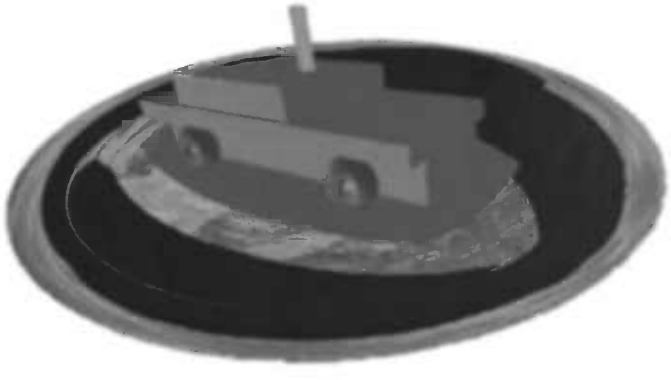
Omnidirectional Active Vision in Evolutionary Car Driving

M.Sc. Graduation Thesis in Artificial Intelligence

Jacob van der Blij[1]

Supervisors:
Prof. Dario Floreano[2], Mototaka Suzuki[2], Dr. Bart de Boer[1]

August 2005



[1]Artificial Intelligence, University of Groningen (RuG), Grote Kruisstraat 2/1, 9712 TS Groningen, The Netherlands. Web: *http://www.rug.nl/ai*, E-mail: firstname@ai.rug.nl

[2]Laboratory of Intelligent Systems, Swiss Federal Institute of Technology (EPFL), Station 11, CH-1015 Lausanne, Switzerland. Web: *http://lis.epfl.ch*, E-mail: firstname.surname@epfl.ch

Abstract

Perception in intelligent systems is closely coupled with action and the actual environment the system is situated in. Embodied robots exploit by means of sensory-motor coordination the environment in simplifying a complex visually guided task, yielding successful robust perceptual behavior. This *active vision* approach enables robots to sequentially and interactively select and analyze only the task-relevant parts of the total available visual scene. In this study, active vision and feature selection operating on an omnidirectional visual scene are co-evolved in simulation by a genetic algorithm, yielding neural controllers of a robotic scale car equipped with an omnidirectional camera that is capable of driving two differently shaped circuits at high-speed without going off-road. Successfully evolved individuals show the sophisticated strategies of an artificial retina selecting quickly only the task-relevant features in the accessible information-rich visual scene provided by the omnidirectional camera. The evolved behaviors of the robotic car and the corresponding strategies of its retina are analyzed, and additionally the obtained results from the car equipped with the omnidirectional camera are compared with those from the car equipped with a standard pan-tilt camera. Finally, the advantages of the used active vision approach operating on an omnidirectional camera are discussed.

Acknowledgements

The scientific results embodied by this thesis have not been created in seven days, by the sole work of the author. It is the result of a long process with much external influences; the product of the open-ended learning process of five inspiring years of study concluded by six months of dedication to the subject, situated in the stimulating environment of the Laboratory of Intelligent Systems at the EPFL in Lausanne, Switzerland.

No progression would have been possible without the many guiding, useful, and amusing interactions with my new environment during my stay in Lausanne. I truly want to thank all members of the lab for the good times and the interesting exchange of ideas. Dario Floreano of course, my supervisor, for giving me the possibility and the means to conduct this research, and Bart de Boer, my internal supervisor, for the given advice and help. But in the first place Mototaka Suzuki, my daily supervisor, with whom the cooperation has been feeling as natural as the match of scientific motivations, ideas, and preference of strong espresso.

Furthermore, the many lab movie nights (occasionally without movie) and barbecues at the lake (occasionally with movie) have always been a nice way to integrate in the lab and to relax after another week of hard work. The challenging struggle for life with the cheesy fondue and raclette at every event has made my stomach robust for nutrition-poor times, and the impressive diversity of Swiss nature forced me everyday to admire life and kept feeding the curiosity for its underlying principles.

Last but not least, I want to thank my friends and family, roommates and especially my girlfriend, for their attention and interest in all ways. They helped me in living abroad and adapting to my new surroundings.

In short, to conclude with my best French: *Merci, c'etait vachement cool!*

Vision without action is a daydream.
Action without vision is a nightmare.

