
**Universidade de Aveiro** Departamento de Engenharia Mecânica
**2009**

**Procópio Silveira Stein**

# Infraestrutura para Condução de um Robô Autónomo usando Aprendizagem

# Framework for Visual Guidance of an Autonomous Robot using Learning

**Procópio Silveira Stein**

# Infraestrutura para Condução de um Robô Autónomo usando Aprendizagem

# Framework for Visual Guidance of an Autonomous Robot using Learning

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Automação Industrial, realizada sob a orientação científica do Doutor Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro

para Ari e para minha família

**o júri**

presidente

**Prof. Doutor José Luís Costa Pinto Azevedo**
professor auxiliar da Universidade de Aveiro

**Prof. Doutor António Fernando Macedo Ribeiro**
professor associado da Universidade do Minho

**Prof. Doutor Vítor Manuel Ferreira dos Santos**
professor associado da Universidade de Aveiro

**acknowledgements**

A lot of thanks!

To Professor Vítor Santos, for his time spent with me, for his invaluable advices at precise times, and for helping me understand that failures are part of the learning. To Miguel Oliveira, for his friendship, his immense work with the robots and for his music suggestions.

To Remi Sabino, for his mechanical skills, Jorge Almeida, Rui Martins, David Gameiro for their help and companion. To Marco Santos, for his singing skills and great help with the design of robot structures. And to everyone that in way or another contributed to create the Atlas Project.

To Professor Vitor Costa and Jorge Ferreira for believing in my capacities.

To the laboratory mates.

To robots.

**palavras-chave** navegação de robôs, aprendizagem de máquinas, redes neuronais artificiais

**resumo**

Este documento apresenta os trabalhos de desenvolvimento de uma infraestrutura de aprendizagem para a condução de robôs móveis. Este método de aprendizagem utiliza redes neuronais artificias para calcular uma direcção capaz de manter um robô dentro de uma estrada. A rede "aprende"a calcular esta direcção baseada em exemplos de condutores humanos, replicando e, de uma certa forma, imitando comportamentos.

Uma abordagem de aprendizagem pode superar alguns aspectos de algoritmos clássicos para o cálculo da direcção de um robot. No que se relaciona à velocidade de processamento, as redes neuronais artificiais são muito rápidas, o que as torna ideais para navegação em tempo real. Além disso as redes tem a capacidade de extrair informações que não foram detectadas por humanos e, por conseguinte, não podem ser codificadas em programas clássicos. A implementação desta nova forma de interacção entre humanos e robôs, que estão simultaneamente "ensinando"e "aprendendo", também vai ser destacada neste trabalho.

A plataforma de testes utilizada nesta investigação será um robô do Projecto Atlas, desenvolvido como robô autónomo de competição, para participar da prova de Condução Autónoma que ocorre anualmente como parte do Festival Nacional de Robótica. Para transformar o robô numa plataforma robusta para testes, uma série de revisões e melhorias foram implementadas. Estas intervenções foram conduzidas a nível mecânico e electrónico, e também a nível de software, sendo este último de grande importância por estabelecer uma nova infraestrutura de desenvolvimento e programação para investigadores.

**keywords** machine learning, robot navigation, artificial neural networks

**abstract**

This work describes the research and development of a learning infrastructure for mobile robot driving. This method uses artificial neural networks to compute the steer direction that a robot should perform to stay inside a track. The network "learns" to compute a direction based on examples from human drivers, replicating and sometimes even improving human-like behaviors.

A learning approach can overcome some aspects of classical algorithms used for robot steering computation. Regarding the processing speed, artificial neural networks are very fast, which make them well suited for real-time navigation. They also have the possibility to perceive information that was undetected by humans and therefore could not be coded in classical programs. The implementation of this new form of interaction between humans and robots, that are simultaneously "teaching" and "learning" from each other, will also be emphasized in this work.

The platform used for this research is one of the robots of the Project Atlas, designed as an autonomous robot to participate in the Autonomous Driving Competition, held annually as part of the Portuguese Robotics Open. To render this robot able to serve as a robust test platform, several revisions and improvements were conducted. These interventions were conducted at mechanical, electronic and software levels, with the latter having a big importance as it establishes a new framework for group and modular code development.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Among the several desired capabilities mobile robots ought to have, one of the most important is a fast response to an ever changing and unknown environment. For car like robots, this response usually comes in the form of speed control and direction of steering.

The robot used in this research is called Atlas 2008 (Figure 1.1) and it has won three times the Autonomous Driving Competition held annually during the Portuguese Robotics Open [Festival, 2009],[Wikipedia, 2009]. This Competition tries to simulate, in a simplified manner, a real car driving into a city's streets, so one vital aspect of a successful robot is to identify and quickly adapt to unknown situations, simulating real life driving.

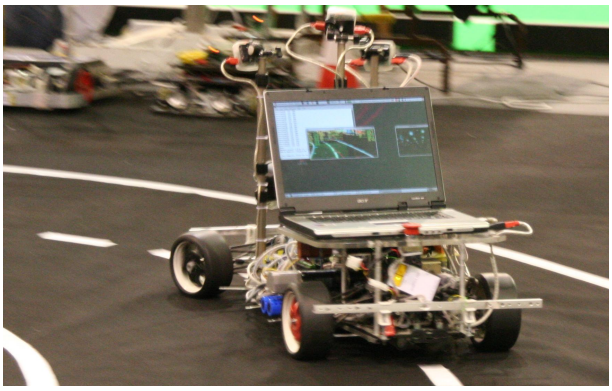

Figure 1.1: Atlas 2008 racing in Autonomous Driving Competition

This robot is not only a competition robot, but also a platform to study technologies that may be used in real cars to aid the driver. Because of that, the robot structure tries to replicate, in a simple form, the mechanisms used in real cars for steering and moving. Several research areas are related with this competition/research robot that can be implemented in real cars, as: speed control, visual signals detection, collision avoidance and others.

One important program created for the Atlas robot is responsible for the computation of a steering direction capable of maintaining the vehicle inside a road [Oliveira and Santos, 2007]. This algorithm searches for points in an image of the road and then, using a series of rules, create several possible goto points based on the geometry of the detected road. These rules that define the creation and choice of a goto point are based on the typical road width and work as on/off switches, that are triggered if a threshold is reached. Figure 1.2 shows several

candidates points calculated with this methodology.


Figure 1.2: Geometrical goto point calculation

These rules result in a deterministic algorithm that is highly dependent on the position and orientation of the cameras used to acquire the road image. Besides that, in a situation that was not anticipated by the programmer, the rules used for the creation of goto points may fail to provide correct calculations.

The objective of this research is to overcome the limitations of this steer direction computation algorithm, by trying to reproduce the behavior a human would have when driving in a road, using machine learning techniques. The most important idea behind this form of computing the steering, is that there may be unexplored visual information that was unnoticed by the programmer and therefore could not be incorporated into the deterministic algorithm.

To achieve this objective, a learning framework using artificial neural networks is proposed. In this way, it is expected that the new algorithms will be able to take advantage of previously unexplored features. Artificial neural networks are very fast and computationally efficient, two important characteristics for a vehicle navigation system. They also have the capacity to adapt and generalize to new situations, making them an interesting choice for real-world navigation.

In this framework, the artificial neural networks were trained with information that was recorded while humans were driving the robot. So, in this sense, the artificial neural networks actually learned based on human steering examples.

To prepare the Atlas robot for the implementation of the proposed learning infrastructure, several interventions were necessary. At the hardware level, the support of the cameras were replaced by an adjustable but stiff support, necessary to guarantee the position and orientation of the cameras for the data acquisition and subsequent tests.

At software level, a deeper restructuring took place, with the implementation of a modular, scalable architecture. This new architecture splits the program in several small parts that can intercommunicate with each other. This way, new modules can be incorporated in a seamless manner.

Modules can also perform redundant tasks, with different techniques being compared and even running simultaneously, each casting a decision vote to a higher level decision module, resulting in a more robust performance. This new architecture also aims better group development, as modules can be assigned for programmers who do not need to have knowledge of the whole program, but only of the assigned module.

As different fields of research are encompassed in this document, there will not be a global state of the art chapter. Instead of that, revisions and comments about previous works will

be distributed and detailed locally, along the chapters.

This document has the following structure: Chapter 2 will detail the mechanical and software improvements conducted with the robot. First, a new support for the cameras will be specified in a way that it combines versatility, providing a stable and reliable platform for sensors fixation. After that, a new modular software architecture, that aims joint group development and expansibility will be described and implemented.

Chapter 3 presents the learning framework created. This will give a description of the structure of the created artificial neural network, detailing how the information is represented. Then the created methods for data acquisition and processing are explained, followed by the description of the training process for the artificial neural networks.

After the creation of networks, tests will be conducted to evaluate the performance of the proposed learning framework. The tests will be performed with different configuration parameters to assess the impact of these parameters in the overall performance of the learning framework. The description of the tests, together with the outcome of the experiences and the analysis of their performance will be given in chapter 4. At last, chapter 5 will state the conclusions and recommend possible future lines of research.

# Chapter 2

# Robot Setup and Improvements

The Atlas 2009 robot (former Atlas 2008) was first deployed in 2005, and after all this time in competitions and exhibitions, a lot of knowledge was obtained and this know-how lead to the design of Atlas MV, a modern version of its predecessor. But with the team effort focused in building a new robot, not too many improvements were conducted in Atlas 2009.

This chapter will describe the interventions that were performed in Atlas 2009, based on previous experiences with the robot and on its performance in the Autonomous Driving Competition (ADC). The redesigns took in account the necessity to develop a reliable platform for data acquisition and testing, both in hardware and software levels.

## 2.1 Mechanical and Electronic Enhancements

During the ADC - 2008, that took place in Aveiro, Portugal, one aspect of the ATLAS design proved problematic: the camera support. During all the preliminary tests the robot was always inside the laboratory and the problem was not evident. But during the transportation of the robot to the competition place, the cameras went out of their calibrated position and orientation.

The result was that frequently the cameras position and orientation had to be readjusted in an attempt to match the previous extensively tested configuration. As the structure had no visual markings to aid positioning, all the attempts to reposition the cameras were empirical, using previous knowledge of what was desired. Besides that, only actually testing the robot in the road could confirm a correct configuration.

After any change in the cameras orientation or position, algorithms needed to be executed to merge images from two road cameras, configuring simulated distortions and overlap size (Figure 2.1). It was also necessary to store information about the road typical length after the image merge.

Figure 2.1: Road image before (left) and after (right) merging

These operations were time consuming and if the results were not satisfactory, had to be repeated. Besides that, it is difficult to rely in configurations that had not been extensively tested, as unexpected consequences might degrade the overall performance.

In this way, a lot of time was spent in something that had to be guaranteed *a priori*: a rigid position of the robot's core sensors, its cameras. Figure 2.2 is an overview of the problematic support. Note that, as the poles have a telescopic structure, each segment can rotate over the pole central axis, which will result in the rotation of the supported camera.

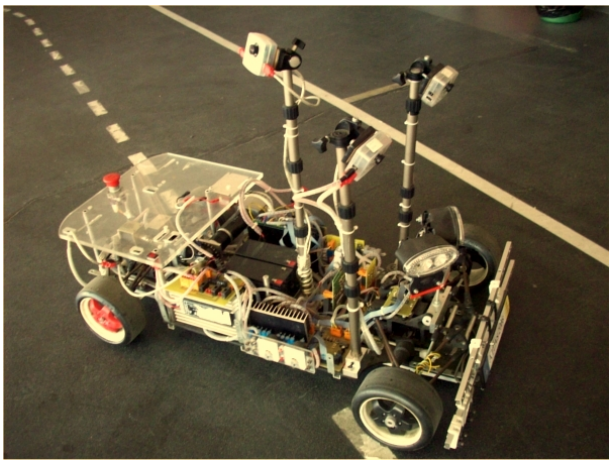

Figure 2.2: Overview of Atlas 2008 old cameras support

Figure 2.3 shows the base joint of the telescopic structure, where the handwritten markers can be seen. With this configuration, the whole structure enables each camera four degrees of freedom (Figure 2.4), but needs seven joints to accomplish that adjustment.

Another problem that arose one month before the competitions of 2009, was the malfunction of one traction wheel. The cause was that the coupling of the wheel with the engine's shaft was mostly destroyed by several years of using.

The first choice was to buy a new wheel, but unfortunately these wheels belonged to a very

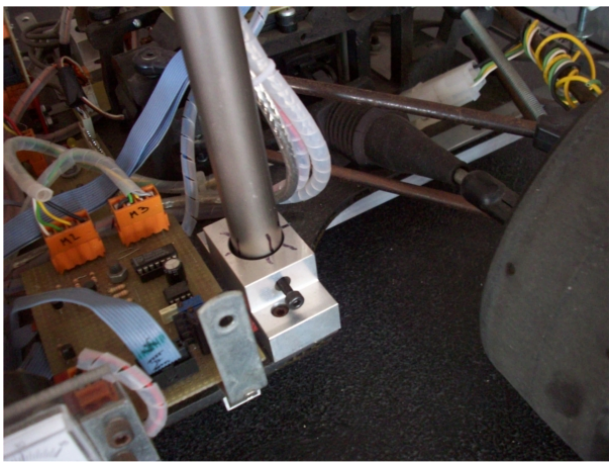

Figure 2.3: Detail of the old base support joint

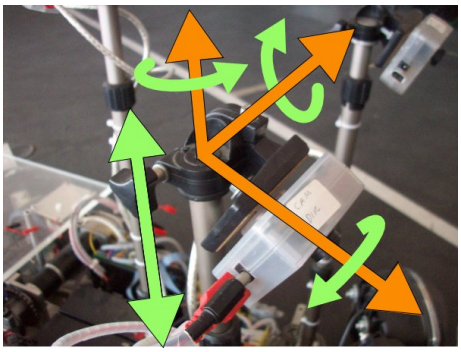

Figure 2.4: The four degrees of freedom of each camera in the old support

old F1 remote control car, and that specific scale was no longer sold. To solve this problem, a new coupler was designed, based on discussions with other team members to serve at least as a backup until the competition.

Other group of mechanical and electrical redesigns are proposed and some performed, but are not going to be detailed in this work as they are not essential for the goals aimed in this research. Nevertheless these improvements are essential to keep the robot design efficient, reliable and up to date:

- revision of the condition and wiring of the servo responsible for the steering;
- specify new tires and stock reserves;
- repair or redesign the chassis reinforcement;
- reposition the batteries, for better weight distribution and lower center of mass;
- remake all the on-board electronics to improve simplicity and protection;
- make bumpers stronger and redesign for better energy absorption;

The next section will detail the proposed solutions for the mechanical limitations and present the new manufactured structures.

### 2.1.1 New Camera Structure

The project of the new support for the vision system was based in three main requisites:

1. the joints positions should be rigidly fixed with bolts, and should have markings in all of its degrees of freedom;
2. it should be easy to adjust, dismount and mount;
3. it should be extensible to support other equipments and other sensors as a laser range finder or a stereo camera, for example.

This redesign begun with a prototype design using CAD software. This was an iterative process with constant modifications until a prototype that met all the specifications listed

above was reached. The next step was to perform a deeper analysis of the prototype, adapting and adjusting features in the original concept to simplify the design and cope with the University workshop capability. In this way, some parts were simplified while others were removed, with its functions being distributed to the remaining parts.

Also the material of the majority of parts changed from aluminium to acrylic glass (*Plexiglas*). The reason is that this material has a density of less than half of aluminium's, making all the structure much lighter.

This is very important considering that this structure reaches up to $650mm$ high, and is subjected to strong accelerations. The details and comparison between the CAD project and the real structure are shown in Figures 2.5 and 2.6.

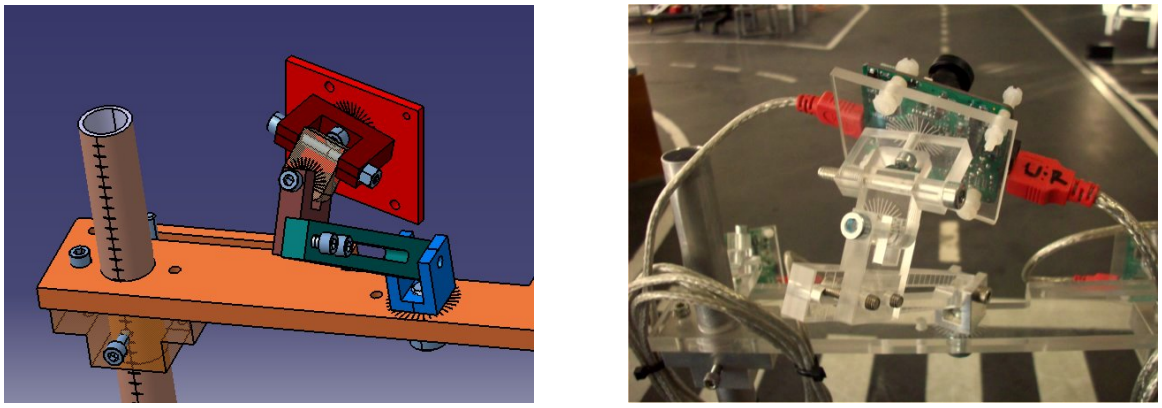

Figure 2.5: The new extensible camera support

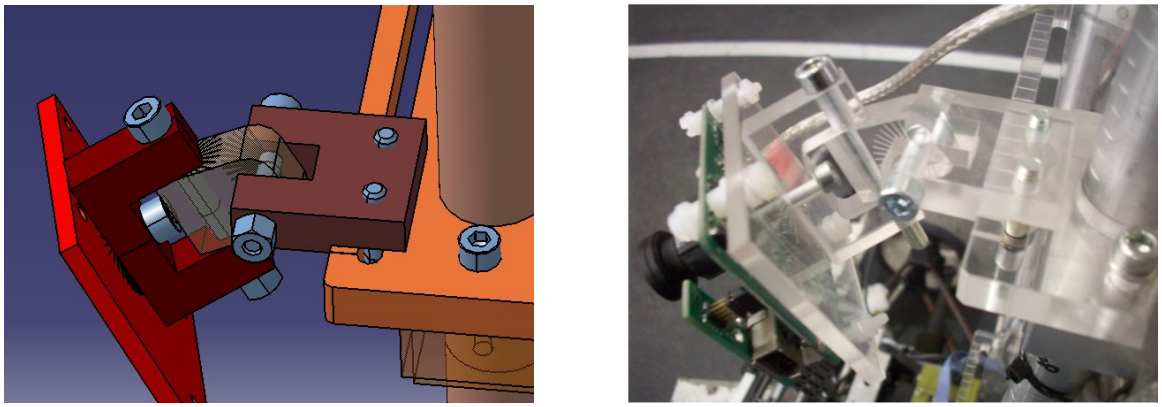

Figure 2.6: The new road camera support

The new structure was then installed in ATLAS 2009 (Figure 2.7). To validate this new installation, the cameras were positioned similarly to the older installation, then calibration routines were performed to store information about the typical road width. After that, the standard algorithm that was working with the older support was successfully tested. The main reason for this success is the robustness of the navigation algorithm regarding the form the features are extracted from the road images [Oliveira and Santos, 2007].

Furthermore, the new structure also brought benefits for the navigation algorithm. The cameras are now mounted in a higher position and with a higher angle between then, resulting

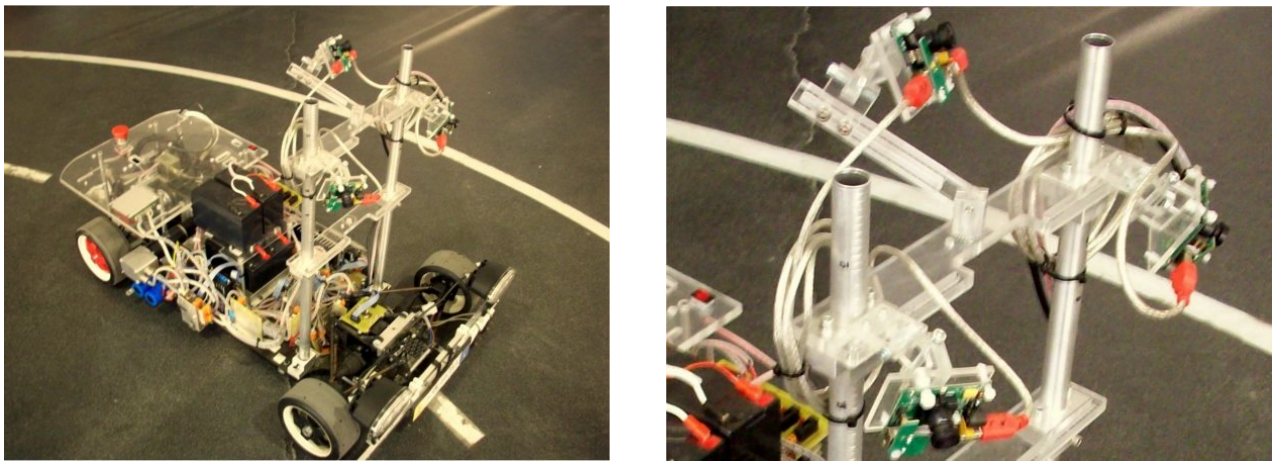

Figure 2.7: Overview of the new camera support

in a broader view of the road. Figure 2.8 shows a comparison between road images taken with the previous structure during the ADC-2008 and images taken with the new structure during lab tests.

