the distance and angle to objects, or the width of the agent's current lane. The second sensor type returns information about the agent's vehicle. This includes its speed, acceleration, heading, wheel angle, and fuel level. In addition, the agent can receive orders from the user. All information that is sent to the agent is received by the communication module that contains knowledge of the used communication protocols. When a message is received, the communication module tries to recognise the message format, the sender and its content. When the message is ok, the input section of the communication module temporarily stores it until all received messages can be written to the agent's memory. Temporary storage is necessary since one does not want data in the memory to be read and written at the same time. Outgoing messages can be sent immediately since no conflicts can arise there.
Next, all incoming messages are transferred to the agent's memory and the short-term planner makes a fast prediction of the position of all moving objects in the environment. The actual reasoning process of the agent is performed by the behaviour rules, also called behaviours. They specify what to do in different situations. Based on the available data in the agent's memory, every behaviour can propose an action. All action proposals have a tally or priority rating. The arbiter selects the best proposal based on the priority ratings and sends it to the communication module. Finally, the communication module translates the proposal to control instructions that can be understood by the vehicle.

4.2 Behaviour rules

The agent's driving task is divided into several subtasks that are automated by independent behaviour rules. This way the agent's functionality can be expanded easily without any modifications to the existing behaviours. The used behaviour rules are very much dependent of the agent's environment. Instead of a highway environment often used in traffic simulations, we have chosen to let the agent drive in an urban environment. The reason for this is that an urban environment is one of the most difficult and complex traffic scenarios. In a city a lot of unexpected events can happen and the agent has to deal with many different situations. This way we can show the potential of our driving agent concept. Note that the design of our agent does allow driving in other environments. Only the agent's behaviour rules might need to be adapted or expanded. For the city environment we designed the following behaviours:

Road following

The road-following behaviour is responsible for keeping the agent driving on the road. Besides controlling the lateral position of the agent's vehicle, based on the distance to the road and lane edges, the road-following behaviour also influences the agent's speed. It makes sure that it slows down for curves and on straight roads it will accelerate until the desired speed set in the behaviour parameters is reached.

Intersection / changing directions

If the agent is approaching an intersection, its speed is reduced, precedence rules are applied, and the agent will choose one of the sideroads. Usually, this direction is chosen randomly, but it can also be set by the user. The changing-directions behaviour can be split up into several sub-behaviours, one for each type of intersection. This is consistent with the fact that humans use different strategies to handle different types of intersections.

Traffic lights

The traffic-lights behaviour makes sure that the agent stops for red or yellow traffic lights if possible. The behaviour checks if the sensed traffic light regulates the agent's current lane and slows down the vehicle. The agent's braking start-point depends on its preferred braking pressure (deceleration rate) and is set in the behaviour parameters.

Car following

The car-following behaviour ensures that the agent does not bump into any other vehicle. If another car is driving in front of the agent, speed is reduced to match that car's speed. The precise braking pressure depends on the speed difference between the agent's vehicle and the other vehicle, the distance between them, and the set gap acceptance of the agent.

Overtaking and switching lanes

Related to the car-following behaviour is the overtaking-and-switching-lanes behaviour. If a slower vehicle is in front of the agent, it may decide to overtake this vehicle. This decision depends on the velocity difference between the two vehicles and the available space to overtake the vehicle, both in front and to the left of the other vehicle.

Applying other traffic rules

Besides traffic lights and precedence rules at junctions, other traffic rules need to be followed. Examples are, not driving at speeds above the local maximum, driving on the right side of the road as much as possible (in the Netherlands), and no turning in one way streets. Only aggressive drivers have a tendency to break some of those rules.
For this behaviour it is necessary to keep track of the traffic signs and restrictions encountered by the agent. Because the memory of the agent will clear data on a regular basis to save space, the traffic-rules behaviour needs to keep track of these signs itself, in its own private memory space. This memory space is embedded within the behaviour. Note that the behaviour also needs to keep track when the signs and rules apply. Usually, turning onto a new road will reset most of the current restrictions.

Collision detection and emergency braking

The collision-detection-and-emergency-braking behaviour is a special kind of safety measure that is activated when the agent is on a collision course with an object. At all times the behaviour needs to ensure that the vehicle can be halted before it hits the object. Actions from the emergency-braking behaviour have the highest priority and always overrule all other behaviours.