Japanese proverb

Contents

1 Introduction 8

2 Theoretical Background 10
- 2.1 Philosophical Groundings . . . 10
- 2.2 Artificial Neural Networks . . . 11
  - 2.2.1 A Neuron Model: The Perceptron . . . 11
  - 2.2.2 Network Architecture . . . 12
  - 2.2.3 Learning . . . 12
  - 2.2.4 Advantages . . . 13
- 2.3 Evolution . . . 13
  - 2.3.1 Genetic Algorithms . . . 13
  - 2.3.2 Advantages . . . 14
  - 2.3.3 Evolutionary Robotics . . . 14
- 2.4 Active Vision and Feature Selection . . . 15
  - 2.4.1 Active Vision . . . 15
  - 2.4.2 Co-evolution of Active Vision and Feature Selection . . . 15
  - 2.4.3 Former Work . . . 16

3 Research Objectives 20

4 Methods 22
- 4.1 Vortex Simulation Toolkit . . . 22
  - 4.1.1 Architecture . . . 22
  - 4.1.2 Conventions . . . 23
  - 4.1.3 Process flow . . . 24
  - 4.1.4 Vortex XML File Format . . . 24
- 4.2 Car Modeling . . . 25
  - 4.2.1 The car . . . 25
  - 4.2.2 The model . . . 25
- 4.3 Circuit Modeling . . . 26
  - 4.3.1 Prerequisites . . . 26
  - 4.3.2 Conceptualization . . . 26
  - 4.3.3 Realization . . . 27
  - 4.3.4 XMLCircuitGenerator . . . 28
  - 4.3.5 Circuit Properties . . . 29
  - 4.3.6 Simplified Circuit . . . 29
- 4.4 Robot Simulator . . . 31
  - 4.4.1 Structure . . . 32
- 4.5 Modifications for the Car Robot . . . 33
  - 4.5.1 Motor Modeling . . . 33
  - 4.5.2 Camera Modeling . . . 33
- 4.6 Evolutionary Active Vision Set-up . . . 36
  - 4.6.1 Neural Network Architecture . . . 36
  - 4.6.2 Genetic Algorithm . . . 37
  - 4.6.3 Retina Movement . . . 38

4.7 Analytical Tools 38

5 Experiments and Results **40**
5.1 Pan-Tilt Camera Experiment 40
5.1.1 Experimental Set-up 40
5.1.2 Evolved Behavior 40
5.2 Omnidirectional Camera Experiments 41
5.2.1 Ellipse Shaped Circuit 41
5.2.2 Banana Shaped Circuit 41
5.2.3 The Ellipse-Evolved Individual On The Banana Circuit 42

6 Discussion **48**
6.1 Evolved Omnidirectional Strategies 48
6.2 Advantages of Active Vision with the Omnidirectional Camera 49
6.3 Future Work 49

7 Conclusion **52**

Chapter 1

Introduction

In the quest for scientific understanding of intelligent biological systems that we find everywhere and in all its diversity around us in nature, including ourselves, nowadays multiple scientific disciplines are combining their knowledge, methods and strengths in order to reveal the secrets of that what has been fascinating humanity already for a long time: intelligent life.

While philosophy, and later psychology and biology have classically focused on the research on life and intelligence, these days neurologists, computer scientists and robot engineers take part in it as well. It has been however only recently that this manifold involvement has yielded a paradigm where all disciplines can enhance, influence and benefit from each other. Computer scientists and robot engineers build biologically inspired models and robots under consideration of the underlying principles in nature that yielded intelligent systems, and results from this synthetic approach of understanding intelligence [18] influence again the analytical approach of the classical fields observing natural intelligence.

This study concerns the generation of perceptual behavior. Using biologically inspired and plausible mechanisms and methods, the synthetically obtained visually driven behavior of a robotic scale car equipped with an omnidirectional camera while driving a circuit is analyzed and discussed. This study aims to provide understanding about the principles that allow living and artificial systems to recognize features and visually interact with their environment in a self-organizing adaptive way. Furthermore the used method simplifies the computational complexity of visual processing by off-loading information in the environment, which makes this method for recognition and navigation much more efficient than the computational expensive methods of traditional computer vision accounts.