The capability to easily unmount and mount the new structure was also tested. After successful results with the navigation algorithm, the robot was partially disassembled for a structural and electronic check-up. After reassembling the robot, the same navigation

Figure 2.8: Comparison of road images with old structure (left) and the new one (right)

algorithm was tested and the robot performed all the tasks without the need of recalibration routines, something that would be unthinkable with the previous structure.

Finally this new setup also makes better use of space, since it can support three cameras (and other equipment) using only two support poles, instead of three. Also, the road camera support has now one extra degree of freedom, responsible for lateral translation (Figure 2.9), and the camera extensible supports have seven degrees of freedom. In conclusion, the new support filled the project requirements and successfully replaced the old camera support of Atlas 2009 with several advantages, mainly stiffness and extensibility.

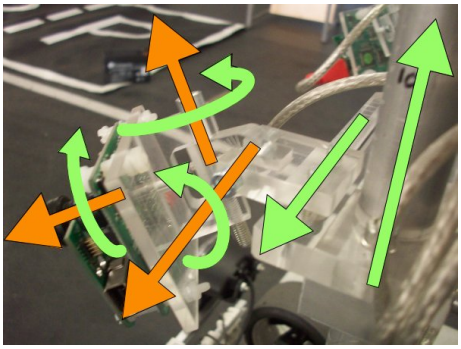

Figure 2.9: The five degrees of freedom of the new camera support

## 2.2 Software Redesign

The biggest effort in the enhancements conducted in the Atlas robot, was the software redesign. This was also a very difficult task since the previous program comprised about 15.000 lines of code and was already well experimented along several years of tests and competitions.

Albeit its success, the previous code architecture had several drawbacks. The main algorithm consisted of a single loop that was responsible for initialization routines based on a configuration file. After that, the program sequentially executed functions related to image acquisition and processing, sensors reading, motor outputs, states management, motion planning, and others.

The implemented architecture posed several problems. The single loop of sequential functions meant that new information available would only be read after the sequential loop reached the function responsible for acquiring that specific information. This meant that during motion planning functions, for example, any new information available to the robot's sensors would be lost.

If a single function failed, even only once, the whole program could stop working. It was also very hard to add or remove new code features, as the algorithm was so big that it was hard to track a modification repercussion. This last point was specially problematic for the integration of new members into the team of programmers. Also, all configuration parameters, as the camera's distortion and the robot's maximum speed, could not be reconfigure in runtime, requiring the program to constantly be restarted, and sometimes recompiled, to test different configurations.

To address these problems, a modular architecture was proposed. It was based on *CARMEN*, the Carnegie Mellon Robot Navigation Toolkit [Montemerlo et al., 2003], a col-

lection of open source modules for mobile robots that provides useful tools for new modules development and for information exchange between modules.

The intercommunication of this toolkit relies on IPC [Simmons and Apfelbaum, 1998], that provides means for connecting and exchanging data amongst processes using $TCP/IP$ sockets. It was developed for NASA and has been used in Carnegie Mellon University robots and also in the winner system of the DARPA Grand Challenge, Stanley, from Stanford University [Simmons, 2009, Montemerlo et al., 2008].

## 2.3 The Modular Architecture

The main concept behind the CARMEN Toolkit is modularity. Instead of a single and massive program, several modular programs work together and intercommunicate to achieve better performance. This is specially true in modern computers with two or more processing cores, because programs can run actually, not virtually, in parallel. In the previous architecture, the true power of multiple processors would never been used.

Due to the modular characteristic, program modules can also be spread in multiples machines (possibly with multiple cores each) in a seamless manner, creating powerful processing clusters. A possible scenario involving a mobile robot would be a computer responsible only for information acquisition, a second computer responsible for information processing and a third computer responsible for the interface with the motors, all intercommunicating with each other.

Even if this separated modules were in the same computer, the fact that there are modules dedicated to specific tasks can improve the overall performance. For example, a module dedicated only to sensor readings could work in parallel with others, making information misses unlikely to occur.

The modular architecture also decomposes the complexity of an extensive code, as each module has a simple group of tasks and are more comprehensive for new developers. The programmers do not need to understand all the robot's aspects but only a small part of the whole. There is also an increase in the robustness, as it is possible to have redundant modules, responsible for the same task, and providing the same interface type.

An example of this would be an obstacle detection module. The first module could use color segmentation and pixel count to identify an obstacle, while a second module could use specific artificial intelligence techniques to detect the same obstacle.

Both modules would create a message with the detected obstacle information, together with a degree of confidence in that detection. Then this message can be sent to a decision module that, based on the context, will make a choice. This way, even if a module fails, its "partner" can compensate, giving enough time for a coordinator module to restart the failed application.

Another very useful tool provided by the CARMEN Toolkit is a module capable of modifying configurations parameters on-the-fly (while the programs are executing). When this module starts, it reads a configuration file that has several parameters related to the robot, such as limit speed range, limit steer range, cameras port numbers, among many others.

After that, each module that is launched "asks" this server what is the value of a certain parameter. The innovative feature is that parameters can be changed after all the modules are already running, because the parameter server will "tell" every module about the modifications, so they can update the changed value. This is extremely practical during tests,

as different configurations can be experimented with immediate results, without the need of restarting the programs.

### 2.3.1 Architecture Overview

The development of an adequate robot architecture is an old issue in the robot control area. Different architectures tried to address different aspects of autonomous robots functionalities, with several advances in this area over the years. One of the first and most common architectures is the *sense-plan-act* (SPA), that follows a deliberative model. All the acquired information follow the same path in the program, in a sequential way. This is the architecture paradigm of the previous program of Atlas robot.

Another developed architecture that had significant influence in the robotic community is the subsumption architecture [Brooks, 1986]. This architecture brought the concept of reactivity to robots, creating a stack of layers with different priorities. In this scheme, an arbitrator was responsible for suppressing lower layer behaviors (as obstacle avoidance) in advantage of higher layers behaviors (opening a door, for example), according to necessity.

Modern architectures recognized the advantages and weaknesses of this two seminal architectures, and combined them in layers or tiers, giving origin to the well known three tier (3T) architecture, composed by a planning layer, a executing layer and a behavioral layer [Siciliano and Khatib, 2008].

It is hard to fit the proposed architecture in one of the presented categories. It can be perceived as a degenerated case of the 3T architecture, where the higher tier (planning) has always a constant goal, which is to complete a lap on the road.

The second tier, the executive, is able to "use" the functions of the module in the behavioral tier to achieve the goal of completing a lap in the circuit. The executive can activate or deactivate behaviors of the lower level tier, according to a state machine that anticipates the possible situations during the competition, using a methodology closely related to the work of [Rosenblatt, 1997].

Over this paradigm, the developed modules can then be grouped into two tiers, the executive and the behavioral, which can be seen in Figure 2.10.

The behavioral layer can be further divided into two subgroups of modules. The hardware interface modules are responsible for all hardware related tasks, as motors steer direction, motor speed, pan and tilt orientation, digital inputs and outputs, sound playback, telecommand inputs, image acquisition, etc. The information generated by these modules can be considered as a *raw* data, and the modules are only concerned with acquiring new data and forwarding that to others.

The information processing modules operate over the *raw* data, extracting relevant information from them and then passing this information forward using feature descriptors, which are much smaller in size. As an example, the **traffic lights detection** module receives images as inputs but outputs only information specifying what was the detected sign, drastically reducing the information size that flows among modules.

Finally, the executive tier is responsible for coordinating the behavioral modules to accomplish a task. For example, to move freely along the road, the behavioral modules related to road image acquisition, motion planning and steering are active, while the modules responsible for semaphore detection, as the semaphore image acquisition and traffic light identification, are inactive.

There are two modules in this tier: the context manager, that coordinates the correct

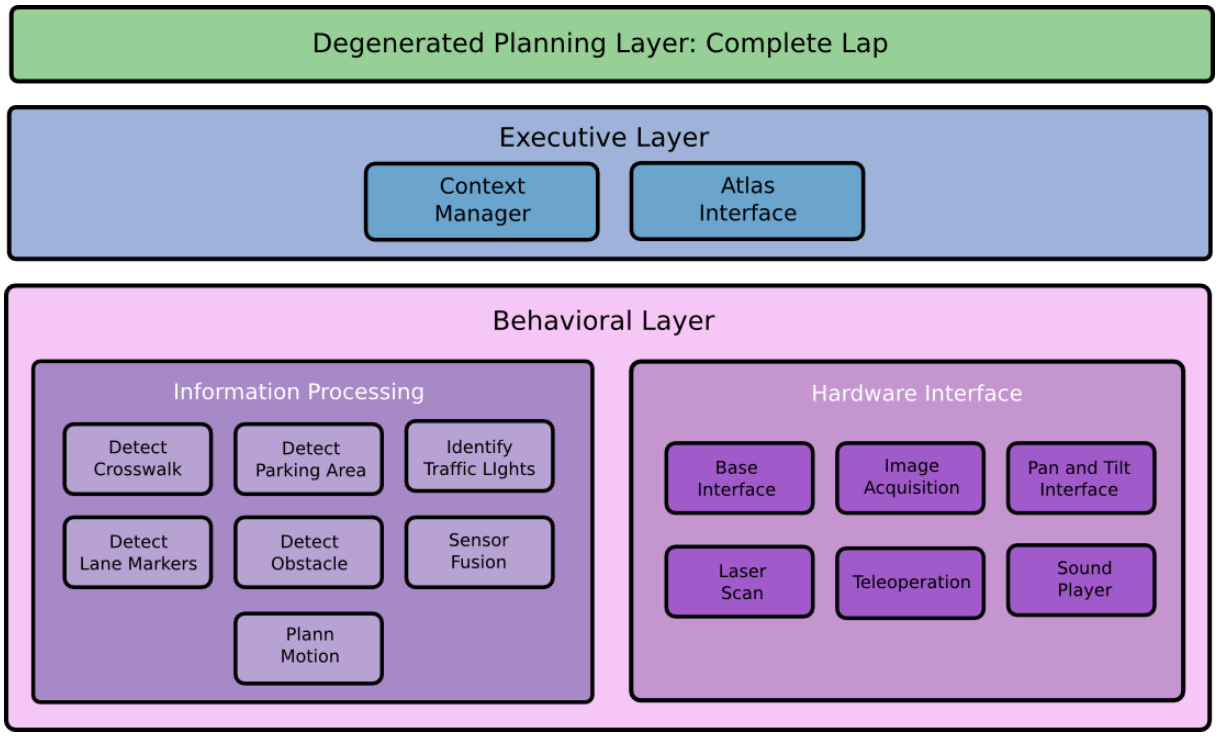


Figure 2.10: Overview of the modules categories

sequence of actions necessary to complete a lap, with the knowledge of the present context; and the interface module, that can be a debugging tool and also a module manager.

A very interesting aspect of a modular architecture, is that the form in which the modules are organized and their relations, can be freely reorganized. In this way, new architecture proposals can be easily experimented and the robot programs can be constantly upgraded. The organization of the modules presented above, is merely conceptual, an attempt to ”fit” the developed architectures in one of the several models that exist.

The true power of this implementation is its capability to be expanded, in the number of modules and in layers of abstraction, what makes possible the use of this concept in different robots and for different applications.

### 2.3.2 Information Exchange

Probably, the most important component in a modular architecture is how the information is exchanged amongst modules. This is accomplished in CARMEN Toolkit using IPC [Simmons and Apfelbaum, 1998]. It is a software package that provides several high level tools for data exchange using $TCP/IP$ sockets, so information can be exchanged between distinct computers and different operational systems.

In this architecture, the information flows encapsulated in messages. Each message is defined with a name and a structure with known fields and data types. Also, communication routines responsible for sending or receiving different messages types have to be defined. In this way, whenever a message is sent from one module to another, IPC serializes the structure of a message into a data stream, deals with all the aspects of the stream transmission and

deserializes the stream back into a message structure, at the destination module.

IPC provides two forms of message transmission, the *publish-subscribe* and the *client-server* methods. Each of them has advantages and drawbacks.

### Publish - Subscribe

In this method, modules are classified as **publishers** if they are designed to send messages and **subscribers** when they receive messages. There is also a central module that is responsible for the coordination of the information exchange. When a **subscriber** module starts, it connects to the central module and inform it what are the message types it wants to receive. The **publisher** module, in its turn, also connects to the central module and defines what message types it is able to send.

When new information is available, the **publisher** module creates a message structure and call the associated **publish** function. This message is sent to the central module that opens a connection with all the **subscriber** modules that want that message type, and forward that message to them. In the last phase, the **subscribers** store the received messages in their local variable structures. Figure 2.11 shows this exchange, the **publish** module is represented by the left column blocks and the **subscriber** by the right ones.

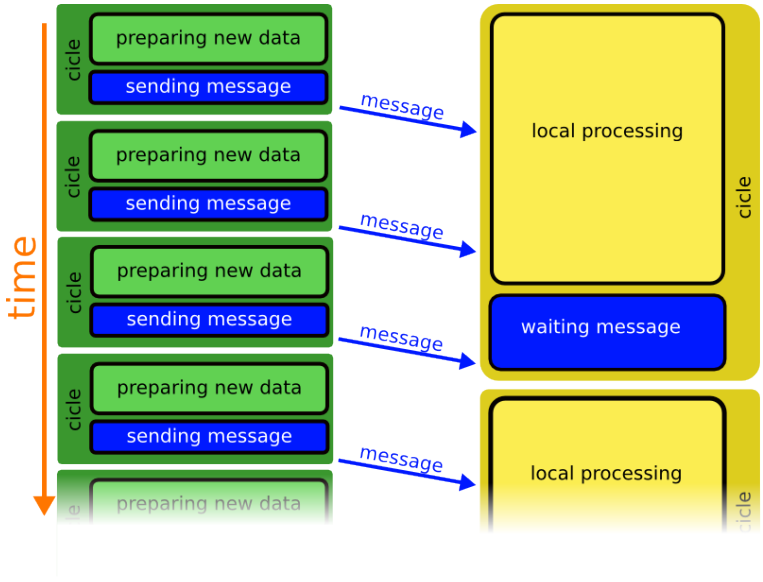


Figure 2.11: Publish - subscribe message flow

The advantage of this method is its broadcast capability, a **subscriber** module will receive a message whenever it is published, and does not need to explicitly ask for it. This is particularly important with messages that have a higher priority than others as all modules that subscribed to it will be informed of a publication and may change their behavior based on that.

The disadvantage is that sometimes a **publisher** module may have a higher frequency than the **subscriber** modules. In other words, a module can publish a message faster than the **subscriber** can process it, so messages are going to accumulate in the central module as it waits until the **subscriber** is done with its processing and is able to receive a new message.

This can lead to an overflow in the central module's buffer (specially with large messages as images or laser scans), blocking communication among all modules. Due to this limitation, this methodology will be used when the subscriber has a higher operation frequency and the messages to be exchanged are small or their priority is high.

### Client - Server

In the client - server paradigm, there is a module that receives messages, called **client** and another that sends messages, called **server** module. The main difference from the previous method is that if one module needs a certain message type, it has to actively request that message to the **server**. This method needs an additional message called *query message*, which is a request from the **client** to the **server**.

Also an auxiliary synchronization message can be used to make the information exchange more efficient: the *heartbeat message*, that is periodically sent, signalling the **client** that new information is available at the **server** side. These auxiliary messages are exchanged using the **publish/subscribe** routines. Figure 2.12 illustrates the message exchange process of this paradigm, the left column blocks represent the server module and the right column the client module.

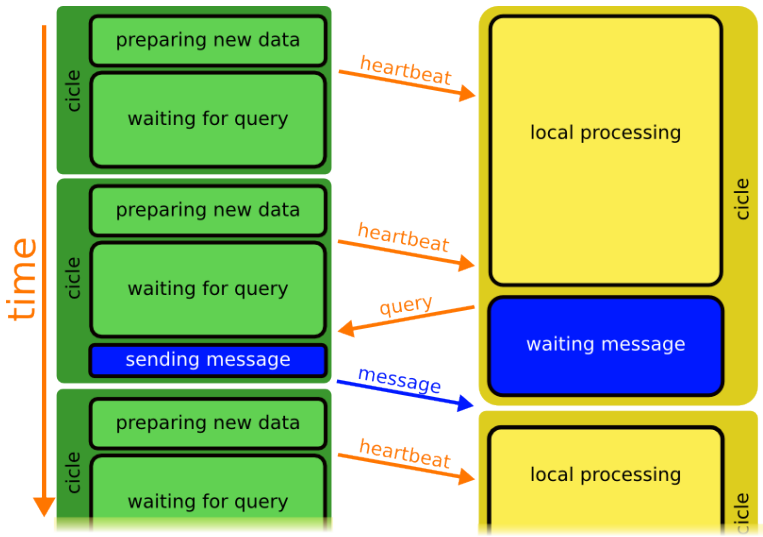


Figure 2.12: Client - server message flow

An example of a message transmission would be as follows: when a **client** module starts, it subscribes the *heartbeat message* associated with a certain message type. The **server** modules informs the central what type of heartbeat it will publish. When a message is available to be sent, a *heartbeat message* is published and the **server** waits for a fixed amount of time for a possible request (a high wait time means the **server** is more available for queries, but as a consequence, has a slower cycle).

The client will receive the *heartbeat message* and may decide to query (request) that information or not. If it decides to query, a *query message* is directly sent by the **client** to the **server** . When the **server** receives that message, it will establish a connection with the **client** and send it the requested message.

The disadvantage of the client - server approach is the higher complexity in message definitions and the existence of three messages types for transmitting a single type of information, instead of only one as in the previous method. Besides that, if the number of **clients** and the messages they are requesting are large, the **server** would spend too much time dealing with the several requests received. In an extreme situation, the time a **server** waits for requests will not be enough to address all requests from the **clients**.

The advantages are that messages will not be transmitted if no module requested them, saving precious bandwidth. It also makes much more sense to use this method in the case of a image processing modules, for example, as while the image is being processed, new image acquisitions are not important and do not need to be sent. In this way, the information exchange frequency is dictated by the receiver, avoiding a growing queue of unprocessed messages. This is going to be the method used for large messages transmission, as images and laser scans.

### Shared Memory Method

To overcome the limitations of large messages transmission over $TCP/IP$ connections, a new method was created that is based on the **client - server** paradigm together with shared memory [Oliveira et al., 2009]. The shared memory is a form of distributing information between programs with the allocation of a Random Access Memory (RAM) area that can be accessed (shared) by distinct processes.

The main difference from the previous methodology is that instead of the **server** transmitting the information message to the **client**, it only informs the **client** that new information is available at a shared memory address. This process is shown in Figure 2.13 with the **server** represented in the left column and the **client** in right one.

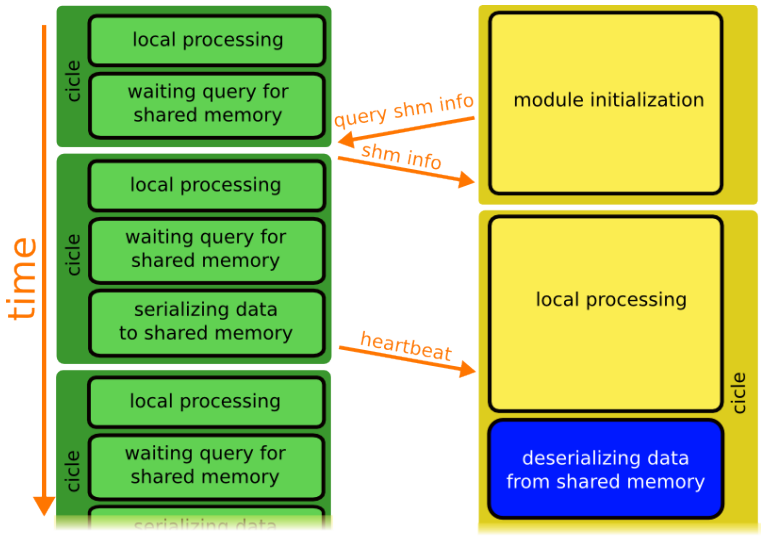


Figure 2.13: Client - server using shared memory message flow

When a **client** module starts, it checks if it should use the shared memory methodology by accessing the parameters server. If so, it will query the **server** for information about the shared memory address and size.