4.3 Behaviour parameters

In order to create different driving styles all behaviour rules are influenced by behaviour parameters. One of the most important (visible) factors is the driver's choice of speed. This choice has a large effect on the different driving subtasks. Drivers that prefer high speeds are more likely to overtake other vehicles than slower drivers. Another implemented factor is the distance the driver keeps to other cars, also called gap acceptance. Aggressive drivers keep smaller gaps than less aggressive drivers. A third parameter is the driver's preferred rate of acceleration or deceleration. Again, aggressive drivers tend to accelerate faster than less aggressive drivers.
Besides the above-mentioned behaviour factors, other aspects can influence an agent's driving behaviour, for example the reaction time of an agent and the range of its sensors. An agent's reaction time can be altered by changing the length of its reasoning cycle. The sensor range determines the visibility of the agent and can be used to simulate fog or bad weather conditions.

5 Implementation

We have constructed a prototype traffic simulator program to test our driving agent design. The programming language we used to build the simulator is Borland Delphi 5 Professional for NT. We have chosen this language in part since we were already familiar with it, but mainly because Delphi is an easy language, very suitable for quick prototyping.
Our simulator uses a kinematic motion model that deals with all aspects of motion apart from considerations of mass and force. The model implements smooth motion of vehicles, even during lane changes. Furthermore, the vehicles can move along realistic trajectories, but since forces are not modelled, the vehicles will perform manoeuvres without slipping.

5.1 The prototype simulator

The simulator program roughly consists of four elements: a user interface to provide visual feedback, a simulation controller, an environment containing simulated objects, and the driving agent model. The task of the simulation controller is to start, pause or stop a simulation run, and keep track of the elapsed time. The simulation controller also initialises, starts and stops the driving agents. During simulation, the controller regularly sends an 'update' order to the environment. The environment then calculates the new values for all its objects and sends relevant visual feedback to the screen. This 'simulation update' loop is shown in the left part of Figure 32. By default the update frequency is about 20 times per second, but this rate can be adjusted so that the program can be run on slower computers.
The environment is formed by all the simulated objects together. Different environments can be loaded via Map Data Files. These files contain a description of a road network and traffic control systems. Loading a Map Data File initialises the environment and data about the simulated objects described in the file is stored in the environment. Our current simulator implementation contains multi-lane roads, intersections, traffic lights, traffic light controllers and vehicles.