Co-development of active vision and feature selection has proven to be a successful and biologically plausible method for simplifying the computational complexity of visual processing. By using genetic algorithms to shape the synaptic connections of a deliberately simple neural network architecture with direct pathways between visual and motor neurons, behavioral machines are autonomously evolved in simulation. These machines are able to actively exploit visual features dependent on the sensory-motor contingencies related to their task in the environment. An artificial retina is evolved to select behaviorally relevant features in the visual scene to enable the driving car to stay on the road. By providing omnidirectional images for active vision to operate on, the retina can immediately access the total available visual scene in any direction in the image. The analysis of the resulting evolved behavior of the retina reveals interesting strategies and successfully manages to keep the car on the circuit.

An introduction to the basic ideas and assumptions fundamental to this study and a review of the corresponding background theories are provided in Chapter 2. Here the philosophical groundings, neural networks, genetic algorithms, evolutionary robotics, and Active Vision are assessed. In Chapter 3 the explicit research objectives will be elaborated. Subsequently, Chapter 4 explicates the used methods in answering the research question. It will discuss the used simulator and the modeling of the robotic car, its omnidirectional camera and its environment. Furthermore the evolutionary process, the neural architecture and the Active Vision set-up are elucidated. Then, the conducted experiments and its corresponding results are presented in Chapter 5, followed by the discussion on the interpretation of the results in Chapter 6. Finally, possible future directives are described in Chapter 7, concluded by a final conclusion of the study in Chapter 8.

Chapter 2

Theoretical Background

2.1 Philosophical Groundings

In the classical approach of modeling intelligent systems, inspired by psychological Behaviorism, the available computational model, and introspective intuitions, the prevailing view on cognition was that of symbol manipulation of an internal explicit symbolic representation of the outside world by a logical reasoning device - the brain. In this view perception, cognition and action are three separated sequential processes, and thus also possible to investigate separately.

This approach has been criticized currently for many reasons. Firstly, the problem of the grounding of the meaning of the symbols in a formal symbol system arises [8]. When the logical reasoning device is able to manipulate knowledge representing symbols logically, the semantic interpretation of the symbols is still left over to the human designer or interpreter, and is thus parasitic on the meanings in the human brain. To ground the syntactically manipulated symbols itself in the real world, a bottom-up grounding in non-symbolic representations directly related to the outside world is needed, namely the proximal sensory projections or its invariant features. This means, a cognitive system needs its own sensors and actuators corresponding to the physical entities in the world, without inclusion of implicit abstract information provided by the designer.

Furthermore, a logical symbol system has to process all incoming information in updating its internal model of the world to be able to start its symbol manipulation. This internal world modeling does not only take too much time in a real-time dynamic world demanding quick reactions, but also is hardly possible to achieve completely, bearing in mind the almost endless information richness of the real world. Instead, cognition emerges from the multiple parallel interactions with the world, translating every sensory input directly to an appropriate simple action in the world. These actions will change the environment and will affect the feedback sensory input, and together result in emergent intelligent behavior. Instead of a knowledge-representing approach of intelligence with internal world models, a behavior-based approach with the world as its own best model should be followed. For the grounding in the real world a cognitive system needs its direct connection with the world (*situatedness*), and to be able to affect this world by behavioral actions, the cognitive system needs to be a real embodied robot (*embodiedness*) [3].

This transition to view cognition as emerging from behavior-based interactions with the world is much more biologically plausible, as the brain is evolved to control the actions of the body in the world. It does not allow treating cognition apart from perception, action and its environment. Even more, cognition is dependent on its environment by exploiting it as an external memory, channeling the many parallel interactions towards intelligent behavior, and cannot be explained on the basis of internal mechanisms only.