When new information is obtained by the **server**, it uses IPC tools to transform a message structure into an array of bytes (serialization), stores this sequence of bytes in a shared memory location and then publishes a *heartbeat message* signaling that new information of a certain type is available.

When the **client** receives a *heartbeat message*, it can access that address, read the serialized byte sequence, and then use IPC functions to recreate the message structure (deserialization).

The main disadvantage of this method is its impossibility to work over a distributed network, as the memory can only be shared among programs executing in the same machine. The positive points are that this is a very fast method (indeed the fastest of all), and large information, as images, do not use bandwidth for transmission. In this sense, this form of information exchange is preferably used when dealing with large amounts of information.

### 2.3.3 Modular Learning Schema

During the implementation of the new software architecture, more than twenty modules were created by a team of programmers, to be used with the Atlas 2009 and Atlas MV robots. However, not all of them are necessary at the same time; during teleoperation mode, for example, only two modules are needed. The created learning framework requires a total of eight modules. The overview of the modules schema and information flow is shown in Figure 2.14.

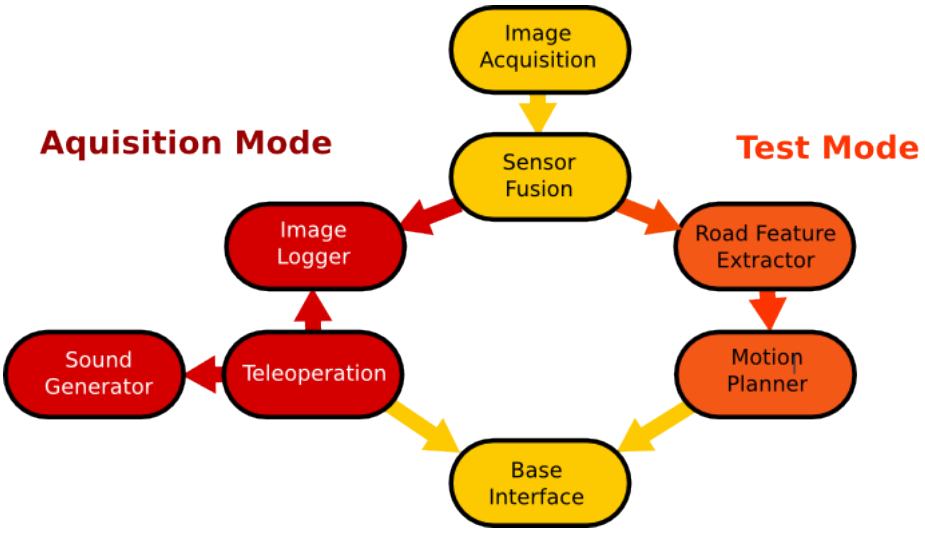


Figure 2.14: Information flowchart of used modules for data acquisition and test

Within the learning framework schema, two working modes are possible: acquisition and testing. In the first, the robot is used as a data acquisition platform. Images are acquired from the cameras and stored together with the teleoperation commands, with an audible feedback to the user.

In the test mode, the teleoperation and the image logger modules are not present, and the merged road image is passed forward to the road feature extractor module. This module, in its turn, sends a representation of the road to the motion planner module, that uses artificial neural networks to compute a steer angle. Finally, the value of the computed steer direction is sent to the base module, responsible for the steer motor interface.

Next sections will detail each one of the modules involved in this process. The graphics will follow a pattern with input messages at the left of the main block, output messages at the right side of the main block and hardware interface under the main block. This pattern is roughly based on the IDEF0 standard [IDEF0, 2009], as Figure 2.15 shows.

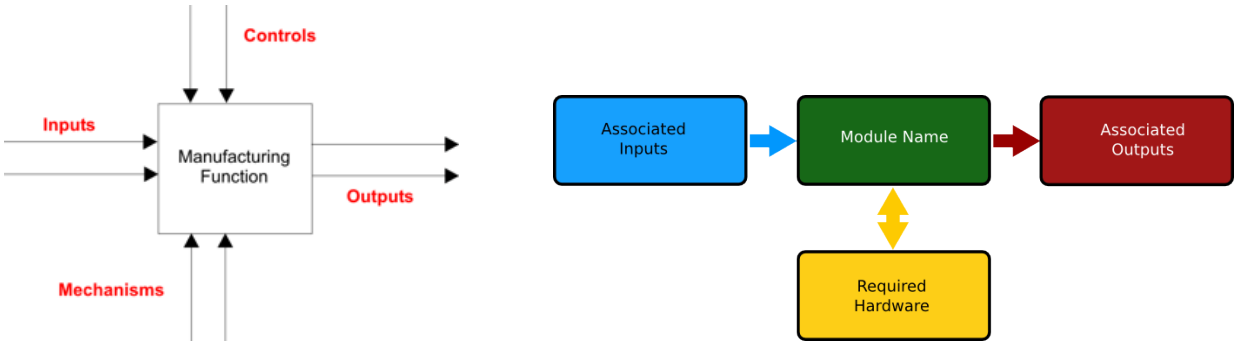


Figure 2.15: Comparison of IDEF0 standard (from [IDEF0, 2009]) and created block pattern

### Image Acquisition

This module is always operating at its maximum frequency as it is responsible for the image acquisition, which is essential for the Atlas robot navigation. As images should be distributed to several other modules, the implemented method to exchange messages is the *client/server* approach, using *shared memory*.

It can also set cameras configurations, as brightness, gain, exposure and others, and also has an option for applying (or not) undistortion corrections in the acquired images before making them available to other modules (Figure 2.16).

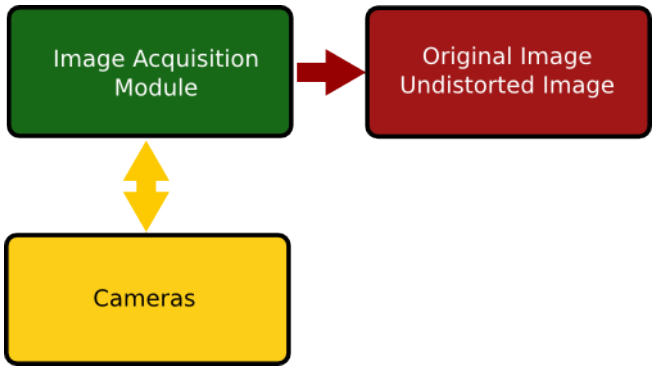


Figure 2.16: Image Acquisition module flowchart

### Sensor Fusion

With Atlas 2009 robot, the image fusion is a middle step between image acquisition and image processing (Figure 2.17). It is responsible for merging images from two cameras to form a single road image. With other robots, this module can also receive readings from several different sensors as a laser range finder, and to perform reference frame transformations of

the images. As this module works with images, it also uses the *client/server* approach, with *shared memory*.

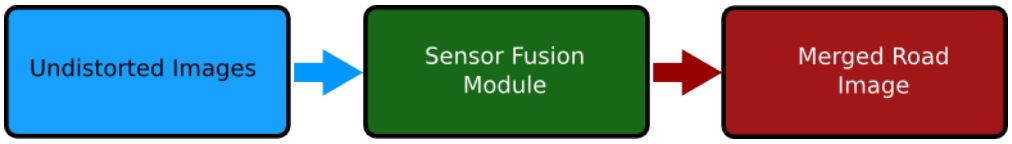


Figure 2.17: Image Logger module flowchart

### Road Feature Extractor

After receiving a merged road image, this module has the task to extract relevant information about the road (Figure 2.18). It can search for the road boundaries, or for the road middle markers, using different techniques. This is the last module that works with images in the workflow presented here. This module uses the **publish/subscribe** method to send sequences of points that represent the road limits, or boundaries.


Figure 2.18: Road Feature Extractor module flowchart

In this research, this module is responsible for extracting the features (points) that will be the input for an artificial neural network. This module can also work offline, processing images stored in the computer to create text files that are used to train the artificial neural network program. These processes will be described with more details in the next chapter.

### Motion Planner

The calculation of the steer direction of the robot is performed by this module. Although several techniques are possible to compute a steer angle, in this research this will be done using an artificial neural network. This module receives the road descriptors created by the previous module and computes a desired steer angle (Figure 2.19) using a previously loaded file that contains a trained neural network information.

The **publish/subscribe** method is used to send the computed steering to the Base module. The motion planner module that uses artificial neural networks can also be used for creating and training networks, using training data processed by the Road Feature Extractor module.

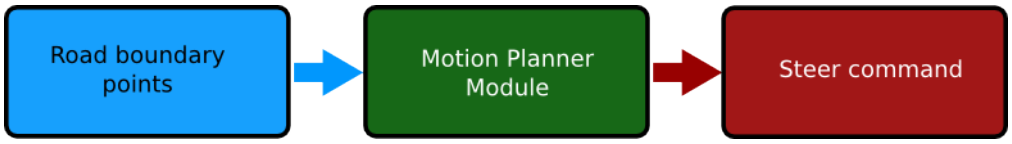


Figure 2.19: Motion Planner module flowchart

### Robot Base

It can be considered as a *driver* module, as it should adapt generic messages to the specific robot platform it is working with. For example if a message commanding the robot speed is sent to this module, it is responsible for *translating* that information to the robot protocol. Different robots should have different base modules.

This module uses the *publish/subscribe* method and is responsible for the digital inputs/outputs (IO), and for the commands of the motors related to the robot speed and steer direction. This module also constantly publishes a message with information of the current state of the motors and of the IO (Figure 2.20).

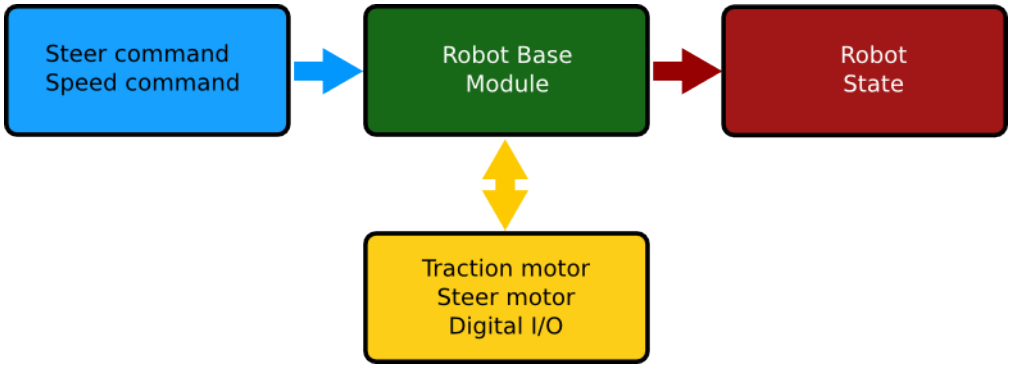


Figure 2.20: Base module flowchart

### Teleoperation

This is a very important module in the new architecture, as it is responsible for commanding the robot during data acquisition, necessary to train the artificial neural networks. Figure 2.21 illustrates this module. It serves as an interface for a wireless gamepad, which can remotely command the robot and other modules.

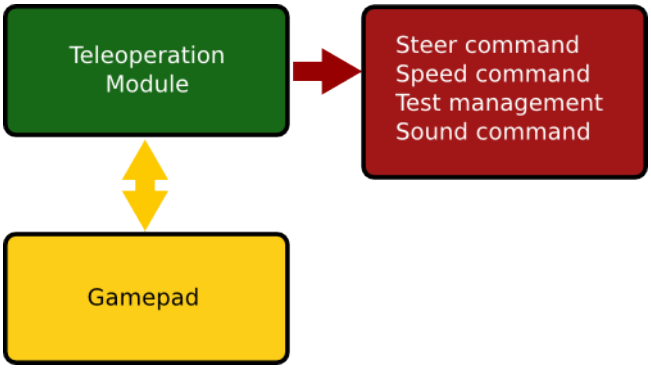


Figure 2.21: Teleoperation module flowchart

Different gamepads can be used, with the creation of similar programs that uses the same messages structures, in a transparent way for the rest of the modules. The gamepad is used to command the robot speed and steer direction, and also for managing the data acquisition

process, by remotely starting or stopping log routines. It also has a tutorial mode that interacts with the sound generator module to give an audio feedback for the user.

### Sound Generator

Although this module is not essential for the learning framework, it brings several advantages, providing audible feedback for several robot functions, specially during data logging.

This module is capable of receiving a sound command message by the *publish/subscribe* method, that specifies a sound file to be played or a text to speech, as shown in Figure 2.22. Together with the gamepad module, this functionality resulted in a very flexible form of training data acquisition that will be detailed in next chapter.

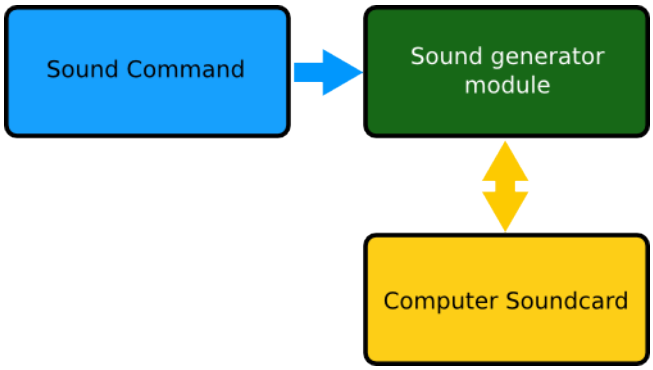


Figure 2.22: Sound generator module flowchart

### Image logger

This module's task is acquiring and storing information of the robot for further use in the artificial neural networks training. It receives a message from the Sensor Fusion module, containing a road image, and also steer and speed command messages sent by the Teleoperation module. All that information is combined and stored in an image file in the form of metadata. Figure 2.23 illustrates this data flow.

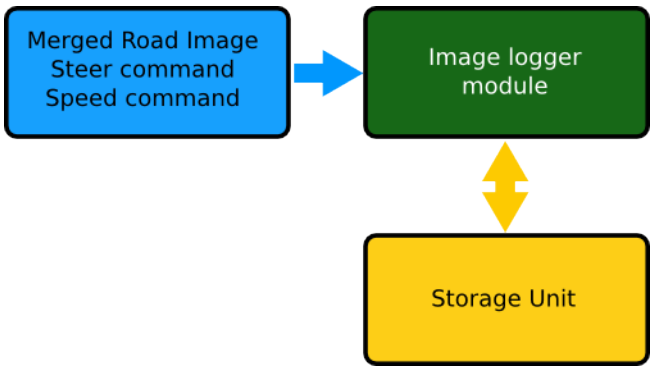


Figure 2.23: Image Logger module flowchart

# Chapter 3

# The Learning Framework

The Atlas Project robots should be seen not only as competition robots but mainly as scientific platforms. The software modularization is a step taken in this direction, aiming the exploration of Atlas possibilities by other researchers.

The proposal of a learning framework for Atlas is based in the belief that algorithms can automatically improve their performances after extracting relevant information from examples, that is not evident to humans programmers [Alpaydin, 2004].

In the navigation area, a human can choose the correct steer angle to keep a vehicle inside a road effortlessly, even in different roads types. However, it is hard for the programmer to code that expertise into a robot program. One solution would be to simplify the problem into logical concepts, such as: find what is road and what is not, then find a goto point that satisfies a criterion, as the middle of the road, for example.

This simple algorithm will very likely work, but simplifying the steer direction computation in geometric terms inevitably leads to loss of information, and this logic conditions do not reproduce human behavior at all. Furthermore, if the robot is put into a different type of road, the simple program will probably fail. The creation of new rules may embrace new road types in the navigation algorithm, but the number of possible different roads is infinitely large and classic codes can hardly cope with that variety.

With the proposed algorithms it is not the programmer, but the algorithm itself that will infer what is relevant, learning what should be the steer direction to stay inside the road, based on human examples. However, the human factor and therefore *bias*, are always present, as it is the programmer who ultimately will decide how information will be represented, what will be the learning technique and what type of computation framework will be used.

## 3.1 Neural Network Architecture for Atlas Robots

The choice of an artificial neural network as the tool for a program capable of computing a steering direction to keep the robot inside de road, was mostly inspired by Pomerleau work with ALVINN (Autonomous Land Vehicle in a Neural Network) [Pomerleau, 1991], [Pomerleau, 1995].

In that work, the grayscale levels of a $30 \times 30$ image was used as the input of an artificial neural network that calculated a steer angle to keep an autonomous vehicle inside a road. To improve training, simulated images and steer angles were generated based on the original road image and steering. This allowed ALVINN to quickly learn how to navigate in new roads.

Besides this landmark work, few was found about the specific application of artificial neural networks to autonomous vehicle steering. In [Kunzle, 2009], the author simulates a vehicle that uses neural networks to compute an acceleration in order to avoid obstacles. All the tests are simulated, with the distance from the vehicle to the obstacles always known and used as the input of the network.

A different approach [Darter and Gordon, 2005], proposes the use of groups of artificial neural networks to calculate the future lateral position of a snow removal vehicle, based on readings of several different sensors. However, no real implementation is shown in this work, and the data used to train and validate the artificial neural network is generated and not acquired in the real world.

In a more recent work [Demcenko et al., 2008], an artificial neural network is trained to replicate the form a human steers in a country-side road. Real road curvature information is used, collected indirectly with a gyroscope. The artificial neural network results are compared with collected data, but the predicted steer is not used to actually command the vehicle, so the real behavior of such net in autonomous vehicle guidance is unknown.

With the Atlas 2009 robot, no simulations were performed and no training data was be artificially created. The system used real sensors, the cameras, and provided output to real actuators, the motors. The analysis of the performance of this system will be based on results obtained in tests conducted in a real test road.

The choice of an artificial neural network most suited for the robot navigation, regarding the form of inputs, outputs and the network structure, played a major role in this research. Several attempts and different approaches were experimented, until an adequate representation of the information for the input and for the output of a network was reached.

The basic network structure used is a feed-forward multilayer perceptron, and the activation function of all the neurons is the sigmoid function (as in ALVINN). Four different network architectures were created, and then trained with two different algorithms (backpropagation and resilient backpropagation), varying the number of inputs (6 or 10) and also the size of a single hidden layers (2 or 10 neurons). The output layer always had only one neuron.

To identify the created networks, their names were composed by four fields: the first is the number of the inputs, the second is the number of neurons in the hidden layer, the third is the number of output neurons and the fourth is the training method used to create the artificial neural network.

For example, the network 6_2_1_*bp*, shown in Figure 3.1, has six inputs, two neurons in hidden layer and was trained with backpropagation algorithm. The following sections will present the steps and adaptations that were undertaken until a working artificial neural network model was created and ready to be used in real tests.

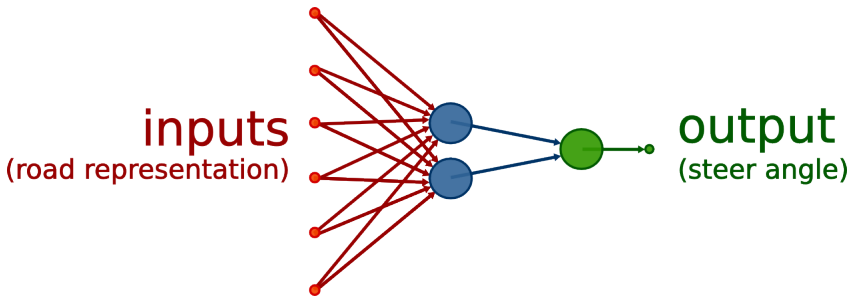


Figure 3.1: The structure of a 6_2_1 artificial neural network

### 3.1.1 Neural Networks Structure

The initial ambition was to create an ANN capable of receiving a ”raw” grayscale image from the road, and computing a desired steer direction, in the same way as the ALVINN network, created by Pomerleau. However, a direct application of this previous research was not possible in this case. The main reason for this was the very peculiar road used for the autonomous driving competition.

While in a real road image, road pixels tend to have a different color and texture from the pixels outside the road, that is not the case with competition roads. Figure 3.2 shows a typical real road image, transformed to grayscale. Note the selection on the histogram, comprised between level 70 and 130, showing approximate separation of road pixel from others, based on their grayscale levels. Also the number of points that can be classified as road pixels, are about half of the total, which means that information is distributed evenly between inside road an outside road pixels.