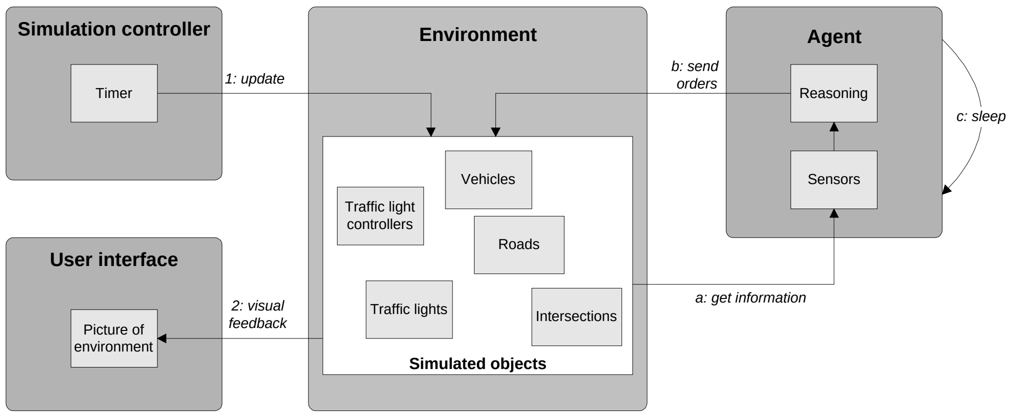


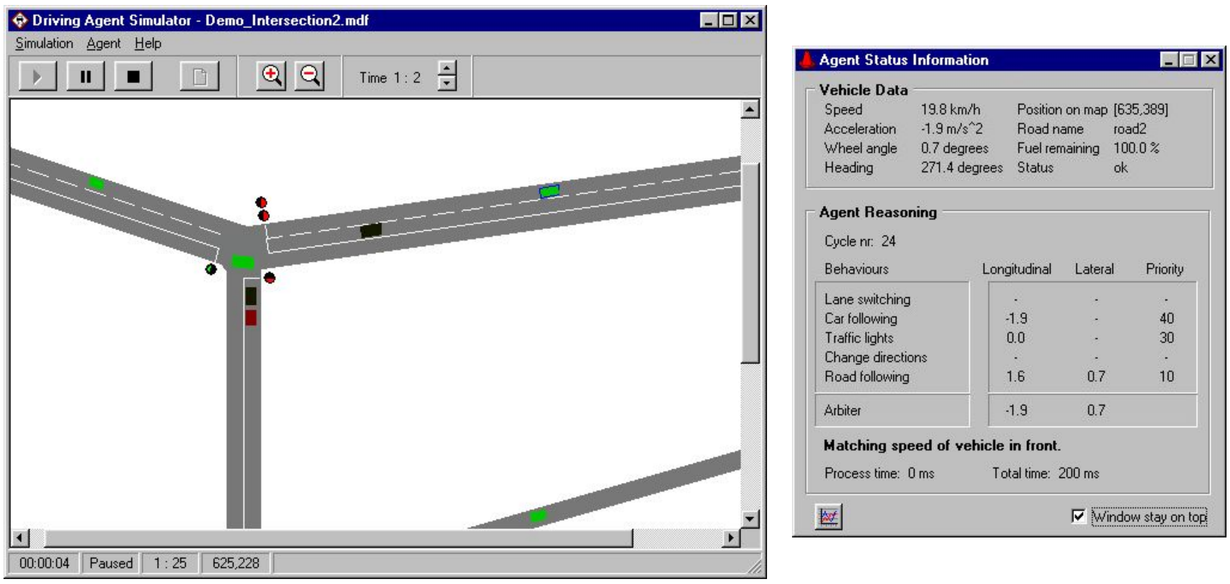
Figure 32: Simulation and agent update loops

5.2 The driving agent

Every vehicle in the environment has its own driving agent, but there is one agent that has the focus of attention and can be 'controlled' by the user. This means that the user can change the settings of this agent's behaviour parameters and can follow its reasoning process in the Agent Status Information window shown in Figure 33.

All agents are implemented as threads, which are lightweight processes started by the simulation program. The advantage of using threads is that the simulation can be faster, running threads in parallel (if the operating system allows it), and that the agents can run independent of the simulation program. The disadvantage is that there is a limit to the number of threads one can use, because the overhead in managing multiple threads can impact the program's performance. The execution loop of an agent is shown in the right part of Figure 32. If the agent finishes a reasoning cycle its thread is put asleep for the rest of the cycle time that is set by the simulation controller. This is done to prevent agents from using all available CPU time. By default the agent's cycle time is 200 ms, so the agents will perform 5 reasoning cycles per second.

The implementation of the behaviour rules is done using if-then rules. All behaviours are divided into several tasks. Tasks are executed in a serial manner, the least important task first and the most important task last. This way the important tasks 'override' the action proposals of less important tasks. The execution of the behaviour rules is also done consecutively, but in this case the execution order does not matter since the arbiter will wait until all behaviours are finished determining their action proposal.



Vehicle Data

Speed	19.8 km/h	Position on map	[635,389]
Acceleration	-1.9 m/s^2	Road name	road2
Wheel angle	0.7 degrees	Fuel remaining	100.0 %
Heading	271.4 degrees	Status	ok

Agent Reasoning

Cycle nr: 24

Behaviours	Longitudinal	Lateral	Priority
Lane switching	-	-	-
Car following	-1.9	-	40
Traffic lights	0.0	-	30
Change directions	-	-	-
Road following	1.6	0.7	10
Arbiter	-1.9	0.7	

Matching speed of vehicle in front.

Process time: 0 ms   Total time: 200 ms

[✓ Window stay on top]

Figure 33: Screen shot of the prototype simulator used to test the driving agent.

6 Results and discussion

We have presented a model of a reactive driving agent that can be used to control vehicles in a microscopic traffic simulator. A prototype simulation program was constructed to test our agent design. Although we have not validated the used parameters yet, preliminary experiments have shown that the implemented agent exhibits human-like driving behaviour ranging from slow and careful to fast and aggressive driving behaviour. The experiments were done using the first five behaviour rules discussed in section 4.2. We plan to test the other two behaviours later on.