This function of the environment as externally scaffolding cognition is well illustrated by the examples of solving a complicated multiplication or a jigsaw puzzle [4]. When multiplications become too difficult we use pen and paper to reduce the complex problem to a sequence of simpler problems. By using an external medium (paper) to store partial solutions, an interrelated series of simple pattern completions coupled with external storage can bring us to the final solution. While solving a jigsaw puzzle, nobody solves the whole puzzle by pure thought, determining only by reason whether the pieces fit in certain locations. We pick up the pieces,

rotate them to check for potential spatial matches, and then try out some possible candidate locations. By rotation and trying we use the simplifying feedback information from the environment to guide our partial solving interactive behaviors further towards the final problem solution.

To be able to exploit the interaction with the environment, a cognitive system should develop the appropriate *sensory-motor coordination* to generate the proper behavioral actions from the corresponding sensory stimuli, related to the current task in its environment. The act of running to catch a ball is done by simply running so that the acceleration of the tangent of elevation of gaze from catcher to ball is kept zero, which will automatically result in intercepting the ball before it hits the ground [4]. Here a simple coordination between sensory input and motor output suffices, and again no internal computation of the anticipated trajectory of the ball is done in order to know the proper location to catch the ball before coming in action. It is hypothesized that our conception of objects is grounded in the corresponding sensory-motor contingencies, the unique sequence of sensory input corresponding with active exploration of a certain object in the environment [17]. The developed conceptual knowledge of entities in the world fundamentally corresponds to the interactions of perceiving and acting in particular contexts. Research on the performance of infants on the risk of falling off a cliff shows that learned avoidance responses of babies in reaching over a cliff are specific to each postural milestone in development (i.e., sitting, crawling, and walking). Experience with the sensory-motor coordination from an earlier-developing skill is not automatically transferred to a later-developing skill, because each postural milestone represents a different perception-action system with different relevant control variables. The conceptualization of the dangerous cliff is thus dependent on their sensory-motor coordination and not, once learned, an epistemic consistent fact [1].

2.2 Artificial Neural Networks

The processing system responsible for mapping sensor input to eventually motor output in animals and humans is the brain. The brain is a vast collection of interlinked neurons, together forming a complex information-processing network. Neurons are simple processing units communicating with one another across synapses with nerve impulses. Multiple branched dendrites receive input signals, which are turned into one output signal sent over the axon to possibly many synaptic contacts of target cells. At a synapse the signal is transmitted by either an inhibitory or an excitatory neurotransmitter to a dendrite of another neuron. A nerve signal will only trigger the target cell when the neurotransmitter is able to cause the cell to reach its threshold potential. See Figure 2.1 for a schematic representation of the neuron structure.

Artificial Neural Networks [9, 7] are a biologically inspired computational model of the brain and imitate the information-processing principles of a biological neural network. An artificial neural network consists of several simple processing units (neurons) connected by weighted links (synapses) for transmitting signals.

2.2.1 A Neuron Model: The Perceptron

A perceptron processing unit is a simple model of a neuron [15], representing the same basic computational function. The output of a unit y_i is a function Φ of the sum of all incoming signals x_j weighted by connection strengths w_{ij}:

$$y_i = \Phi\left(\sum_j^N w_{ij}x_j\right) \tag{2.1}$$

Most frequently used activation functions Φ are:

- the binary *step function*, returning 1 if the weighted sum is larger than a given threshold, otherwise 0 (or -1);
- the *graded linear function*, $\Phi(x) = kx$. More information is transmitted due to the graded output;
- the *sigmoid* $\Phi(x) = \frac{1}{1+e^{-kx}}$ or $\Phi(x) = tanh(kx)$ functions. The sigmoid function has a graded non-linear output, automatically scaled between 0 and 1. The constant k sets the slope of the response, approximating either more the linear function or more the step function. The $tanh(kx)$ function has similar properties, but with asymptotes at -1 and 1.