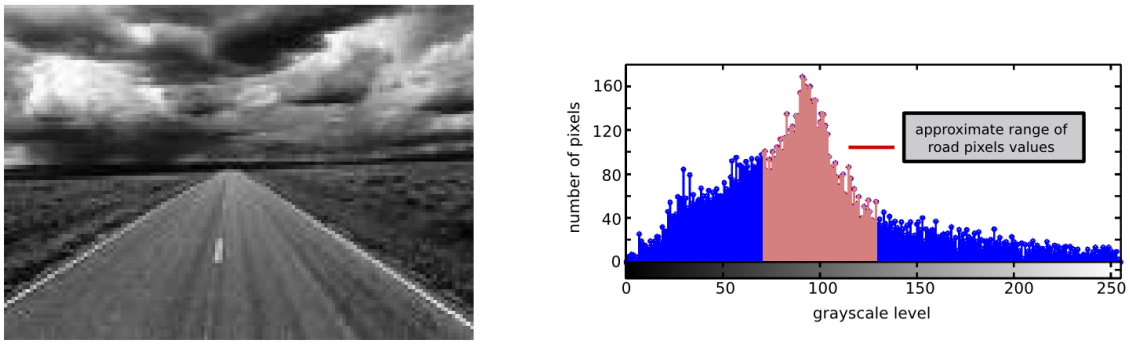


Figure 3.2: A grayscale image from a real road, and its histogram

With the competition road, the same separation - outside and inside the road - cannot be easily performed. The road is only defined with white stripes over a dark background, resulting in pixels with same color and texture, either inside or outside the road, as shown in Figure 3.3.

The relevant information in this image are the white lines that limit the inner an outer side of the road and, in this example, are comprised between gray levels of 80 and 150, shown emphasized in the histogram graph. As can be seen, the number of pixels that carry the information about the road is much smaller than the number of pixels that provide no information, merely 14% of the total in this example.

What ultimately leads for the neural network computation of the steering direction using visual information, are the pixels that define the road in an image. If the number of pixels representing this information is much smaller than the pixels that do not carry distinguishable information (for example, cannot be differenced as pixels inside or outside the road without additional knowledge), it is very unlikely that the ANN will be able to extract the relevant road information from a raw image.

To address this problem, the original proposal of computing a steer direction from a raw grayscale image was split into two problems:

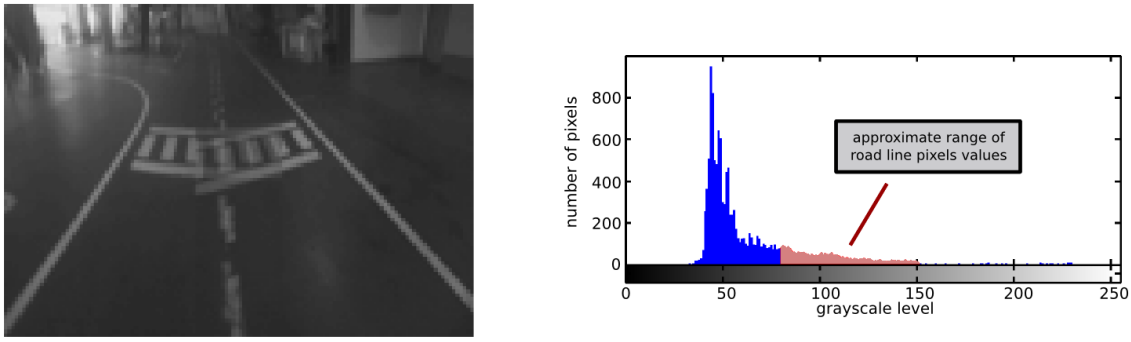


Figure 3.3: A grayscale image from a competition road, and its histogram

- the extraction of features that would contain relevant information about the road;
- the computation of the steer angle for the robot stay inside the road.

The first part of the problem was addressed with well developed techniques, that were used in previous Atlas algorithms. With the focus of the research in the computation of a steer angle to keep the autonomous robot inside the road, an artificial neural network was created, with very good results.

### 3.1.2 Input Data Representation

The input data for the network are the road boundary points. These are extracted from the road by adapting the previous algorithm (detailed in [Oliveira and Santos, 2007]) to the new modular architecture and also incorporating some improvements.

In this algorithm, a road image is formed by merging images from two cameras to obtain a broader field of view (Figure 3.4).


Figure 3.4: Merged road image in grayscale levels

After that, using threshold techniques for binarization, followed by sequences of flood-fill operations over the image (Figure 3.5), the feature extractor is capable of classifying pixels as inside or outside the road (Figure 3.6). This algorithm has been tested and used for several

years as part of the previous Atlas robot program and showed a very stable performance, under a variety of conditions.


Figure 3.5: Binary road image after an adaptive threshold operation with virtual horizon

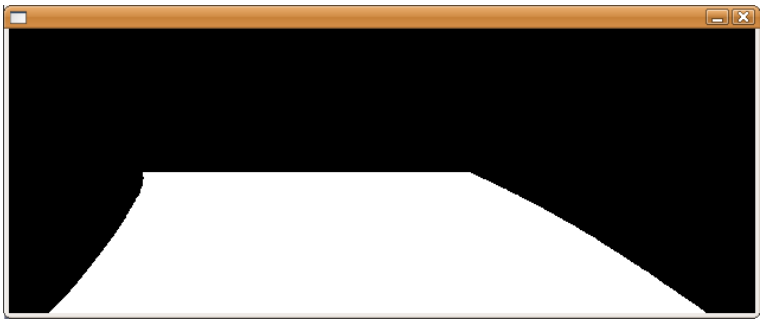

Figure 3.6: Pixels classification: inside road (white), outside road (black)

With pixels in the image classified as inside or outside road, it is possible to obtain descriptors of the road boundaries, shown in Figure 3.7. For each of the borders (the algorithm can detect none, one or two border lines) that limit the road, the user can choose the number of points to be extracted.

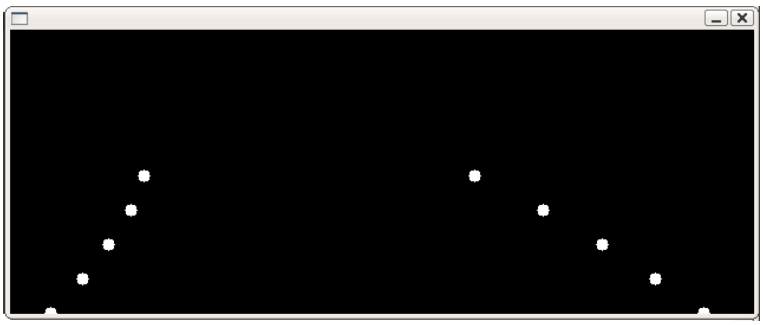

Figure 3.7: Road boundaries at five different heights

The heights of the points are equally spaced between the bottom and a virtual horizon, here set at half of the image height. The minimum number of extracted points is four, with two points at the bottom of the road lanes and two points at the virtual horizon, and the maximum, is the number of pixels that exist between the road base and the virtual horizon.

The input to the network will be a vector of the extracted points, with values that represent the position of road boundaries at constant heights, measured from the image center, in pixels (left displacements have positive values). The vertical displacement of the boundary points are not used in the inputs vectors. With this reference frame, boundary points equally spaced from the center of the image will have similar displacement values, as can be seen in Figure 3.8, where a typical input structure is presented. The input values will be further normalized by the training algorithm.

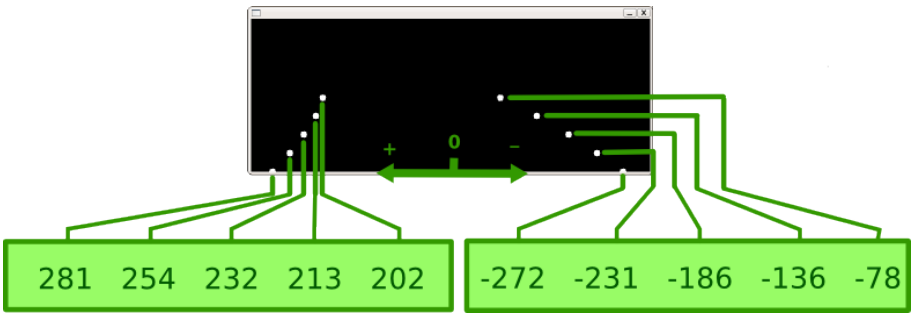


Figure 3.8: Typical ten input network vector

This program can be considered as a data preprocessor that extracts relevant information to facilitate the learning of the artificial neural network. This may sound controversial, as one of the arguments of this work was the ability of the networks to identify information that was disregarded by humans. However, this assumption is still valid as there is also unidentified information that influence the steer angle decision, and that can be identified by the network.

Also, there are plans to further substitute the flood-fill algorithm by a more advanced, probabilistic algorithm, that may also be based on machine learning to detect the road limits, but this will not be the focus of this research.

### 3.1.3 Output Data Representation

The output data was chosen to be the steer angle of the robot, measured in degrees with two decimal places. This value is measured over a reference frame that has the positive *x axis* pointing in the direction the robot is facing, and the positive *y axis* pointing toward the left of the robot. The angles are measured counter-clockwise, with angles growing from positive x to positive y, as shown in Figure 3.9. This reference frame was chosen according to CARMEN programming conventions [Carmen, 2009].

The measured angle represents the mean angle of the two front wheels of the robot, as it is an Ackermann system and therefore the wheels turn asymmetrically. The steer range is mechanically limited from approximately $-11.00$ to 11.00 degrees. The electronic command for the steer motor of the robot is done by a pulse-width modulated (PWM) signal, which is computed using a calibration table.

It is important to notice that this table is only an approximation of the real steer direction at a given signal, this is due to measurement errors and also to mechanical imperfections and worn parts, which have a significant impact in the real steer direction.

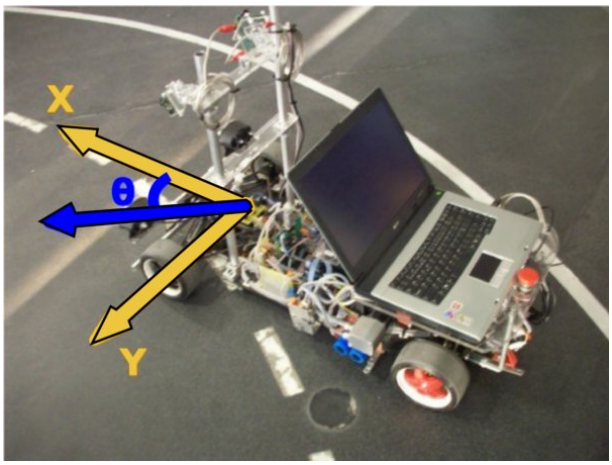


Figure 3.9: Atlas steer direction reference frame

## 3.2 Equipment and System Setup

The test platform used is the Atlas 2009 robot, with a new adjustable camera support installed. This robot has a non-holonomic base, with a servo motor responsible for the steering direction and a traction motor responsible for the velocity of the robot.

The road images are obtained by overlapping images acquired with two *Unibrain Fire-i Cameras*. The overlapped road image size is 240x630 pixels. The computer used was a Acer Travelmate 4000 laptop, with USB and IEEE1394 interfaces. Its processing unit runs at $1.6GHz$, and the operational system is an Ubuntu 8.04 32-bit. The image processing, the network training and network implementation are performed using the OpenCV library [OpenCV, 2009] with minor modifications in the source files, so the programs could output relevant information for further analysis.

This system will have two distinct working modes, one for network training and other for tests. In the train mode, data will be acquired for off-line processing and network training. While in the test mode, a trained network is actually used for the computation of the steering direction to keep Atlas robot inside the competition road.

Figure 3.10 represents a replica of the official road, which is located at the Laboratory of Automation and Robotics (LAR) of University of Aveiro. This road is limited by two white lines separated $1500mm$ over a dark background. All the experiences in this research were performed using this replica.

## 3.3 Data Acquisition

The data acquisition involves the creation of a data bank containing several **input/target** pairs that can be used to train a network. As the objective of this work is to create a program that learns how to drive by real human example, all the data used to train the network is acquired while a person is actually driving the robot, with a gamepad, along the road. No train data is artificially created. The direction commanded by the driver is the **target** for a given group of boundary point, which are the **inputs** of the network.

Figure 3.10: A representation of the laboratory road

### 3.3.1 Image Database

The decision of creating an **image database**, instead of a **road boundaries database**, is due to the fact that the images can be considered the ”raw” data of this process, meaning they represent information before processing algorithms were applied (disregarding the camera internal programs). This gives more flexibility for the research, as different forms of data representation or number of boundary points can be experimented using the same image database. Besides that, the image data bank can also be used for different tests in the future, using different methods for point extraction.

With the possibility of future exploration of the database, it is also preferable to log the most diversified information as possible, without compromising another desired aspect of the logger program, as speed, for instance. The reason is that new forms of image processing may need information that was easily available at the log moment, but was disregarded as unimportant and therefore the database can not be used.

The chosen option to store training data was by means of images with metadata (data about other data), namely using the Extensible Metadata Platform (XMP) [AdobeXMP, 2009] standard to embed information in an image. The choice of this method was based in three main aspects:

- A standard ensures that the means to extract metadata information from an image file is well known and well documented, making easy for other researches to make use of this data set;
- All the desired sensor readings of one instant are grouped together into one single file, resulting in a better management of the database;
- The XMP is an extensible format, so user-defined fields can be created.

Images from the cameras were logged with embedded information about the time it was taken, the camera parameters, and also the steer direction and robot speed. The images are stored using JPEG compression, and the metadata manipulation is accomplished with the aid of the Exiv2 Library [Exiv2, 2009]. Figure 3.11 is an example of a common image viewer

program displaying an image file's metadata which contains the speed and the steer direction of the robot at that instant and also the time elapsed since the test started.

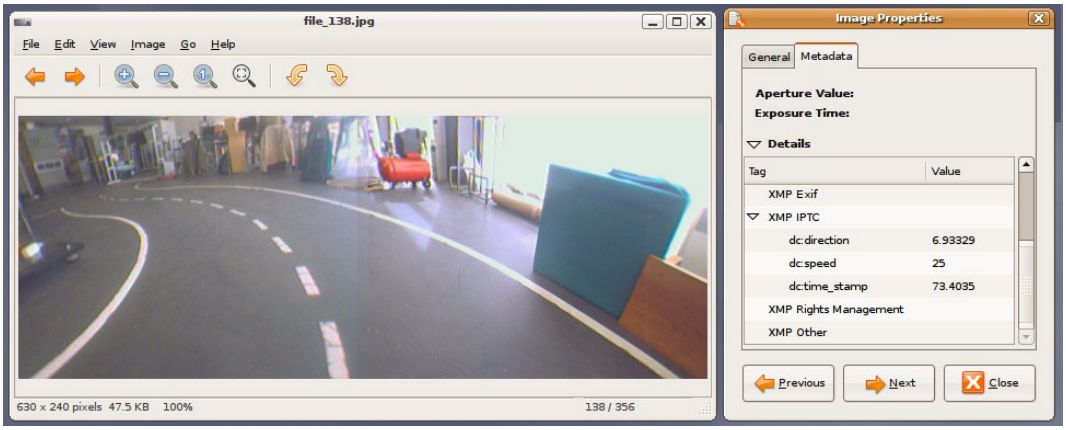


Figure 3.11: Common image viewer displaying image metadata information

### 3.3.2 Acquisition Process

The acquisition of data is performed while someone is driving the robot inside the road. The current steer direction and speed used by the driver to command the robot with a gamepad (Figure 3.12) is merged with the instant image and that information is then stored in the form of an image file with metadata. Figure 3.13 illustrates the process of the gamepad commanding the robot, at the same time that an image is acquired and stored together with the gamepad steer information.


Figure 3.12: Gamepad with steer axis and speed button pointed

For better training convergence, it is important to have a balanced training data set, that covers different situations the robot may encounter. Therefore, the acquisition process has to include situations that the robot reached after wrong decisions, as **failures are part of the learning**. If error conditions are never "shown" to the robot, it will never learn how to escape from undesired situations.

There are two main types of situations where data was acquired. The normal drive are the situations where the robot is moving approximately in the middle of the road, this will be the behavior that the artificial neural networks are expected to learn and replicate. Another

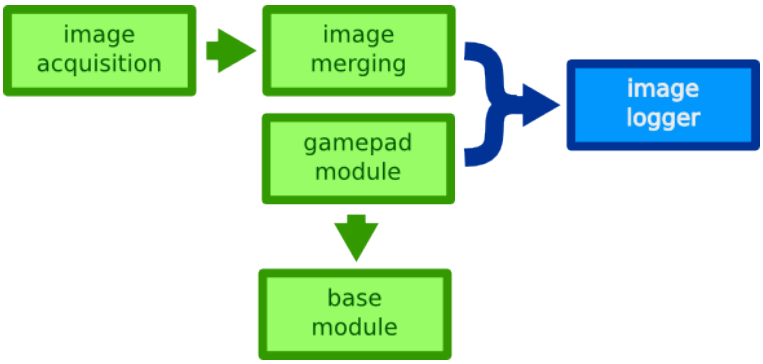


Figure 3.13: Log mode flowchart

situations are the correction maneuvers, in this case the robot will be put in demanding positions and maneuvered toward desired positions. Both situations are illustrated in Figure 3.14.

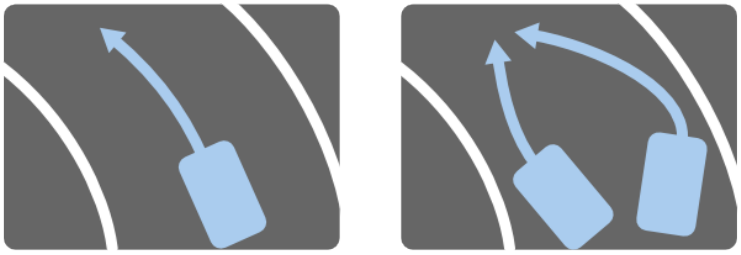

Figure 3.14: Normal drive mode (left) and correction maneuvers (right)

For example, the robot was positioned centered on the road, but facing out for both left and right sides, at that point the data acquisition was started and recorded the movements performed to bring the robot back to the middle of the road. In this situation, the start steer angle was always set at an extreme position in an attempt to create a behavior of active avoidance of undesired situations.

This was also an iterative process, with data acquisition done according to the robot performance during tests. The corrective maneuvers were based in test results and were mostly and attempt to correct certain behaviors, as an unsufficient curve, increasing the reactivity of the robot.

### 3.3.3 Human Machine Interface

Data acquisition plays a major role in neural network training, but still, it is a repetitive and tedious process. For the first training data acquisitions tests, the laptop keyboard was used to send start and stop commands for the logger program.

During the time elapsed between starting the logger and starting to drive the robot with the gamepad, the images acquired had no significant value as the gamepad, when idle, generates a 0 degree direction command. The implication of this is that for each logging test, the first and last portion of the images log had to be discarded to avoid erroneous train sets when the gamepad was not being used.

The initial framework limited successive acquisition of training images, as large amounts of time were spent for setups of small tests, necessary to guarantee that only images taken when the gamepad was active were registered. To create a better acquisition infrastructure, a simple solution was implemented: the traditional interfaces with the logger program (the laptop keyboard and screen) were enhanced with a gamepad and a sound interface.

With that, the driver has direct control over the logger program using only the gamepad, enabling remote log start, pause and stop. The sound interface gives an audible feedback for the driver, who does not need to check in the laptop screen if the logger program is active or not.

This new method completely changed the way tests were performed as it made much more easy to move the robot to a desired position, activate the log, pause, move to other position, restart the log, and so on, only using a gamepad with audio feedback. This addresses not only the problem of constantly repositioning the robot, but also the problem of initial and final images removal, as the driver can start the test while holding the gamepad, when a correct steer direction is already being commanded.

## 3.4 Training

Training will be done off-line (data is first collected and then used to train) and will use both the standard Backpropagation (backprop) and the Resilient Propagation (rprop) [Riedmiller and Braun, 1993] training algorithms. The train data set will consist in two complete laps, and several correction maneuvers in different parts of the road, always in the counterclockwise direction, in a total of 1598 input/target pairs.

The train data is created as a simple text file, according to the Figure 3.15 flowchart. In the first stage, a sequence of images are read from the image database; each image loaded has its metadata read to obtain the steer direction at the moment it was acquired.