Here we present the results of one of our experiments. The experiment consists of two different drivers approaching an intersection and stopping in front of a red traffic light. Both drivers perform this task without any other traffic present. The first driver is a careful driver with a low preferred speed, reasonably large gap acceptance and a low preferred rate of deceleration. We call this driver the ‘grandpa’ driver. The second driver is a young and aggressive driver, with a high preferred speed, small gap acceptance and a high preferred rate of deceleration. The drivers start at the same distance from the intersection. The speed of both vehicles during the experiment is shown in Figure 22.

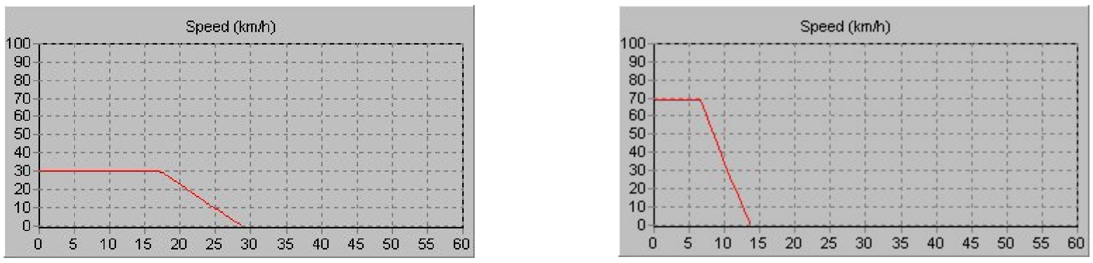


Figure 34: Speed of the grandpa driver (left) and young aggressive driver (right) during the experiment

Since the grandpa driver is driving at a lower speed, it takes a while before he starts braking, but his braking start-point (50m) is closer to the intersection than that of the young aggressive driver (65m), due to his lower speed. The difference between the used braking pressures is clearly visible. Both drivers brake with a relatively stable deceleration (approximately 0.7 m/s^2 and 2.7 m/s^2), which is consistent with human braking behaviour.
The experiment was done several times, but in almost all cases the shown graphs were roughly the same. Also, the precise stopping positions of both vehicles were approximately the same in all experiments. The young aggressive driver had a tendency to brake relatively late and often came to a stop just in front or on the stopping line. The grandpa driver on the other hand always came to a full stop well ahead of the stopping line. The stopping positions of both vehicles during one of the experiments is compared in Figure 23.

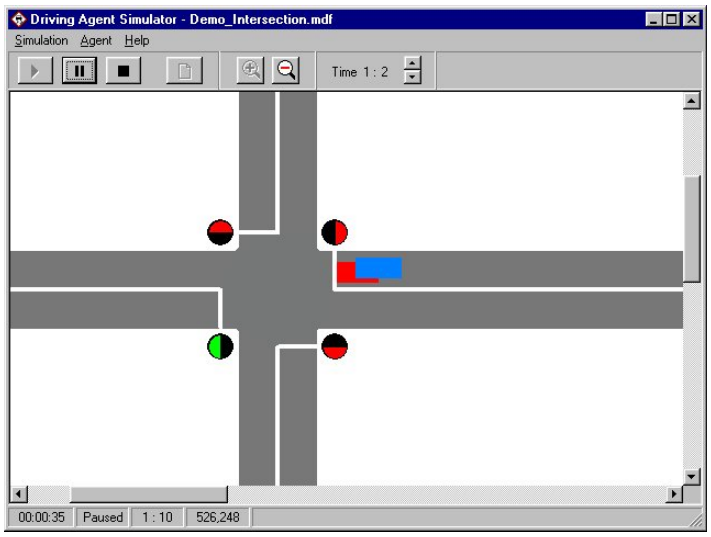


Figure 35: Compared stopping positions of the young aggressive driver (red vehicle) and grandpa driver (blue vehicle)

The aim of our simulation program was to test the design and functionality of our driving agent, but as a result its current implementation is rather inefficient since we did not optimise it for speed. Our main focus was on the correctness of the agent’s driving behaviour and reasoning process. The computer used to implement and test the program is an Intel Pentium III, 450 MHz with 64 MB of RAM, running the Microsoft NT 4.00 operating system. The help-file of our programming language recommends a limit of 16 threads (agents), assuming that most threads are waiting for external events. However, our experiments have demonstrated that at least 30 agents can run in our simulation simultaneously without any problem, so this number might be a bit low.
A drawback of our simulator is that some unrealistic assumptions were made. Agent perception is perfect. All agents have a field of view of 360 degrees and objects are not obscured or covered by

other objects. Further, vehicle actions are atomic. For example, braking at a certain rate occurs instantly after the action is sent to the vehicle. In real life this would occur more gradually. Also, pedestrians, cyclists and crosswalks are not yet modelled so the agent's ability to react to unexpected events was not yet accurately tested.