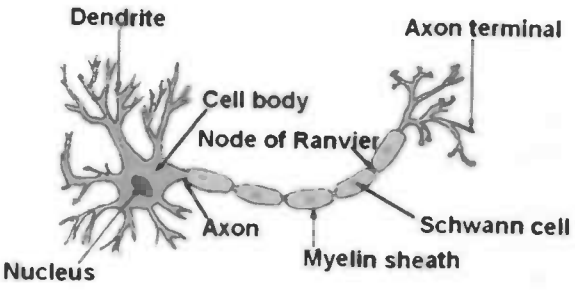


Figure 2.1: The structure of a typical neuron

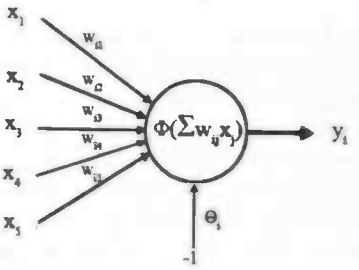


Figure 2.2: The perceptron, a computational equivalent of the biological neuron

To simulate a neuron's threshold potential, a threshold θ_i can be included as the minimal activity level of the total weighted input at which the perceptron becomes active. In the case of a continuous activation function, this can be implemented as an additional weighted incoming connection (a *bias*) from a unit with a fixed value -1. The weight of this connection determines the threshold level, since this value is subtracted from the other weighted input, and can be treated by learning methods like the weights of every other connection.
See Figure 2.2 for a schematic representation of the perceptron model.

2.2.2 Network Architecture

A network of *perceptrons* connected by their weighted links is able to learn to map an incoming pattern of signals on one or more outputs, and can consequently process information, adapt to a context, generalize, extract features and classify patterns. The network learns when the connection weights are adapted in a learning process towards generation of the appropriate output of any given input pattern.

Several network architectures are possible. Commonly, artificial neural networks are structured in layers of units that are processed simultaneously. The input layer represents the units that are fed by the input pattern, the output layer contains the output units and the intermediate layers (*hidden* layers) can serve as extra internal computational abstraction steps. In *feed-forward* architectures the signals flow forward through the layers from input to output units, with every layer being updated after the former. In *recurrent* architectures however, units can be fed by feedback connections from a unit from an upper layer or by its own signal with a time delay, resulting in memory-like temporal dynamics.

2.2.3 Learning

Adaptation of the set of connection weights in a learning process is possible with different learning methods. A learning algorithm achieves learning by repeatedly modifying the weight values at a small rate every time a sample pattern is presented during a learning phase. In the case of supervised learning algorithms, pairs of input and corresponding output patterns are presented to the network initialized with weights at or around zero, and subsequently the weights are repeatedly modified based on the discrepancy between the current output and the desired output. In the case of unsupervised learning the desired output data is lacking and weights are updated only on the basis of the input training patterns, forming an abstracted topological representation of the input space.

Another method to set the weights is by letting an evolutionary process determine the appropriate weights. In behavioral robotics the behavioral input/output mapping is not specified on the level of the individual input/output patterns from sensors to motors per time step, and it is therefore impossible to give a detailed specification of the desired output response for every individual input pattern, needed for supervised learning. However, behavioral learning should be based on correlating input patterns with the desired behavior, which can be achieved by an evolutionary search strategy selecting the satisfactory results. The evolutionary approach is elaborated in the next section.

2.2.4 Advantages

The use of artificial neural networks in behavioral robotics is beneficial in many ways. Firstly, they are biologically inspired and therefore a logical and appropriate choice as a control system for generating behavior corresponding with what we see in nature. The parallelly distributed information processing is not solely dependent on single elements and yields robust, fault tolerant computation, not easily disturbed by erroneous signals due to noise in the robot's sensors or its environment. They are very well gradually adaptable to little changes in the resulting behavior without loosing immediately the formerly achieved performance level, and thus suitable for replicating adaptive behavior. Furthermore, the possibility of time-dependent recurrence in the networks makes temporal and memory based behavior possible.

2.3 Evolution

The Evolution Theory initiated by Charles Darwin in 1859 explains the existence of the extraordinary variety of highly adapted life forms in nature. In short, an organism's *phenotype* (its physical appearance and constitution) is genetically defined by its *genotype*, which is inherited partly from each of both of its parents. This *crossover* process together with *mutation* (an error during duplication or translation of genetic material) yields genetic *variation* in the population. *Natural selection* ensures only the survival of the fittest (most well adapted) of all the variant individuals. That is, only organisms that are able to survive are able to reproduce and increase the frequency of their genes in the gene pool, while organisms poorly adapted to their environment will become extinct.