The road limits are then extracted and the train information containing a target (the steer direction read from metadata) and several inputs (extracted road boundary points from the images) are stored into a text file, with fields separated by spaces and distributed along columns. After a sequence of images is read, the text file will have one line for each processed image.

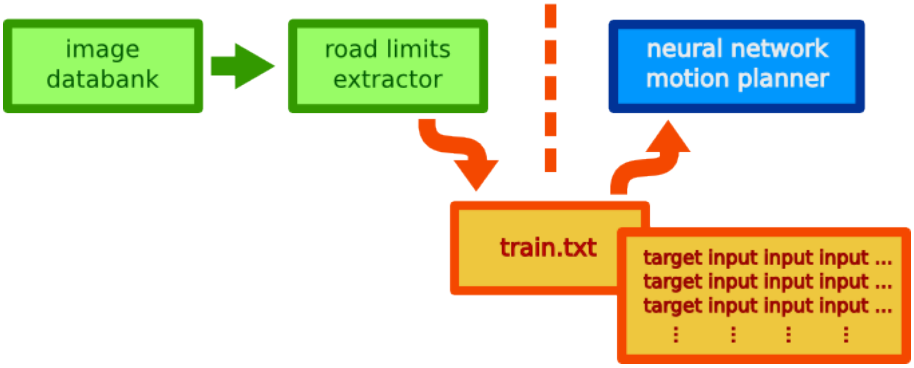


Figure 3.15: Train mode flowchart

A text file has the advantage of making easy the management of different train sets, enabling the use of a simple text editor for training sets manipulation, and also allowing the

use of other programs to access the training data set.

After a training file is created, which may be a composite of several train types (normal drive and correction maneuvers), the artificial neural network program loads the training file, and then starts the training using two termination criteria: the maximum epochs number (number of weight updates) and a performance goal, based on the mean squared error of the outputs.

The train process is also an iterative and empirical process. One of the limitations found is the impossibility to validate a network train without actually testing it with the robot. As a performance estimation after the training stage, sequences of input and desired target can be presented to the trained neural network to measure how close the computed steer direction is from human steer direction at that moment.

However, this is not a very precise form of assessing the trained network performance because, differently from real tests, the computed steer direction will not affect the acquisition of the next image. Analysis of stand-alone road images also fails to provide a reliable performance measurement as the effect of the computation of a wrong steer angle that lasts only a tenth of a second will probably not degrade the overall performance.

Figure 3.16 illustrates how is the steer computation process during real navigation with artificial neural networks. It can be represented as a closed control loop with visual feedback, where the classical control reference does not appear, as it can be considered inherent to the artificial neural network behavior in this model.

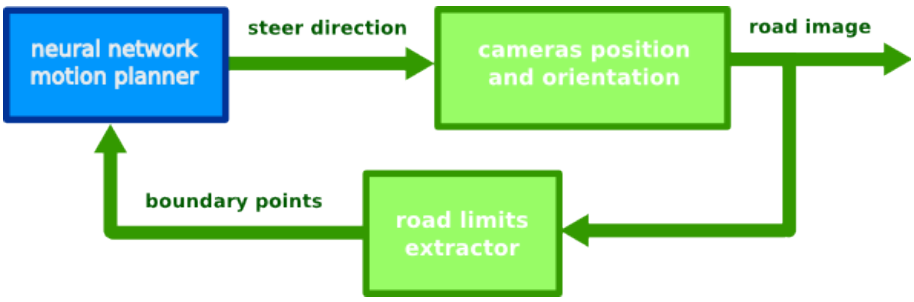


Figure 3.16: Steer direction control loop, using visual feedback

To simulate a test of the neural network after training, a model that substitutes the *cameras position and orientation* block would be necessary. However, the creation of such model will not be developed in this research due to its high complexity: it should take into account a prior knowledge of the road, a model of the robot motion according to a steer direction and speed, how that motion would affect the next image acquisition, mechanical imperfections, image reference frame transformations, lens distortion and others.

Another possible method for the training evaluation could be a geometrical ideal steer angle, that would be constant along curves and straight lines. This is also not feasible as the calibration of the robot steering was not very precise and worn mechanical parts deteriorated even more a correct steer angle measurement for the robot. For example, while driving in a straight line the ideal steer angle would be 0 degree, however in real tests it is necessary to command a steer angle of about 3 degrees for the robot to keep a straight direction.

Besides that, the human driver seems to have an unstable and coarse steering behavior while using the gamepad. For example, if the robot is not going straight when the gampepad is idle, a human usually makes punctual corrections instead of maintaining a constant correction

angle. Or during curves, humans may turn more or less the steer axis, to keep the robot centered on the road. This behavior can be seen in Figure 3.17, where the human driver has almost discrete-like levels for steering during a constant curve, this values will be used as network output targets during the training process.

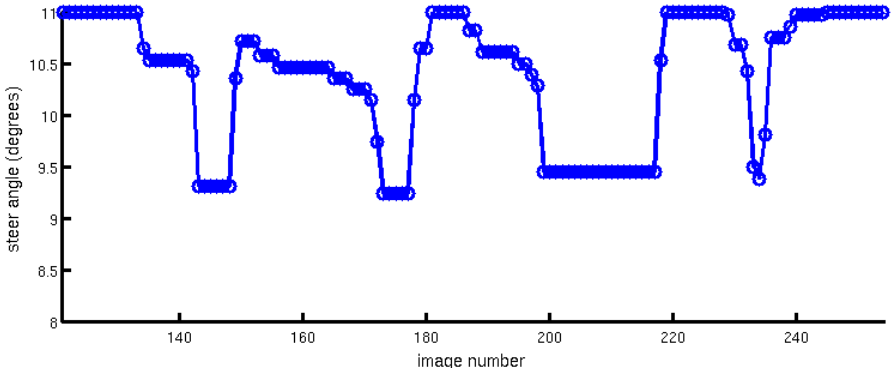


Figure 3.17: Detail of steer angle commanded by a human driver during a constant curve

The trained artificial neural networks are stored in a YAML (**Y**AML **A**in't **M**arkup **L**anguage) file [YAML, 2009]. It contains the values of the connection weights between neurons, information of input/output scale, training parameters and algorithms, network structure and others. Figure 3.18 shows an example of a neural network YAML file.

```
%YAML:1.0
my_nn: !!opencv-ml-ann-mlp
   layer_sizes: !!opencv-matrix
      rows: 1
      cols: 3
      dt: i
      data: [ 6, 2, 1 ]
   activation_function: SIGMOID_SYM
   f_param1: 1.
   f_param2: 1.
   min_val: -0.95000000000000000
   max_val: 0.95000000000000000
   min_val1: -0.98000000000000000
   max_val1: 0.98000000000000000
   training_params:
      train_method: RPROP
      dw0: 0.10000000000000000
      dw_plus: 1.20000000000000000
      dw_minus: 0.50000000000000000
      dw_min: 1.1920928955078125e-07
      dw_max: 50.
      term_criteria: { epsilon:0.0199999995529652, iterations:5000 }
   input_scale: [ 0.0354268648719219, -10.0851433119989888,
       0.0213655225561634, -4.9282781935247586, 0.0140405281383082,
       -2.4787114966473225, 0.0304893558281898, 8.1051475478671797,
       0.0287171355150313, 5.6005247386118091, 0.0182746445227470,
       2.1874532074663651 ]
   output_scale: [ 11.57894736842105531, 0. ]
   inv_output_scale: [ 0.0863636363636364, 0. ]
   weights:
      - [ 0.3648584823669642, -0.8809498939078735, -0.5896259281772361,
          1.9746108684626036, -0.4204595702281466, 1.2956634782342265,
          -1.0268746350999354, 3.1657642201322513, 1.5149014999909467,
          -4.5097194769268425, 0.3747869379907624, 1.5228273856667360,
          0.2118073273130010, -0.3288834765694164 ]
      - [ 4.2916883894622666, 2.1010640782790326, 0.8634301445417745 ]
```

Figure 3.18: Example of YAML file containing neural network information

### 3.4.1 Backpropagation Training

The Error Backpropagation (backprop) is the most common training method for artificial neural networks. In this method, the adjust of training parameters, namely the learning rate ($\eta$) and the momentum ($\alpha$), are performed empirically and are critical for the convergence of the network, as they define how much weights ($w$) will change, according to the weight update equation:

$$\Delta w(t) = -\eta \nabla error(t) + \alpha \Delta w(t-1)$$

This requires several experiences to find an appropriate set of parameters that results in the training convergence with minimal number of iterations. For the sake of simplicity, not all combination of these two parameters are detailed here. The best results were attained using a momentum of 0.5 and a learning rate of 0.01.

The backpropagation training algorithm used in this research performs incremental learning. This means that the weights are updated for each training pair of input/target of the current training set. Here the number of epochs means the number of times the algorithm sweeps across the entire training set using each one of the entries. In other words, for 1000 epochs and with a training set of 1598 input/target pairs, the training algorithm will perform 1598000 weight updates.

The performance of the training is measured as the mean squared error (MSE) of the network outputs in relation to the training targets. Two criteria may finish the training process: the maximum number of epochs, set at 5000; and the MSE goal, defined as an error of 2% in relation to the targets, or $20.00 \times 10^{-3}$.

In the following description of the tests, if a network does not reach the goal before 5000 epochs, two performance graphs will be shown for better understanding. One of the initial epochs (until epoch 500) using the default scale and other of the final (500 to 5000) epochs, with a zoomed scale.

The training performance graphs are presented from the most complex artificial neural network, to the simplest. Figure 3.19 is the performance of the network 10_10_1_*bp*, which shows a fast convergence in initial epochs. As Table 3.1 shows, this network successfully reaches the MSE goal on the epoch 2290.

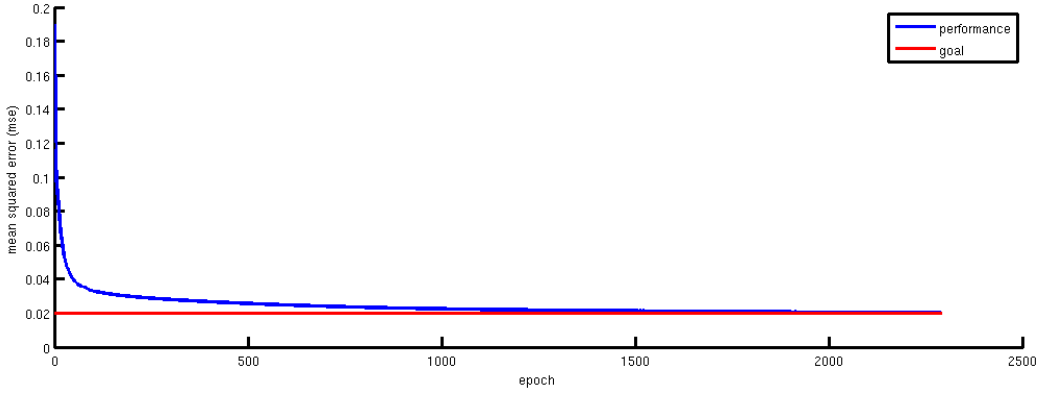


Figure 3.19: Training performance of ann 10_10_1_*bp*

| 10_10_1_bp | |
|---|---|
| training time (seconds) | 6.94 |
| number of epochs | 2290 |
| final mean squared error | $19.97 \times 10^{-3}$ |

Table 3.1: Training results for ann 10_10_1_*bp*

Artificial neural network 6_10_1_*bp* has its performance shown in Figure 3.20 and results in Table 3.2. In the contrary of the previous net, the goal is not reached, so the performance graph is split in two parts. The first one displays the fast initial convergence and the second shows, in a different scale, a slower but continuous convergence to the established goal, that would probably be reached if more epochs were given.

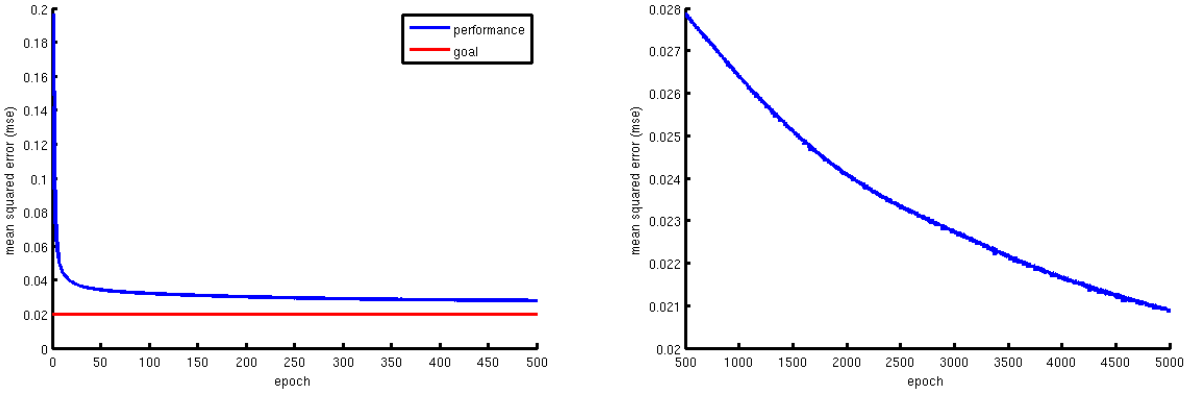


Figure 3.20: Training performance of ann 6_10_1_*bp*

| 6_10_1_bp | |
|---|---|
| training time (seconds) | 14.15 |
| number of epochs | 5000 |
| final mean squared error | $20.86 \times 10^{-3}$ |

Table 3.2: Training results for ann 6_10_1_*bp*

Figure 3.21 shows the performance of network 10_2_1_*bp*, which also fails to reach the established goal, and has a slower convergence in initial epochs. The right graph shows in a different scale that the training performance seems to become asymptotic and the defined goal probably would not have been reached. Table 3.3 shows the training results.

| 10_2_1_bp | |
|---|---|
| training time (seconds) | 9.94 |
| number of epochs | 5000 |
| final mean squared error | $33.61 \times 10^{-3}$ |

Table 3.3: Training results for ann 10_2_1_*bp*

The simplest network, 6_2_1_*bp*, trained with backpropagation has an interesting perfor-

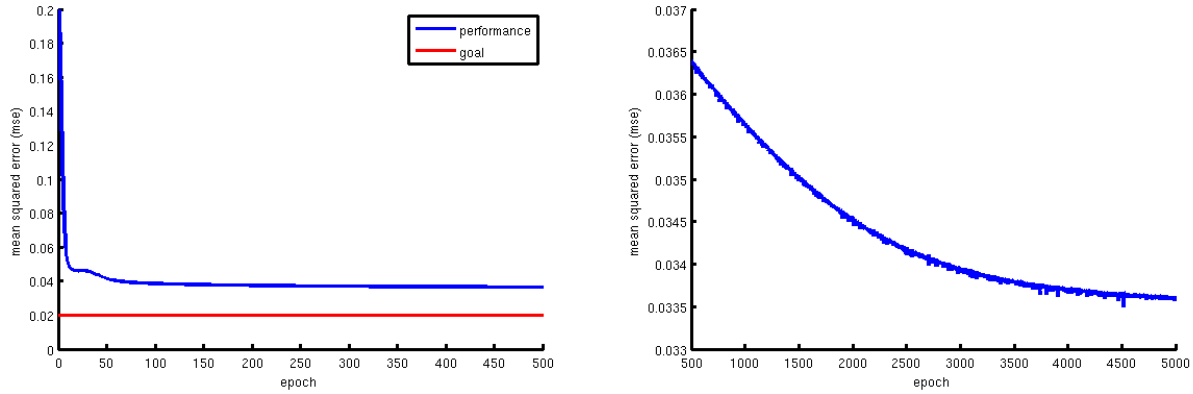


Figure 3.21: Training performance of ann 10_2_1_*bp*

mance graph, shown in Figure 3.22. The first graph shows an oscillatory convergence in the initial epochs, but the second graph clearly shows that the network had reached an optimal MSE at about epoch 500, and as the training proceeded the performance worsen with the number of epochs, also becoming asymptotic in the final epochs. The training results are shown in Table 3.4.

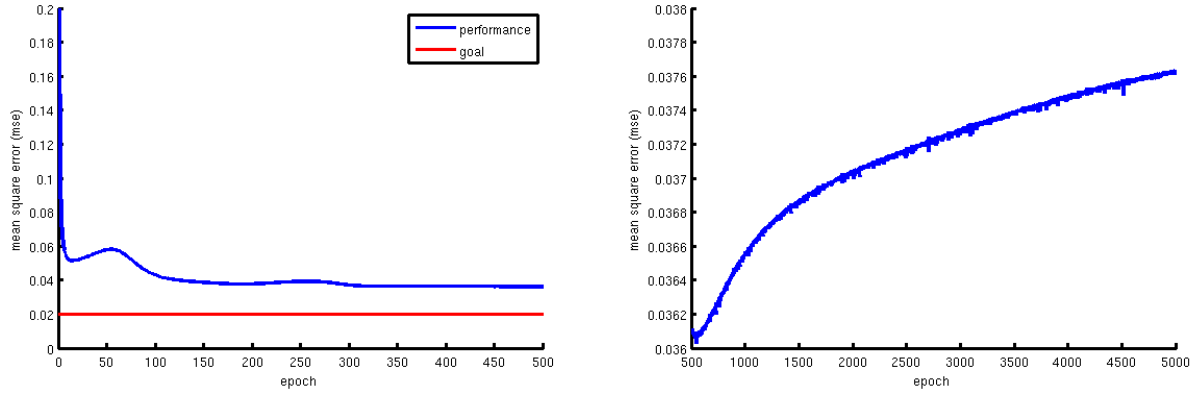


Figure 3.22: Training performance of ann 6_2_1_*bp*

| **6_2_1_bp** | |
|---|---|
| training time (seconds) | 9.36 |
| number of epochs | 5000 |
| final mean squared error | $37.64 \times 10^{-3}$ |

Table 3.4: Training results for ann 6_2_1_*bp*

### 3.4.2 Resilient Backpropagation Training

The Resilient Backpropagation (rprop) method uses an **adaptive learning rule** to change train parameters while the training is occurring [Riedmiller and Braun, 1993]. The adaptation

of the training parameters is based only on the sign of the weight's partial derivative (and not on the size), and performs local adaptations of the weights, usually resulting in faster convergence.

One of the greatest advantages of this method, is that, due to its adaptive nature, the tedious task of trying different combinations of momentum and gradient is unnecessary [Sarle, 2009], and although it still allows different parameter configurations, modifications have little impact in the convergence of the training.

Different from the previous training method, the resilient propagation uses batch learning. In this case, the weights are updated only after the whole training data set is processed, which results in faster training and improved convergence. The number of epochs are the number of times the weights are updated and are unrelated to the size of the training data set.

As on the previous section, the termination criterion will be the maximum number of epochs, set at 5000, and the MSE goal, set at 2%. Performance graphs will be split, for better visualization, if the network fails to reach the MSE goal before 5000 epochs. It is important to have in mind that the graphs that are divided in two, have completely distinct scale, so that information should be evaluated with care.

With the resilient propagation technique, network 10_10_1_*rp* reaches the defined MSE goal almost 20 times faster compared to the backpropagation training, as shown in Table 3.5. Figure 3.23 also show the very fast convergence.

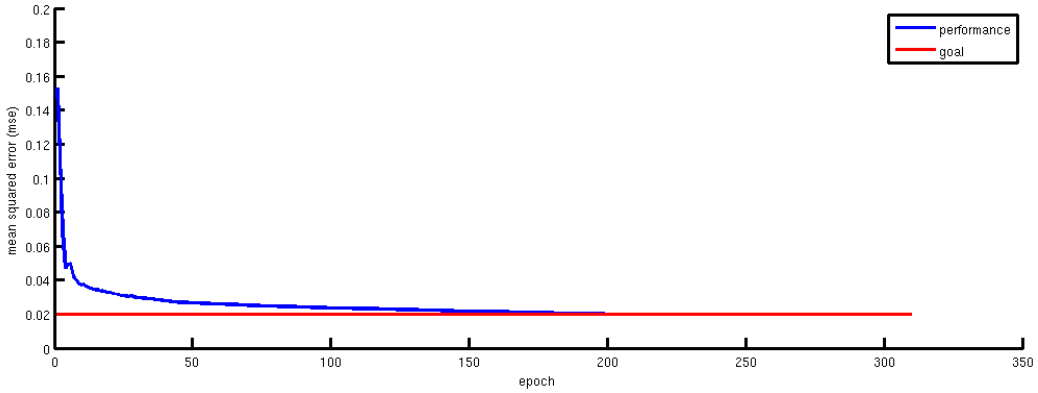


Figure 3.23: Training performance of ann 10_10_1_*rp*

| **10_10_1_rp** | |
|---|---|
| training time (seconds) | 0.24 |
| number of epochs | 199 |
| final mean squared error | $19.96 \times 10^{-3}$ |

Table 3.5: Training results for ann 10_10_1_*rp*

As in the previous case, network 6_10_1_*rp* training performance also reaches the MSE goal of $20.00 \times 10^{-3}$, only with a bit slower convergence, as shown in Figures 3.24 graph and also in Table 3.6.