7 Conclusions and future work

The main advantage of agent-based microscopic traffic simulation over the more traditional macroscopic simulation is that it is more realistic. Instead of using general traffic-flow models, traffic becomes an emergent property of the interaction between agents. Another advantage is that agent-based simulation is more flexible. Changes to traffic scenarios can be made quickly by altering the position of individual vehicles and changing the agent's parameters. A disadvantage is the increase of computational resources and the higher number of parameters that need to be set and validated. Preliminary experiments have shown that our driving agent exhibits human-like driving behaviour and is capable of modelling different driving styles, ranging from slow and careful to fast and aggressive driving behaviour.
At the moment we are experimenting with different types of agents in several scenarios. The goal is to study the current possibilities of our traffic simulator and agent in order to improve them further. The simulation environment can be made more realistic by adding new objects, such as busses, trucks, emergency vehicles, pedestrian crossings, cyclists, traffic signs, trees and buildings. Once the simulator is improved with the new objects the agent's functionality must be extended to deal with these objects. In addition, the simulation environment needs to be validated. Although we have tried to use realistic values for vehicle acceleration, turn radius, road size etc., the used settings might prove to be inaccurate. We also need to study human driving behaviour more extensively in order to validate our driving style models. Other intended improvements are the random or stochastic appearance of new vehicles driving in from outside the environment and creating pre-determined driving styles, such as a typical commuter, a drunk driver, or a 'grandpa' driver.
The drawback of the proposed improvements will be that both the simulation environment and the agent will need more computation time and will run more slowly. Therefore, we are considering using a distributed approach in the future so that the driving agents can run on different computers. The simulation controller and environment can act as a server and the agents can be the clients communicating to the server.

8 References

[1] Ferber, J. (1999) *Multi-agent systems: an introduction to distributed artificial intelligence.* Addison Wesley Longman Inc., New York.

[2] Bomarius, F. (1992) *A Multi-Agent Approach towards Modelling Urban Traffic Scenarios.* Research Report RR-92-47, Deutches Forschungszentrum für Künstliche Intelligenz, September 1992.

[3] Chan, S. (1996) *Multi-agent Traffic Simulation – Vehicle.* MSc dissertation, Department of Artificial Intelligence, University of Edinburgh.
http://www.dai.ed.ac.uk/daidb/staff/personal_pages/mingshu/dissertations/stanleyc-dissertation.ps.gz

[4] Chong, K.W. (1996) *Multi-Agent Traffic Simulation - Street, Junction and Traffic light.* MSc dissertation, Department of Artificial Intelligence, University of Edinburgh.
http://www.dai.ed.ac.uk/daidb/staff/personal_pages/mingshu/dissertations/kenc-dissertation.ps.gz

[5] Shoham, Y. (1993) *Agent-oriented progamming.* In Artificial Intelligence 60, pages 51-92.

[6] Sukthankar, R., Hancock, J., Pomerleau, D. Thorpe, C. (1996) *A Simulation and Design System for Tactical Driving Algorithms.* In *Proceedings of artificial intelligence, simulation and planning in high autonomy systems*.

[7] Sukthankar, R. (1997) *Situation awareness for tactical driving.* Phd Thesis, Technical report CMU-RI-TR-97-08, Robotics institute, Carnegie Mellon University, January 1997. http://www.cs.cmu.edu/afs/cs/usr/rahuls/www/pub/thesis.ps.gz

[8] Wittig, T. (1992) *ARCHON: an architecture for multi-agent systems.* Ellis Horwood Limited, England.

[9] Rao, A. S., and Georgeff, M. P. (1995) *BDI Agents: From Theory to Practice.* In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, June, pages 312-319.

[10] Brooks, R.A. (1986) *A robust layered control system for a mobile robot.* MIT AI Lab Memo 864, September 1985. Also in *IEEE Journal of Robotics and Automation Vol. 2*, No. 1, March 1986, pages 14-23.

[11] Arkin, R. C. (1998) *Behavior-based robotics.* The MIT Press, Cambridge, Massachusetts.