2.3.1 Genetic Algorithms

A computational abstract model of the biological evolutionary process is a genetic algorithm [10]. A genetic algorithm correspondingly operates on a variant population of individuals, defined by their artificial genotype, by selecting the fittest individuals for reproduction, simulating crossover and mutation. By repeating selection on every new generation, the average fitness of the population will increase, as only the most successful individuals are allowed to reproduce and thus pass their genes to the next generation.

In the case of evolving the connection weights of a neural network, the *artificial genotype* is a (binary) string encoding all the weight values in the network. Selection is done by means of a *fitness function*, which mathematically defines the performance level of every individual. For every individual in the population its personal genotype is translated to its phenotype (i.e., the connection weights of the robot's neural network are set correspondingly to the prescriptions in the gene string), and the resulting candidate is tested in the environment to allow the calculation of the fitness according to the fitness function. After all individuals in the current generation have been evaluated on their fitness, selection for reproduction in the next generation starts. Various selection methods exist:

- *The roulette wheel method*:
  The individuals are selected with a probability proportional to their fitness. Higher fitness means a higher probability of generating offspring.
- *Rank based selection*:
  Individuals are sorted from best to worst and the probability of being selected is proportional to their rank.
- *Truncation selection*:
  Individuals are also sorted, but only the N best individuals are being selected.
- *Tournament based selection*:
  Individuals are selected by randomly taking pairs of individuals from which either the one with the highest fitness or the one with the lowest fitness is selected for reproduction with a given probability distribution.
- *Elitism*:
  The best individual of the current generation is preserved for reproduction in the next generation. This additional method ensures that the best solution found so far is not lost.

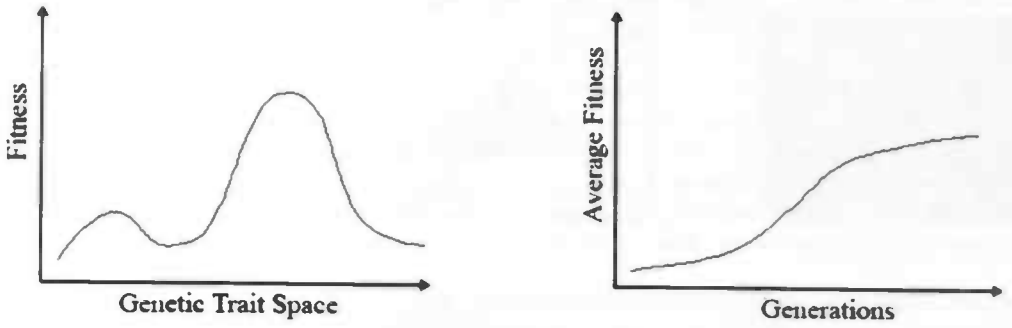


Figure 2.3: The fitness landscape (left) represents the fitness mapping corresponding to the total available genetic trait space, including all possible different individuals. A genetic algorithm climbs over generations (right) the hills in the fitness landscape in search for the top, where the genetic definition of the best performing individual is located.

Once the candidates for reproduction are selected, the genotypes for the next generation can be generated by randomly pairing the parental genotypes and applying crossover and mutation. The crossover operator will select with a given probability a random point along the gene string where the following genetic material is swapped between the two parents. Mutation consists of changing each value within the genotype with a given probability. In a binary representation this means the flipping of the selected bits. A genetic algorithm is halted when the fitness of the population stops increasing or is at a sufficient level of performance.

2.3.2 Advantages

When one would map all permutations of the genotype (i.e., all possible combinations of genetic traits) on the corresponding resulting fitness of their phenotypes, the *fitness landscape* of the population is acquired (see Figure 2.3). Evolutionary search is aimed at climbing the hilly landscape to the top with the highest fitness, where the corresponding genotype defines the best performing individual. Genetic algorithms efficiently explore the fitness landscape by isolating and combining partial solutions, instead of exploring all possible combinations of genes. Selective reproduction provides a higher presence of offspring from the genotypes of the individuals with higher fitness. Due to the crossover operator the partial solutions are combined, generating innovation and in this way exploring a large space of solutions. The purely stochastic mutation operator is merely a local search operator, but helps overcoming local maxima in the fitness landscape and ensures the variance in the population needed to be able to deal with a possibly changing selective pressure.