With only two neurons in the hidden layer, network 10_2_1_*rp* does not reach the MSE goal. As Figure 3.25 shows, the initial convergence is very fast, but then the training performance

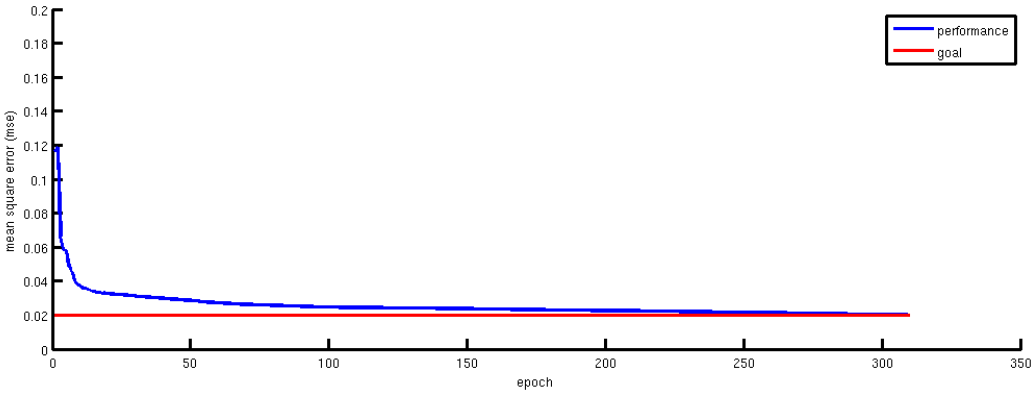


Figure 3.24: Training performance of ann 6_10_1_*rp*

| **6_10_1_rp** | |
|---|---|
| training time (seconds) | 0.30 |
| number of epochs | 309 |
| final mean squared error | $19.98 \times 10^{-3}$ |

Table 3.6: Training results for ann 6_10_1_*rp*

slowly improves over epochs, with an asymptotic behavior nearby epoch 5000. Nevertheless the training is very fast compared the the backpropagation, as Table 3.7 shows.

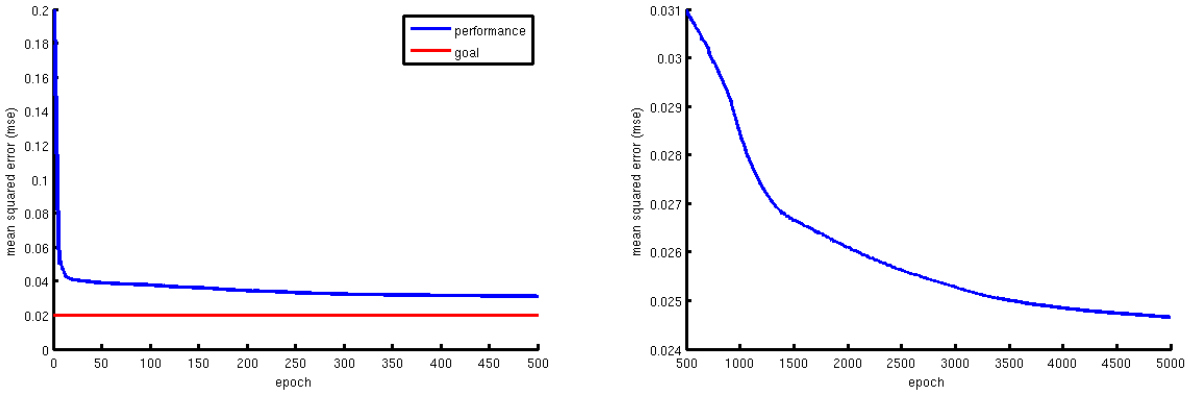


Figure 3.25: Training performance of ann 10_2_1_*rp*

| **10_2_1_rp** | |
|---|---|
| training time (seconds) | 2.51 |
| number of epochs | 5000 |
| final mean squared error | $24.65 \times 10^{-3}$ |

Table 3.7: Training results for ann 10_2_1_*rp*

As the previous networks that have only two neurons in the hidden layer, network 6_2_1_*rp*

in unable to reach the MSE goal. But as in the backpropagation case, the performance graph (Figure 3.26) of this network differs from the others. The left graph shows a very fast convergence in the initial epochs, and the right graph shows that this convergence continues until epoch 1500. After that, the performance reaches an asymptotic state, with little improvement in subsequent epochs. Table 3.8 shows the training results.

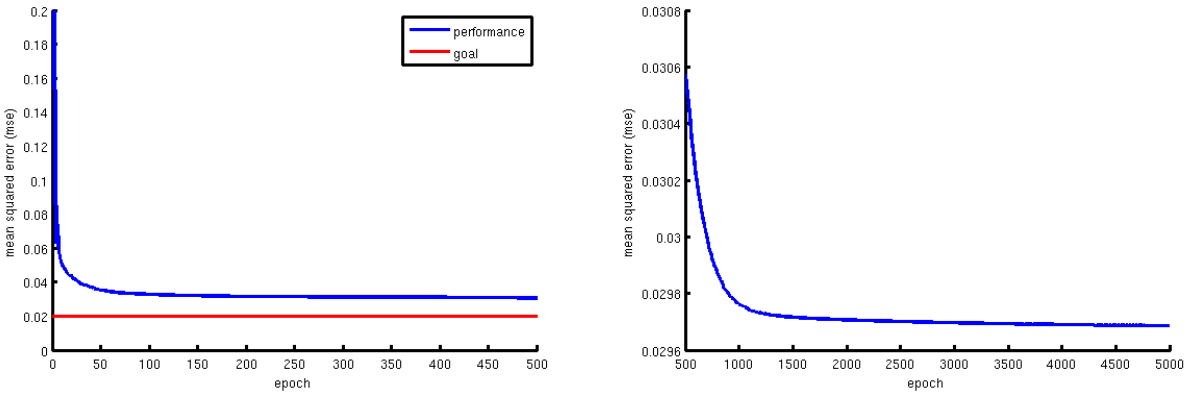


Figure 3.26: Training performance of ann 6_2_1_*rp*

| **6_2_1_rp** | |
|---|---|
| training time (seconds) | 1.71 |
| number of epochs | 5000 |
| final mean squared error | $24.65 \times 10^{-3}$ |

Table 3.8: Training results for ann 6_2_1_*rp*

### 3.4.3 Training Analysis

Although a complete analysis can only be made after the tests and results, the training information allow some conclusions to be drawn. First of all, the resilient backpropagation algorithm proved to be several times faster than the classical backpropagation, without the need of tedious parameters experimentation for an acceptable result.

During training, networks that had 10 neurons in the hidden layer had managed to reach the MSE goal, with exception of network 6_10_1_*bp*, which has a descendant behavior and might had reached the goal, if more epochs were allowed.

This indicates two possible training outcomes. One possibility is that, due to their size, these networks learned very fast how to replicate human steering. But this also may indicate that the networks overtrained, "memorizing" the training data set and, as consequence, becoming incapable of dealing with new situations. In other words, they lack generalization capabilities.

As the smaller networks also converged, but without reaching the MSE goal, there is a high chance that the bigger networks were oversized and, in this case, the very fast convergence is the result of an overtrained net. However, the only reliable form of confirming the suspected characteristics of the trained networks, is to perform real tests with the robot. The tests and results are described in the next chapter.

# Chapter 4

# System Tests and Results

Tests are the final part of an iterative study of this learning system. Although training measurements could provide some insight in the performance of a given network, it is only in real tests that the trained network behaviors can be evaluated.

## 4.1 System Tests

Tests were conducted in the same road of the training, but under several different illumination conditions. The Atlas 2009 robot still did not have a module in the executive layer, that could coordinate the speed control with the steer direction, so the speed was "constant" throughout the test and the robot had to be manually started or stopped.

The default configuration for tests is a speed of 40% of the traction motor capacity, and the working frequency of the image acquisition is limited to $7Hz$. The artificial neural networks that had the best results in the default tests were selected for more tests, but with different parameters, as a different speed, frequency and initial positions, to evaluate the influence of these parameters in their performance.

In the test mode, a YAML file containing information about a neural network is loaded by the Motion Planner program. This module receives the road limits that are extracted from the road image, then compute a steer direction and send it as a message to the Base Module, responsible for the low-level interface with the Atlas robot. Figure 4.1 better illustrates this process sequence.

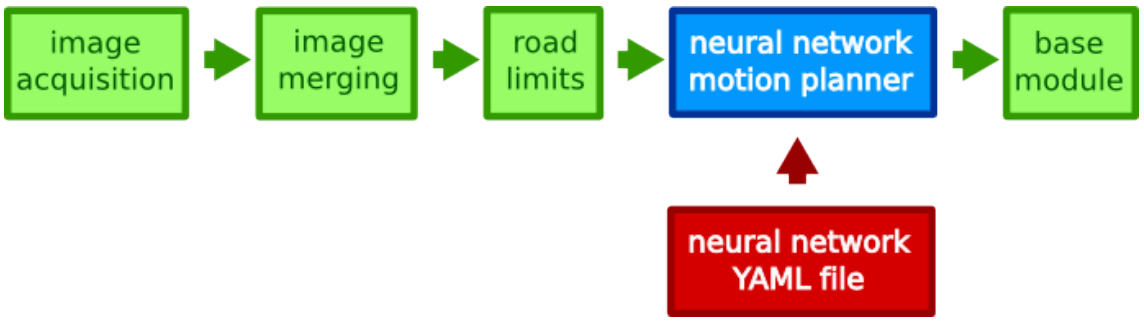


Figure 4.1: Test mode flowchart

As the robot had no odometry system installed, tests regarding the route deviation or

shortest path, could not be performed, but the time spent to complete a lap could serve as an indicator of the distance covered by the robot. Unfortunately, initial tests showed that this indicator could not be used because the time differences were so small, that they were not significant to assess the traveled distance.

Therefore, the performance of the developed learning framework will be evaluated in two aspects: the capacity to complete a lap (qualitative analysis) and the commanded steer angle measurements over a lap (quantitative analysis).

The capacity of the robot to complete a lap was the first tested situation. The robot was positioned at the right lane of the road, manually started and then filmed for a qualitative analysis of the performance. Figure 4.2 shows the initial and final position of this test.

Figure 4.2: Tests start (left) and stop (right) positions

During the tests performed for the qualitative analysis, the steering direction of the robot was recorded for the quantitative analysis. The steer direction analysis allows a deeper understanding of the artificial neural network behavior, that is not evident in the qualitative analysis.

For a correct evaluation, the logged steering needs to be timestamped. To accomplish this, a mechanism that could signal the begin and the end of a test was created. This used two sensors, shown in Figure 4.3, installed in the front bumper of the robot, that would detect the presence of a white line, used as start/stop mark.

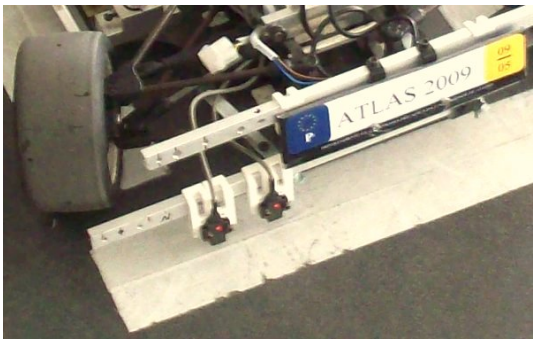


Figure 4.3: Detail of the cross sensors at the start of a test

Whenever the sensors pass over this mark, a flag is activated in the logger program, and a time counter is set to 0. After this, all the subsequent recorded angles are timestamped

relative to this 0. When the robot passes again over the start/stop mark, the time counter is again reseted, automatically signaling the last measurement of a test lap.

The acquired information is stored in a text file containing two columns, the first has the angles, in degrees, that were commanded by the artificial neural network program. The second column has the timestamp, in seconds, associated with each steer entry. The final part of a file generated with this process is shown in Figure 4.4.

```
-5.31685 19.4719
-5.3105 19.5721
-5.23219 19.6942
-4.97675 19.9
-4.82427 20.0038
-4.42064 20.2079
-3.92051 20.3119   test end
-3.16807 0.0676231
-2.33986 0.0158681
```

Figure 4.4: A sample of a steer log file

## 4.2 Lap Completion Domain Results

In this domain, the capacity of the robot to complete a lap over the laboratory road is evaluated. A lap performed by the robot is considered successful if the robot reaches the same position where it began without going out of the road. All the laps were conducted in the counterclockwise direction.

To compare neural networks that successfully completed laps, the behavior of the robot in critical portions of the road will be labeled and scored for better analysis, in a qualitative way. Four representative regions are chosen for performance comparison, shown here in Figure 4.5:

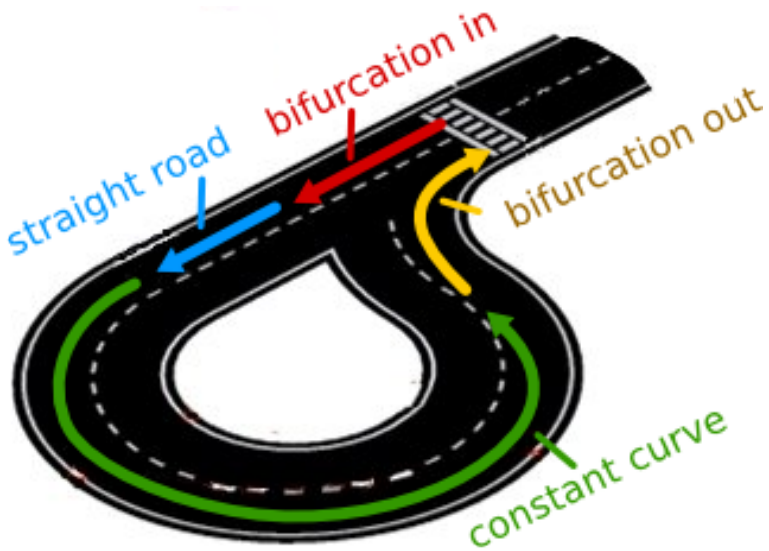


Figure 4.5: Road division labels for qualitative analysis

- **bifurcation in (BI)**: represent the transition that occurs when the robot has two

options to follow, the robot performance will be classified according to the presence ($\mathbf{P} = -1$) or absence ($\mathbf{A} = +1$) of a visible oscillation during this transition;

- **straight road (SR)**: allows an analysis of stability during straight road, performance will be classified regarding the presence ($\mathbf{P} = -1$) or absence ($\mathbf{A} = +1$) of visible oscillations;

- **constant curve (CC)**: this will classify the robot position during a constant curve, three positions are possible: a left lane curve ($\mathbf{L} = +2$) when the right front wheel is on the left lane; a middle lane curve ($\mathbf{M} = +1$) when front wheels are on different lanes; and a right lane curve ($\mathbf{R} = 0$) when left front wheel is on right lane, with the latest presenting a higher risk of the robot going out of the road, and the first being the most secure and shortest path;

- **bifurcation exit (BO)**: it is the region where the two roads merge back to one, there is a sudden absence of the left line during a curve in this case. The performance will be classified as open turn ($\mathbf{OT} = 0$) and close turn ($\mathbf{CT} = +1$), with the latest being the desired behavior, as the robot can better return to a straight position. Also, if the robot step over or pass the right road limit line, then the behavior will be classified as a very close turn ($\mathbf{VC} = -2$), which is highly undesired.

According to this metric, the **ideal lap** would not have oscillations in the steering direction during the *bifurcation in* and the *straight road*, performing a *constant curve* in the left lane and finishing with a close turn in the *bifurcation out*. Such a case would have a score of 5 points. In the other hand if a network fails to complete a lap, no score will given in the failure portion of the road and it will be given a total score of $-5$ points, discarding the previous sums. Table 4.1 shows the results of this metric applied to each of the networks.

| | backprop | | | | | rprop | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *ANN Structure* | **BI** | **SR** | **CC** | **BO** | ***Score*** | **BI** | **SR** | **CC** | **BO** | ***Score*** |
| **10_10_1** | A | A | R | – | **-5** | P | A | R | CT | **+1** |
| **6_10_1** | P | A | R | OT | **0** | A | P | R | VC | **-2** |
| **10_2_1** | A | P | M | CT | **+2** | A | P | R | CT | **+1** |
| **6_2_1** | A | A | M | CT | **+4** | A | A | M | CT | **+4** |

Table 4.1: Artificial neural networks goal based performance for speed level of 40%

### 4.2.1 Qualitative Analysis

These test results clearly indicate that the best networks created in this research were the smallest ones, specially the networks containing only 2 neurons in the hidden layer. Bigger networks may have faster convergence, but this does not translates directly in good performance. In the current tests, what happened was quite the opposite.

This reinforces the suspicion of the previous chapter, that the bigger networks had overtrained, "memorizing" the training set and becoming unable to perform generalization and adapt to new conditions. This is an indication that the MSE alone may not be the most adequate criterion for train termination, in the context of this research.

Another possible explanation for the better performance of the small networks is that the proposed problem of computing a steer angle may not be a very complex one. So a large network would incorporate a complexity that was unnecessary, degrading the performance.

## 4.3 Steer Domain Results

A concept that can be related with a good driving is the steer smoothness. This is due to the fact that when the road has a smooth or continuous shape, be it a straight line or a semicircle, a good driver should maintain an almost constant steer angle (ideally 0, in straight roads, but mechanical and electronic imperfections can offset this value).

This can also be understood as a reference for a feedback control system, where a constant road shape would be a constant reference value. Often in control systems, it is desired that the system output (steer angle in this case) follows a reference without oscillation and small errors.

This is a performance indicator and can be measured indirectly, in this system, by calculating the derivative of the steer angle. The evaluation will be made in terms of the frequency and amplitude of the oscillation. In the derivative graph, peaks indicate abrupt changes in the steering angle, while values near zero means a stable, or smooth steering.

As mentioned before, due to mechanical and electronic imperfections, there is no optimal steer angle for each section of the road. So, in order to facilitate the comprehension of the acquired data, the graphs will be divided into regions, and each region will have a "suggested" steer angle.

Three regions can be defined, and are represented in the graphs by two vertical lines. The middle region, where the steer angle goes nearby 11 degrees is the **curve region**. It is the curve region that defines the region divisions, with the **straight road region** before it, and the **road inflection region** after it. These division are illustrated in Figure 4.6.

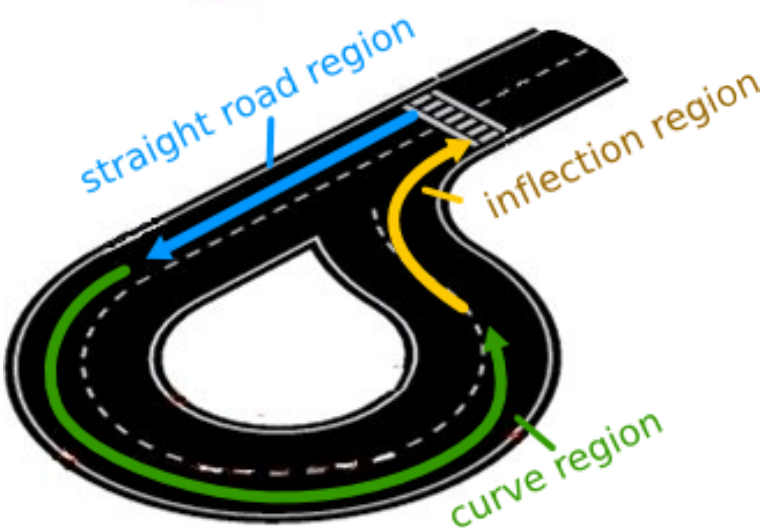


Figure 4.6: Road division labels for quantitative analysis

The only difference compared to the divisions of the qualitative analysis, is that there is only one "straight" part here. The reason is that while it is easy to notice in a video in which

part of the road the robot is located, the same task can not be done while looking to the steering graph only, with the steering angles being the only hint for separations.

Also three horizontal lines are present in the graphs, representing a "suggested" steer angle that could keep the robot inside the road in the different regions. It is important to emphasize that this marks are only an estimation, and are used only as visual aids for comparison and analysis of the tests. Figure 4.7 illustrates the suggested steer angles and the three regions previously defined.

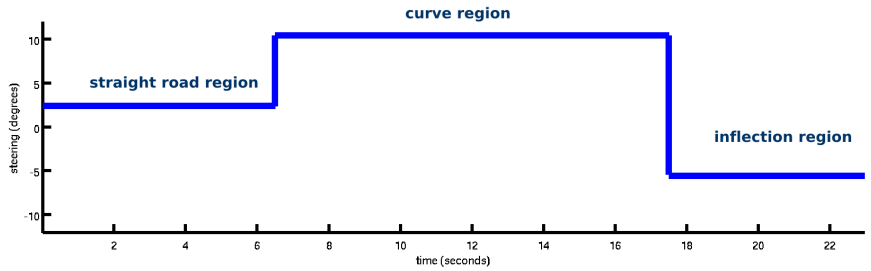


Figure 4.7: Approximated reference angles for the three regions

Using this form of graphical analysis allows a deeper insight in the behavior of the trained artificial neural networks. Regarding the derivative graphs, while in the road inflection region, the robot has to go from a left curve to a right curve, so a high derivative is desired in this case. During the other parts of the lap, however, it is desired a near zero derivative, with exception of the transition between the straight road and the curve region.

For this evaluation, each of the trained artificial neural networks is tested using the default test configuration: speed of 40%, program frequency of $7Hz$ and right lane start position.

After the initial tests and analysis of the trained networks, new tests are conducted but with different parameters, as speed, program frequency and robot start positions. The objective is to study how this parameters affect the overall performance of the robot over a lap.

For each network and parameter set, three tests are performed and the results are shown in a graphical view of the steering direction angle over time, and also in figures showing the derivative of the steering. The same steer angle scale will be used in all the tests for a better comparison between performances.

### 4.3.1 Backpropagation Networks With Speed Level of 40%

The results for network 10_10_1_*bp* are presented in Figure 4.8. During the first region, a low amplitude and medium frequency oscillation can be seen in the first graph. The derivative of the steering shows that this oscillation is quite smooth. During the curve, small corrections are made. In the final part, this network fails to complete the lap, making a very strong turn and going out of the road.

Figure 4.9 shows a hight amplitude oscillatory behavior of network 6_10_1 during the straight line and during the road inflection region. The oscillations are also present during the curve region but have a low amplitude and frequency.

The first graph of network 10_2_1_*bp*, shown in Figure 4.10, clearly shows that oscillations are present throughout the test, as in the previous case. But the steering derivative graph in this case, is almost the opposite from before. It shows that the oscillations in the straight

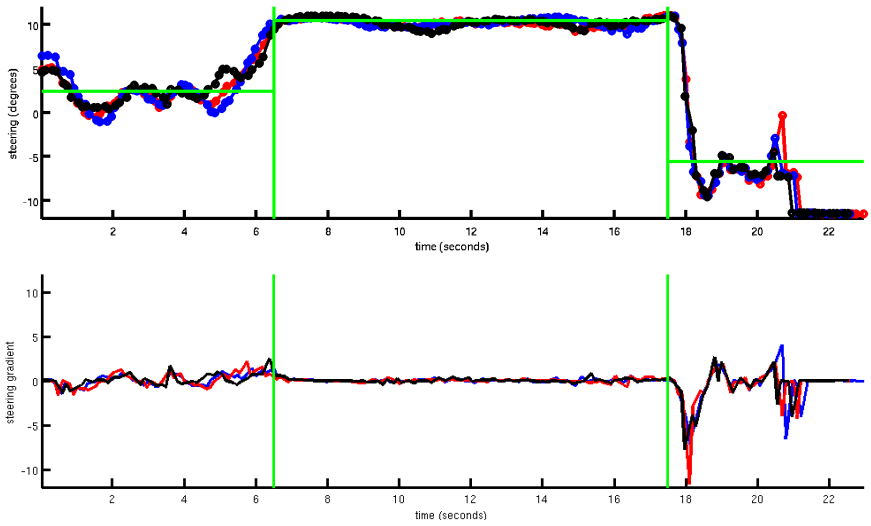


Figure 4.8: Steering direction and derivative, ann 10_10_1_*bp*, speed 40%

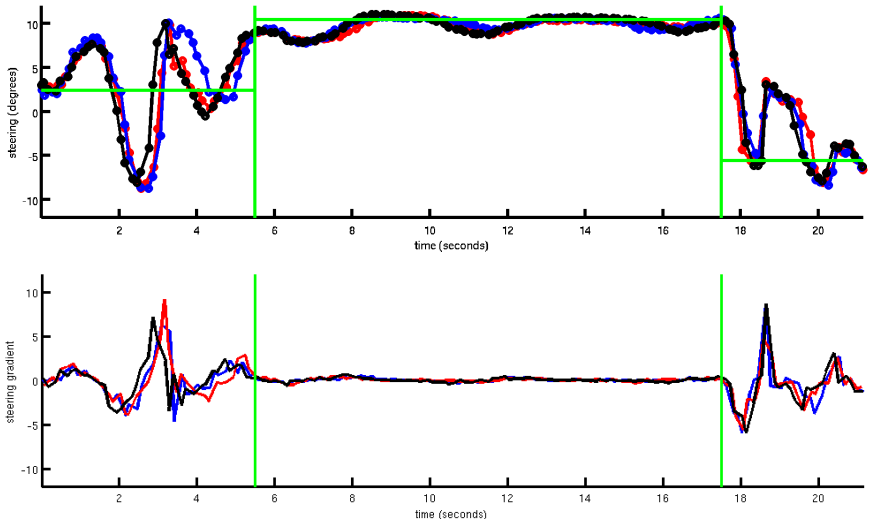


Figure 4.9: Steering direction and derivative, ann 6_10_1_*bp*, speed 40%

road region have a low amplitude, but higher frequency then the previous results. During the curve regions, this high frequency behavior continues, but with a very small amplitude. Finally, at the road inflection regions, no oscillations are noticeable.

As expected due to previous analysis, the network 6_2_1_*bp* (Figure 4.11) have the best performance compared to the other backpropagation networks. The steer graph shows a very smooth driving, with minor oscillations in the first region, and no oscillations at all during the curve and inflection road regions.

The peak present in the final part corresponds to the change in steer that is necessary in the road inflection region. After that, the robot resumes the lap in a smooth way. There is also a resemblance with the performance graphs of the previous network, but without the persistent oscillations in the steering.

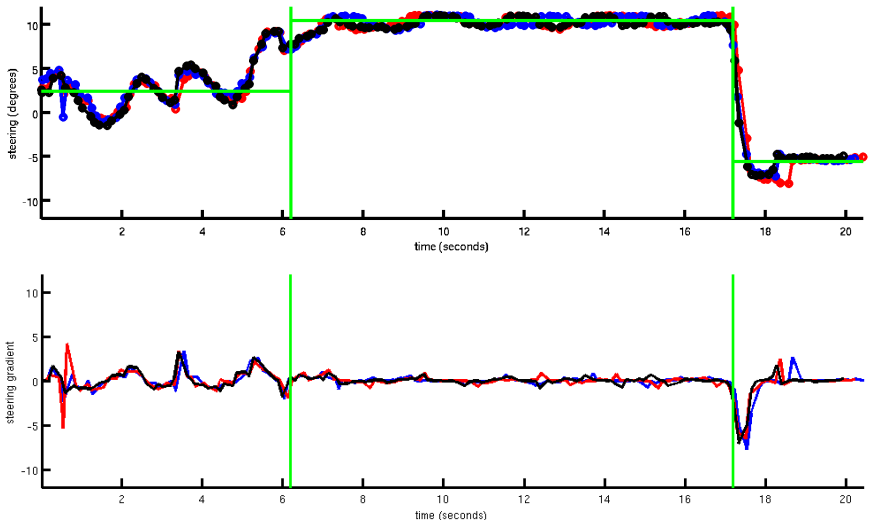


Figure 4.10: Steering direction and derivative, ann 10_2_1_*bp*, speed 40%

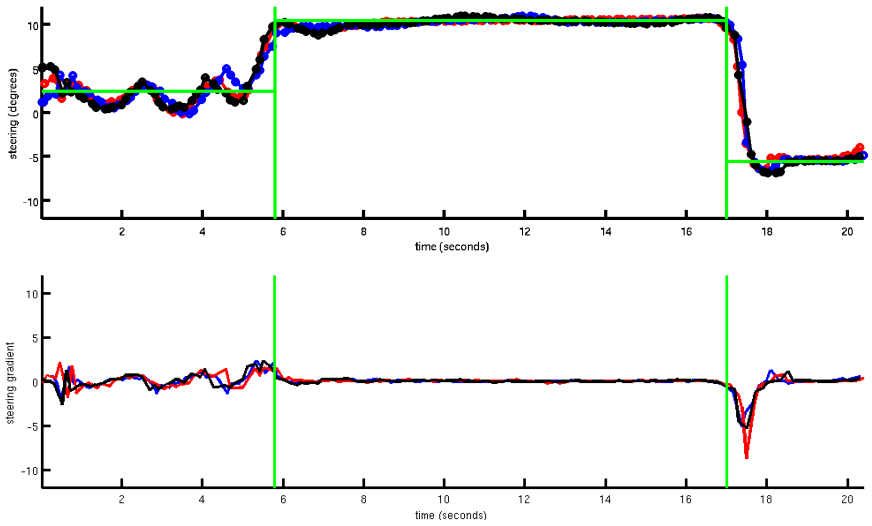


Figure 4.11: Steering direction and derivative, ann 6_2_1_*bp*, speed 40%

### 4.3.2 Resilient Backpropagation Networks With Speed Level of 40%

Network 10_10_1_*rp*, shown in Figure 4.12, has a low frequency but high amplitude oscillatory behavior during the straight line region. The curve regions is performed almost with a constant steer angle, which shows a good stability. In the inflection region, the oscillations behavior returns.

Figure 4.13 shows that the performance of network 6_10_1_*rp* is is worse than the previous. Here oscillations are present during all test regions, but with a higher frequency and higher amplitude in the straight road and inflection regions. During the curve region, the oscillations have a very small amplitude.

It is interesting to notice in the steering graph of Figure 4.13, that a slightly different start

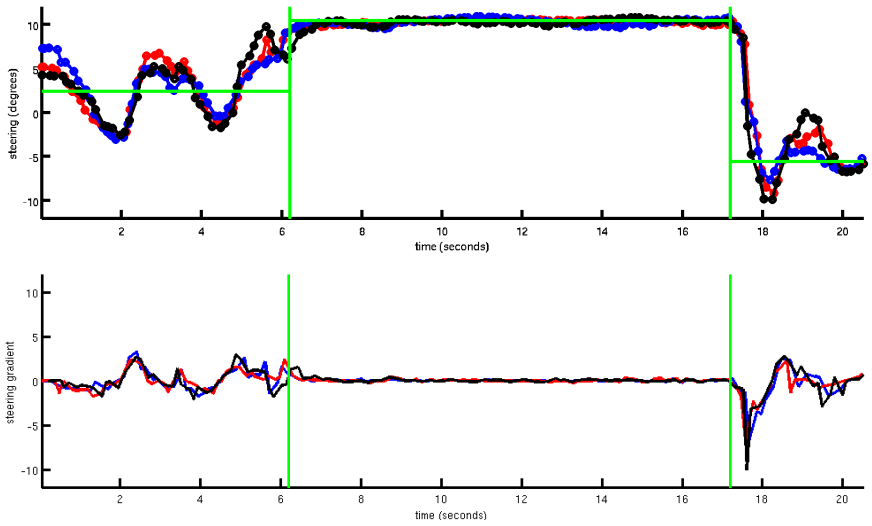


Figure 4.12: Steering direction and derivative, ann 10_10_1_*rp*, speed 40%

condition (different initial steer angle) may result in quite distinct behaviors, indicated by the test line with a smaller oscillation amplitude in the first region. The impact that different start conditions may have in the overall performance will be further analyzed.

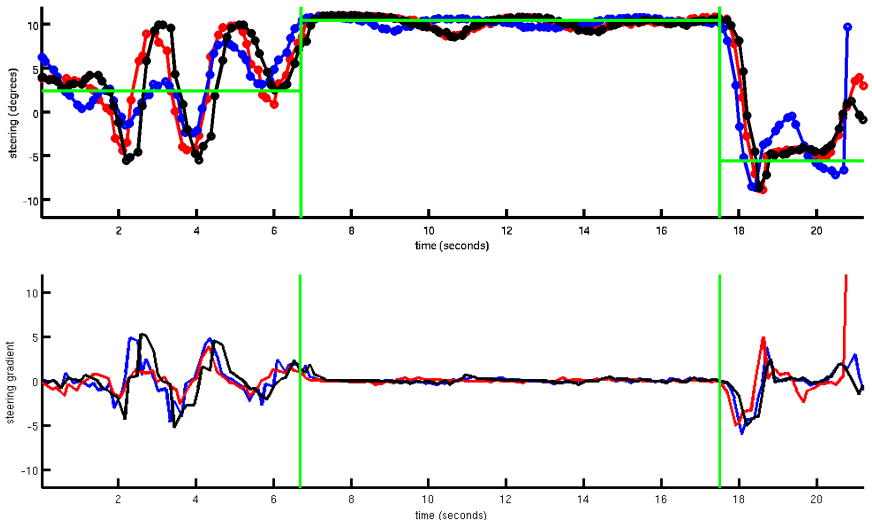


Figure 4.13: Steering direction and derivative, ann 6_10_1_*rp*, speed 40%

Network 10_2_1_*rp* has a very degraded performance, even with a stable behavior in the beginning of the test, shown in Figure 4.14. There is a very high amplitude and frequency in the oscillations along the test. It is interesting to notice, however, that in the road inflection region this network does not respond as fast as other analyzed networks, indicating a smooth change of the steering. Another aspect is the that this network, unlike the others, constantly reaches the maximum steer angle during the curve region.

As in the backpropagation case, the best performance occur with the smallest network.

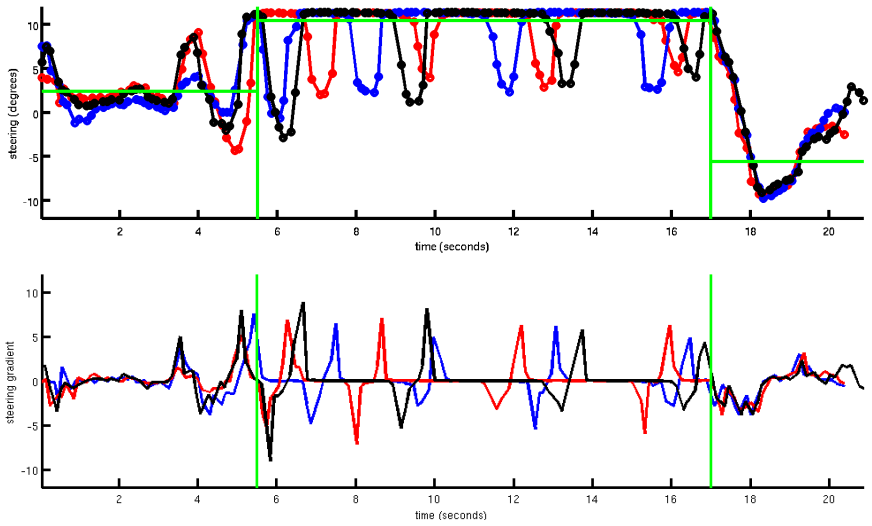


Figure 4.14: Steering direction and derivative, ann 10_2_1_*rp*, speed 40%

The performance of artificial neural network 6_2_1_*rp* is illustrated in Figure 4.15, and shows a smooth overall performance, with little oscillation. The main difference when compared with its backpropagation counterpart is a small oscillation during the curve region and a not so precise turning in the road inflection region.

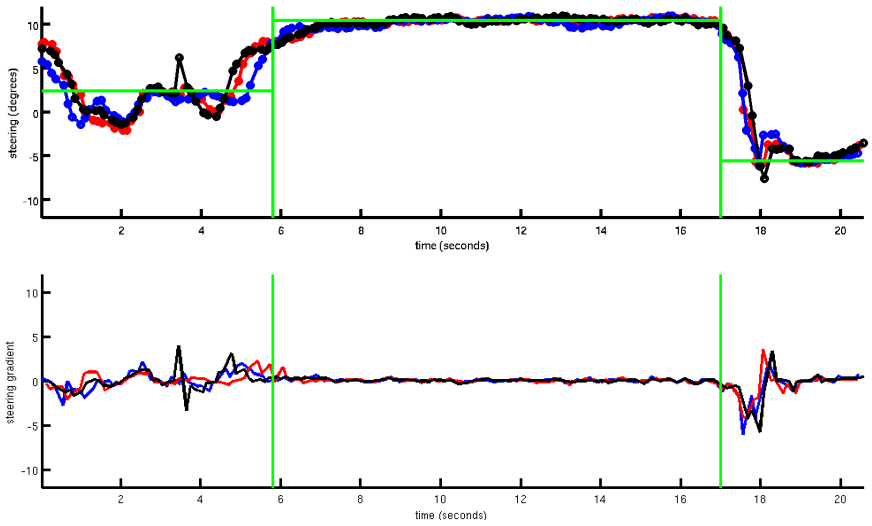


Figure 4.15: Steering direction and derivative, ann 6_2_1_*rp*, speed 40%

### 4.3.3 Quantitative Analysis

As was expected, in the default tests, the smaller networks had the best performance in the steer angle quantitative analysis. Also, although the backpropagation training requires several tests until adequate training parameters are found, in the tests conducted here, it were those networks that had the best steer performance among all.

All the tests performed with the default parameters, showed oscillations in the steer angle. The reason for this is not clear. It may be related to the electromechanic response of the steer motor, or to the low frequency used in the program. Only further tests, that explore more aspects of the networks, can clarify this behavior.

### 4.3.4 Speed Influence Results

As mentioned before, networks 6_2_1_*bp* and 6_2_1_*rp* are selected as they had the best performance in the analysis where standard tests parameters were used. Now their performance will be studied over the influence of a speed of 60% of the traction motor capacity.

In the results of the backpropagation trained network, shown in Figure 4.16, there are oscillations with a low frequency and medium amplitude during the first region. In the curve, the oscillations almost disappear, and are not present at all in the final part.

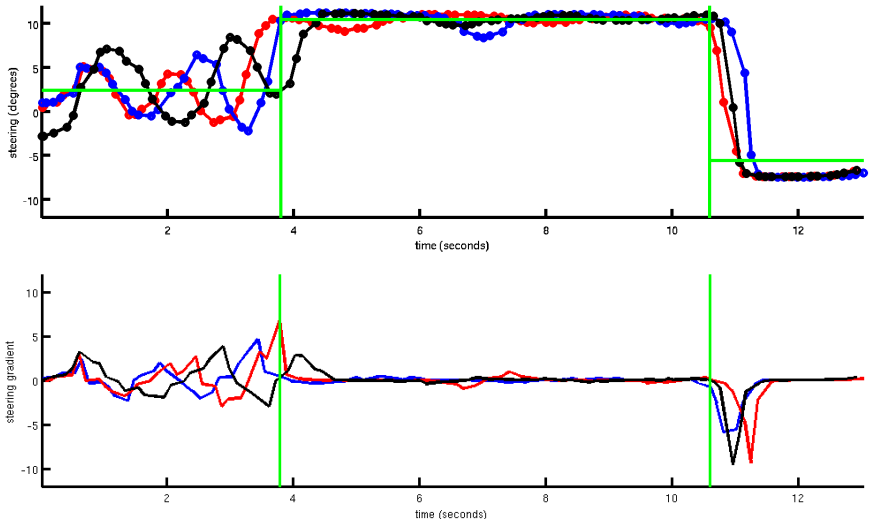


Figure 4.16: Steering direction and derivative, ann 6_2_1_*bp*, speed 60%

The performance graph of the resilient propagation case is shown in Figure 4.17. In this case, there is a high amplitude, but low frequency oscillation present in the first region. During this part of the test, the oscillation even seem to be getting higher, which is interrupted by the begin of the curve region.

Oscillations are also present in the curve region, but with a smaller amplitude. This network was clearly more affected by the speed increase than its backpropagation counterpart.

#### Speed Influence Analysis

An increase on the speed resulted in higher steering oscillation of the tested networks. As in previous analysis, the exact causes of this behavior are not clear, but may be related to the slow frequency used by the artificial neural network program.