2.3.3 Evolutionary Robotics

The use of a genetic algorithm tuning a neural network as a controller for autonomous behavioral robots is very convenient. Already mentioned is that no detailed supervised learning is needed for the neural network to be able to cope with its task in life. The evolutionary tuning suits the embodied and situated function of the neural network and makes it possible to co-adapt the neural structure and the structure of sensory-motor coordination, driven by the interactions with the environment. Only the fitness function, the networks architecture and the genotype to phenotype mapping have to be defined and the rest is done by evolution in a self-organizing process[1]. Even more, the more detailed and constrained evolution is, the less space is left for emergence and autonomy of the resulting behavior.

However, the problem of a too general fitness function is that either (1) uninteresting behavior emerges that does not include the desired competencies, because all interesting and uninteresting behaviors will satisfy the fitness criterion with high fitness and selection will not make a difference anymore, or (2) evolution will fail to find solutions with high fitness at all, because selection in the simple stochastically 'dumb' first generations of individuals cannot find enough guidance in the complex behavior rewarding fitness function at the stage of

[1]In principle no constraints at all on what can be part of the self-organizing learning process are introduced by the evolutionary method itself, and even the characteristics of the sensors and actuators, the neural architecture and the morphology of the robot can be part of the evolutionary process. However, technical limits or the experimental set-up often restrict the freedom of the self-organization.

simple behaviors. The latter is known as the *bootstrap problem* and can be overcome by incremental evolution: gradually rewarding subparts of the desired task. On the other hand, by using incremental evolution more constraints and assumptions are made by the developer, which naturally decreases autonomy and emergence. The generality of the fitness criterion in the former is the case in nature. In natural evolution the only selection criterion is the ability to reproduce. Nevertheless, natural evolution produced several kinds of individuals with very sophisticated competencies. Those competencies are all serving this one fitness criterion though. The complex diverse solutions are achieved by the rich ever-changing dynamics in nature, the interactions with the natural environment, the developmental, ecological, and social processes and contingencies. Exact understanding and possibly replication of this level of sophisticated diversity is still a big challenge.

Another advantage of using artificial evolution is that changes can occur at two different time scales. The behavior of individuals is affected by genetic adaptation over generations of individuals (*phylogenesis*) and can in addition be adapted by lifetime learning of every individual (*ontogenesis*). The phylogenetic learning might deal with the large changes in the neural architecture to adapt the species to their environment and helps adapting to relatively slow changes in the environment, while the ontogenetic learning tunes every individual during its life in their personal environmental interactions and helps adapting to fast temporarily changes in the environment.

2.4 Active Vision and Feature Selection

The field of Computer Vision, enrolled in the computational perception of visual images, traditionally processes images on static features like edges, frequencies, and contrast and illumination differences by applying filters, transforms and masks. The assumption is that all information about the environment can get extracted from single images, while neglecting the importance of behavioral action and time-dependent interactions with the environment the visual information refers to.

2.4.1 Active Vision

A more embodied and situated approach to computational perception is called Active Vision, which emphasizes the role of vision as a sense for robots or other real-time perception-action systems. Its notable characteristic is the sequential and interactive process of selecting and analyzing useful parts of the total visual scene. This process simplifies the computation by selecting only the characteristics of the total available visual scene that are relevant at that moment for the task to be solved, and thus reduces the information load.

The sensory-motor coordination of the robot is organized in such a way that it exploits the interactions with the environment to simplify a complex task. Partial solutions are left behind in the environment or restructure the environment to free the system of the need to memorize or overlook the whole problem space. When revisited subsequently, the environment gives guidance to the following temporally relevant partial behavioral step.

Active Vision is a biologically plausible method, used by many perceptional systems in nature. Insects tend to recognize objects by moving their body to bring selected parts of the image within matching receptive fields, and humans scan their visual scene with sequences of saccadic eye movements specific to the task at hand. It is an illusion we see everything in our visual scene, we constantly revisit the environment to select the information currently needed. In astonishing change blindness experiments [17] it is shown that when attention is distracted from a normally clearly perceptible obvious change occurring in full view in the visual scene, observers mostly failed to perceive the change, while once the change is known, it is unconceivable to be able to miss it. Perception is thus actively selecting only that parts of the visual scene needed for the given task.