In this case, if the robot is going too fast, a small offset in the correct steering would abruptly change the robot position in the road, requiring a strong steering compensation which would lead again to the same situation as before, and so on.

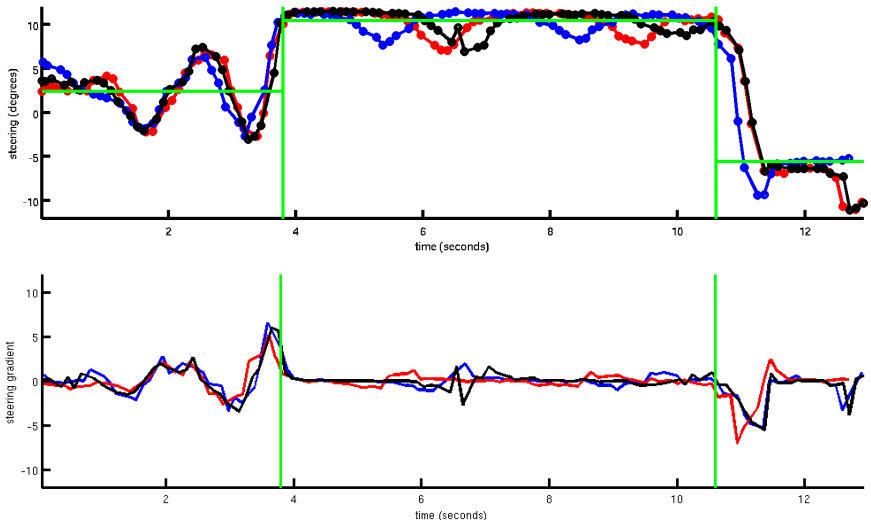


Figure 4.17: Steering direction and derivative, ann 6_2_1_*rp*, speed 60%

Another possible explanation is due to the fact that during training only small speeds were used, and it is completely different to perform a complete turn at slow or at high speed. This may have been resolved if the robot speed was also used as network inputs, in an attempt of teaching the artificial neural networks that at high speeds, the direction should not change so abruptly.

### 4.3.5 Initial Position Influence Results

Experimental tests showed that slightly different initial positions may be the difference between a successful or unsuccessful lap. This characteristic served as inspiration for the current test, where the robot was positioned at three different parts of the road at the begin of each test.

The initial positions were at the right lane, at the middle of the road and at the left lane. Tests already showed that the created artificial neural networks tend to maintain a (mostly) constant position during straight regions of the road, that is a mid term between center line and right lane. So it was expected that the initial position would work as an offset of this usual position and therefore, the first maneuver of the robot would be an attempt to reach such position in the road.

For this test, both experimented networks have similar results, shown in Figures 4.18 and 4.19. When the robot is positioned in the right lane, the first maneuver is a left turn, to approach the robot to the the right-center of the road. In the middle road and left lane positions, the initial maneuver is a right turn, for the same reason as before.

Also in both cases the steering oscillates during the straight road region and when the curve region starts, there is no distinguishable difference in the steering direction of the three different initial positions.

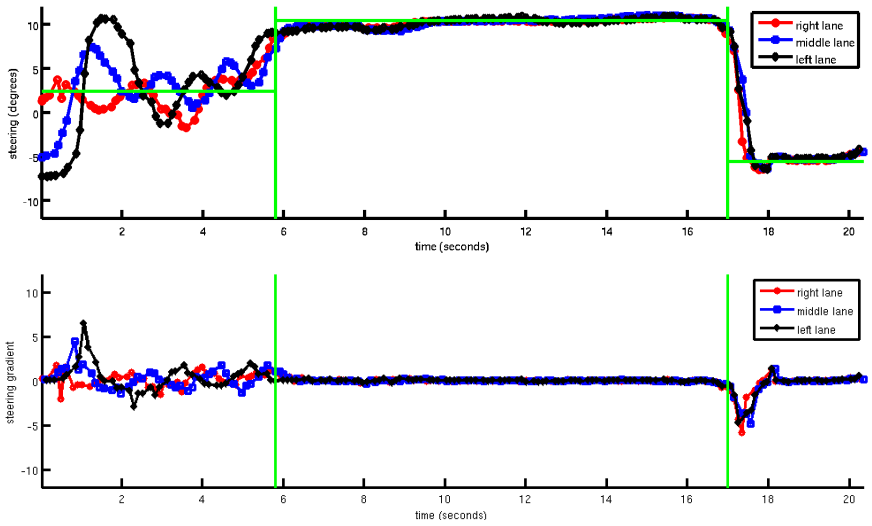


Figure 4.18: Steering direction and derivative, ann 6_2_1_*bp*, speed 40%, different initial positions

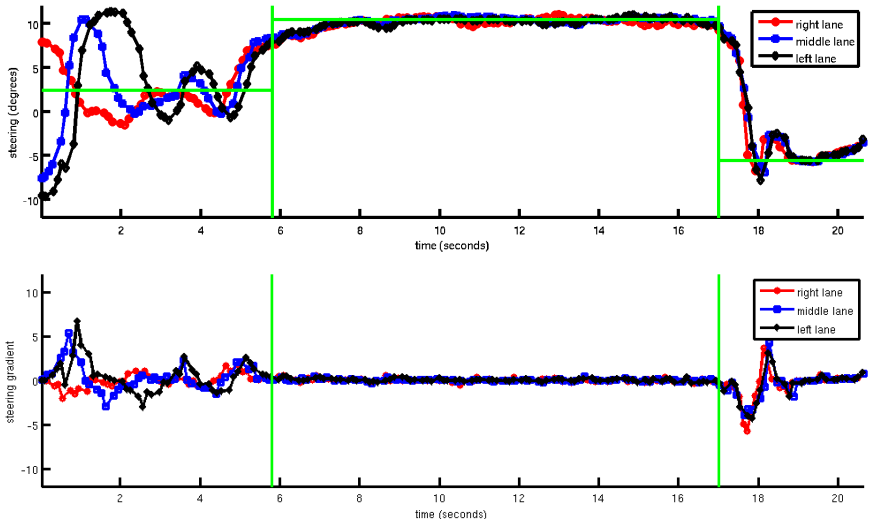


Figure 4.19: Steering direction and derivative, ann 6_2_1_*rp*, speed 40%, different initial positions

### Initial Position Influence Analysis

Tests showed that, for the small speeds, the trained artificial neural networks can successfully compensate diverse initial positions. However at higher speeds, the initial maneuver to bring the robot to the usual lane position could take the robot out of the road, as it would be moving too fast for an abrupt correction maneuver.

This is another reason why the speed of the robot should be used as an input for the neural network, to avoid such situations of large steering angles at high speeds.

### 4.3.6 Image Frame Rate Influence Results

The influence of the the image frame rate in the steering behavior of the robot is of great interest. This may serve as an indication of how well the trained networks can correct from undesired positions, as there is a longer time between commands to the steering motor. For example, if the robot is entering a curve and the last command was to follow in a straight line, when the algorithm is able to send a new command, it has to be able to quickly correct the steering to stay in the curve and inside the road.

The default program frequency was about $7Hz$. The desired frequency was actually $10Hz$, but the way the image processing modules were organized did not allowed a higher processing rate when tests were performed. To test the influence of this parameter, the frequency of the motion planner algorithm was forced to only $2Hz$.

It is important to notice that this frequency is not related to the time spent by artificial neural networks to perform a computation, which is very fast. Instead of that, the adjusted frequency is the maximum rate at which the motion planner program can send messages to the base module.

Network 6_2_1_*bp* has a very good result (Figure 4.20), with a behavior very similar to the one with the default frequency. There is a minor oscillation in the first region, followed by a very smooth curve and a smooth change of direction during the road inflection region.

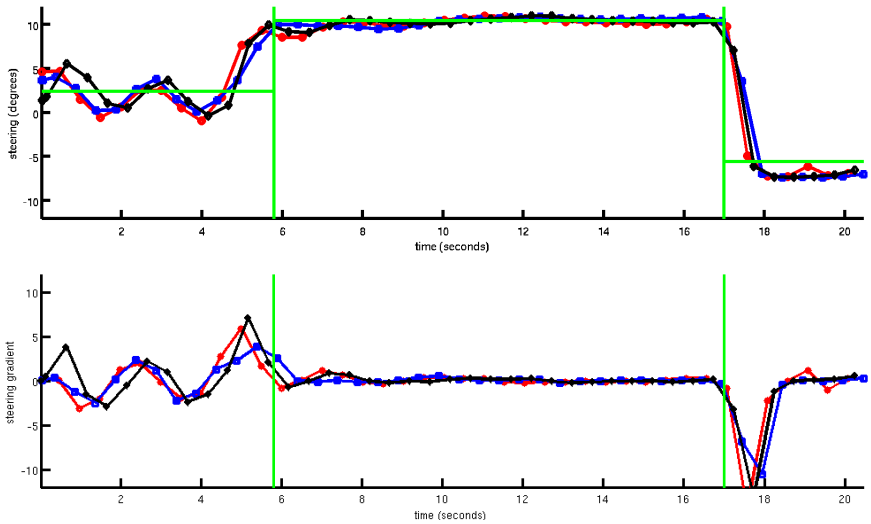


Figure 4.20: Steering direction and derivative, ann 6_2_1_*bp*, speed 40%, 2 Hertz

The network 6_2_1_*rp* suffered more with this limited frequency. Results (Figure 4.21) show a much higher oscillation during the first region and also during the curve. This network seems to have a more reactive behavior, which requires it to perform strong corrections from older decision, which may lead to the presented behavior when using a lower frequency, as it is in this case.

#### Program Frequency Influence Analysis

The program frequency test results are very promising. They show a high robustness of the used artificial neural networks, that were capable of completing a lap, even with a

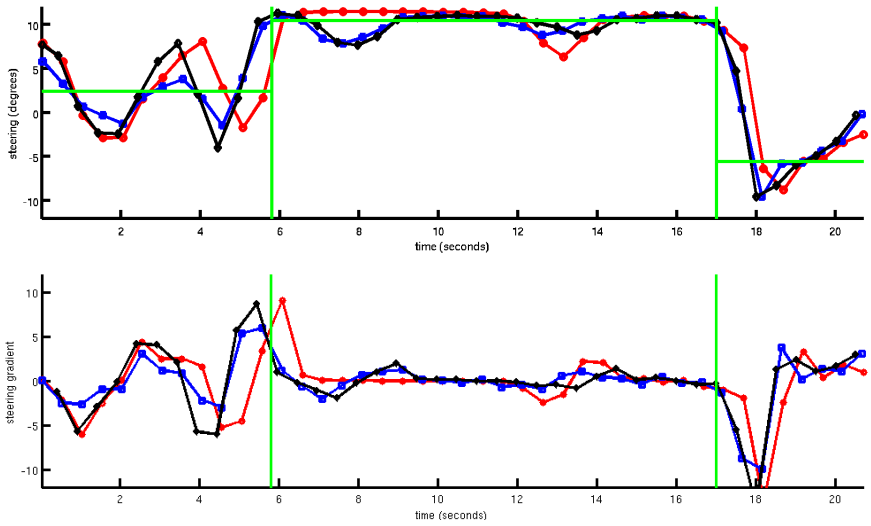


Figure 4.21: Steering direction and derivative, ann 6_2_1_*rp*, speed 40%, 2 Hertz

frequency three times slower than the default one. This represent a good recover capacity and also indicates that the frequency is important, but not crucial for a good performance of the robot.

The possibility to navigate with slower frequencies also indicates that the feature extractor programs may be further enhanced to perform better but slower computations without compromising the overall robot performance. This are very interesting results, as humans tend to have a similar behavior: we have very fast and complex reasoning systems, but we interact with the world very slowly.

# Chapter 5

# Conclusions and Future Work

## 5.1 Conclusions

This research proposed a system where humans could teach an autonomous robot by examples. With this conceptually attractive approach, real world conditions were not modeled into algorithms. Instead of that, a program created its own representations of the world it sensed, and learned how to act on it.

A system that uses this concept was implemented. Based on human examples of steering a vehicle inside a small scale road, artificial neural networks managed to replicate behaviors and generalize to new situations, managing to autonomously drive a robot inside a road.

To accomplish these results, an automated system that combined hardware and software was necessary, integrating mechanical structures, software architectures, theories of classic and modern control and theories of computer science, as artificial neural networks.

A new structure was created for the cameras fixation, to establish a platform where tests could be replicated and repeated. Also a new modular software architecture was implemented to support easy integration of new modules that were necessary for this research.

An important part in this work was the study and development of the artificial neural networks. This comprised training experiences using different computer languages, the development of a differentiated human machine interface for data acquisition and finally the creation of a steer computation program that uses artificial neural networks to compute steer angles capable of maintaining the robot inside a road.

To validate the created networks, real tests were performed instead of simulations. The robot successfully completed several laps in the test circuit, showing good capacities for recovery and for generalization with relatively small training data sets.

The long time spent during the initial stages, with the mechanical and software improvements, was compensated by an extreme flexibility in the final research stage, with a reliable camera support and a modular architecture that allowed the integration of new modules, as the neural network motion planner, effortlessly.

Implementations conducted throughout this work also brought contributions in several levels. Regarding the software architecture, it became the new standard for the robots developed at the Laboratory of Automation and Robotics at the University of Aveiro.

The modular architecture makes easy to new members join the group of developers and allows directly contribution in the main program as the form of modules, expanding the robots functionalities. The new architecture also allows easy integration of multiple computers,

working together in networks and creating powerful processing clusters.

The created learning framework proved to bring several advantages over the classical deterministic techniques. With this approach no model of the road nor goto points computations are necessary for the robot navigation. Also the training process is very quick and the data acquisition is performed in a joyful way.

This allows new networks to be trained to adapt to new roads or situations in very short time, resulting in a big step compared to the other techniques, where all the possible situations had to be anticipated and coded.

The successful implementation of a learning framework also opens several new possibilities for improvements in the programs of the Atlas Project robots. Taking advantage of the new modular architecture, new modules that use machine learning techniques may be integrated with older modules, creating very complex and robust robot behaviors.

In several different tests, the smallest networks had the best performance with both training algorithms that were experimented. Although the backpropagation technique had very good results, the resilient backpropagation training was much faster and it also did not required empirical adjustments of the training parameters.

Networks trained with the backpropagation technique had better overall results, but not enough tests and trainings were performed to determine which method is the most efficient for the presented problem.

During training two criterion were used for stopping: the MSE goal and maximum number of epochs. However, networks that reached the established MSE goal, had a worse performance than the ones that did not reached it. This suggests that other criterion may be experimented together with the MSE, to improve the network trainings.

Finally, the use of metadata as the method for storing training data, was a very good choice. This allowed a reliable data organization, with single image files that contained aggregated sensors readings in the form of metadata fields. This resulted in an efficient and simple data management, that follows well known standards. The image data bank also allows several different forms of road features extraction to be compared, using the same data sets.

## 5.2 Future Work and Suggestions

This work created several open investigation lines for further research. One interesting development would be to use artificial neural network to measure humans performance while driving, giving suggestions and possible scores that could be used for granting a drivers license, for example.

The implementation of odometry, be it visual or using encoders, will provide additional forms of evaluating the performance of the robot.

As the artificial neural networks make very fast computations, the overall program speed is limited by the cameras frequency, and by the feature extraction in the images. This can be further improved with the use of smaller images, resulting in a faster image processing. Another option is to start a research in the use of machine learning techniques for the feature extraction of the road, limiting the overall speed only to the cameras frequency capacity.

During tests the robot was using always a constant speed, as this module was not fully integrated with a module responsible for states management and speed calculation during the competitions. This is a work that will be performed, resulting in a program that can be used in the competition. In this case, different artificial neural networks will be trained for different

desired behaviors, and the management program could switch among different networks.

Besides that, the speed can also be used as an input for the neural network in an attempt of further incorporating human-like behaviors as slowing down on curves and speeding up in straight road, and even using the network itself to compute a desirable speed.

The data acquisition also can be further improved, with a sound feedback not only from the log phases but also from the program steer computation. Another feature that could enhance the human machine interface would be a "shared command" over the robot, where the human driver and the artificial neural network program could simultaneously command the robot, sending speed messages together with a confidence level for a decision module.

If the human driver noticed an abnormal behavior on the robot, the press of a button could change the gamepad confidence to a higher level then the network program, instantaneously taking the robot command and correcting the abnormal behavior. When the robot was brought back to a normal situation, a gamepad button could return the gamepad speed confidence level so the command would return to the robot's artificial network program.

During the human intervention, images would be logged with the human steering information, so this could be further used to train the robot on how to escape and to avoid that abnormal behavior.

Also further tests are suggested, to study the behaviour of the artificial neural networks over extreme conditions to better assess their limitations, so more efficient networks can be created.

# References


[AdobeXMP, 2009] AdobeXMP (2009). Adobe xmp: Adding intelligence to media. Retrieved April, 2009, from `http://www.adobe.com/products/xmp/`.

[Alpaydin, 2004] Alpaydin, E. (2004). *Introduction to Machine Learning*. The MIT Press.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

[Carmen, 2009] Carmen (2009). Carmen robot navigation toolkit. Retrieved April, 2009, from `http://carmen.sourceforge.net/`.

[Darter and Gordon, 2005] Darter, M. and Gordon, V. (2005). Vehicle steering control using modular neural networks. In *Proceedings of IEEE International Conference on Information Reuse and Integration, IRI -2005*, pages 374–379.

[Demcenko et al., 2008] Demcenko, A., Tamosiunaite, M., Vidugiriene, A., and Saudargiene, A. (2008). Vehicle's steering signal predictions using neural networks. In *Proceedings of IEEE Intelligent Vehicles Symposium*, pages 1181–1186.

[Exiv2, 2009] Exiv2 (2009). Exiv2 - image metadata library and tools. Retrieved May, 2009, from `http://www.exiv2.org/`.

[Festival, 2009] Festival (2009). Festival nacional de robótica. Retrieved May, 2009, from `http://www.spr.ua.pt/fnr/`.

[IDEF0, 2009] IDEF0 (2009). Idef0 overview. Retrieved May, 2009, from `http://www.idef.com/idef0.html`.

[Kunzle, 2009] Kunzle, P. (2009). Gamedev.net - vehicle control with neural networks. Retrieved May, 2009, from `http://www.gamedev.net/reference/articles/article1988.asp`.

[Montemerlo et al., 2008] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597.

[Montemerlo et al., 2003] Montemerlo, M., Roy, N., and Thrun, S. (2003). Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*.

[Oliveira and Santos, 2007] Oliveira, M. and Santos, V. (2007). A vision-based solution for the navigation of a mobile robot in a road-like environment. *Robótica*, 69.

[Oliveira et al., 2009] Oliveira, M., Stein, P., Almeida, J., and Santos, V. (2009). Modular scalable architecture for the navigation of the ATLAS autonomous robots. In *9th Conference on Mobile Robots and Competitions - Portuguese Robotics Open*, Castelo Branco, Portugal.

[OpenCV, 2009] OpenCV (2009). SourceForge.net: OpenCV. Retrieved May, 2009, from `http://sourceforge.net/projects/opencv/`.

[Pomerleau, 1991] Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97.

[Pomerleau, 1995] Pomerleau, D. (1995). Neural network vision for robot driving. In Arbib, M., editor, *The Handbook of Brain Theory and Neural Networks*.

[Riedmiller and Braun, 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *Proceedings of IEEE International Conference on Neural Networks*.

[Rosenblatt, 1997] Rosenblatt, J. K. (1997). DAMN: a distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 9:339–360.

[Sarle, 2009] Sarle, W. S. (2009). Neural network faq, part 2 of 7: Learning. Retrieved May, 2009, from `ftp://ftp.sas.com/pub/neural/FAQ2.html`.

[Siciliano and Khatib, 2008] Siciliano, B. and Khatib, O. (2008). *Springer Handbook of Robotics*. Springer, 1 edition.

[Simmons, 2009] Simmons, R. (2009). Robotics institute: Inter-Process communication package. Retrieved March, 2009, from `http://www.ri.cmu.edu/research_project_detail.html?project_id=394&menu_id=261`.

[Simmons and Apfelbaum, 1998] Simmons, R. and Apfelbaum, D. (1998). A task description language for robot control. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*.

[Wikipedia, 2009] Wikipedia (2009). Condução autónoma - wikipédia, a enciclopédia livre. Retrieved May, 2009, from `http://pt.wikipedia.org/wiki/Condução_Autónoma`.

[YAML, 2009] YAML (2009). The official yaml web site. Retrieved April, 2009, from `http://www.yaml.org/`.