2.4.2 Co-evolution of Active Vision and Feature Selection

While Active Vision is able to select the temporally needed visual information, feature selection instead determines what features of the selected visual information are to be processed by the system. Biological systems also filter their visual input to enhance features that are relevant to the task to be solved and discard the rest, by means of receptive fields that maximally respond to only some properties of the image. As the behavior of a perceptual system is determined by its visual input and at the same time this behavior also affects the gathering of this visual input, both interdependent Active Vision and feature selection methods should be co-developed.

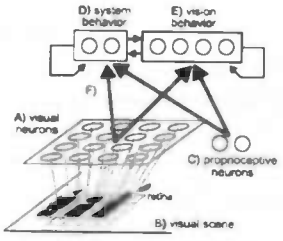


Figure 2.4: The general neural architecture of the Active Vision systems.

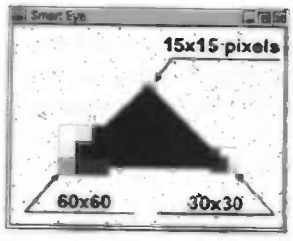


Figure 2.5: The shape discrimination experiment.

Figure 2.6: The car driving experiment.

As explained, artificial evolution is a suitable method to develop solutions with dynamically interdependent parameters without constraining supervision, as it does not explicitly separate perception from behavior.

Floreano et al. [6] investigated the co-evolution of Active Vision and receptive fields using behavioral robotic systems equipped with a primitive retinal system and deliberately simple neural architectures. They showed that complex visual tasks, such as position- and size-invariant shape recognition and navigation in the environment could be solved by individuals exploiting position- and size-variant receptive fields by actively searching and maintaining simple features of the visual scene over sensitive areas of the retina. They describe the following three studies, which illuminate the concept of co-evolution of Active Vision and feature selection and are the preceding studies which the study described in this thesis proceeds further on.

2.4.3 Former Work

The neural architecture and evolutionary method in the three Active Vision and feature selection experiments are similar. A deliberately simple discrete-time feed-forward network with recurrent connections at the output layer and evolvable thresholds (the bias) and connection weights is used (see Figure 2.4). The input layer consists of (1) a set of visual neurons (the retina), receiving the grey-level information of the pixels corresponding to their non-overlapping receptive fields arranged on a grid in the image, and (2) the proprioceptive neurons, giving feedback about the current state of the vision system. The output layer consists of neurons generating the system behavior and neurons generating the vision behavior. These output neurons all have recurrent connections to be able to associate the behavior with the former time step. The vision behavior includes besides the control of the movement of the retina in the visual scene also adaptation of the size of the receptive fields (zooming factor) and the filtering method of the pixels corresponding to one receptive field (processing the average grey-value of all corresponding pixels or taking one sample pixel representing the whole receptive field). The zooming factor can never equal the total available visual scene, in order to preserve the need for active visual movements. The thresholds and connection weights are encoded in a binary genotype string to be evolved using a genetic algorithm. Truncation selection is used as the selection method, i.e. the best 20% individuals are selected for reproduction, crossed over and mutated. A copy of the best genotype of the previous generation is inserted in the new population (elitism).

Shape Discrimination

In this set of experiments is shown that sensitivity to very simple features is co-evolved with, and exploited by, Active Vision to perform complex shape discrimination. The system is evolved to be able to discriminate between size-variant triangles and squares that randomly appear at a location in the visual scene (see Figure 2.5).

The neural architecture consisted of the input layer of a 3×3 sized retina and a proprioceptive input neuron signaling whether the retina tries to move beyond a boundary of the visual scene. In the output layer, four vision behavior neurons affect the movement of the retina in distance and angle, the zooming factor and the filtering method, and two system behavior neurons indicate whether a triangle or a square shape is recognized. A population of 100 individuals was evolved for 150 generations, with a fitness function proportional to the number of correct responses of an individual. Each individual was during its life exposed to 20 images, 10 of which contained a triangle and 10 a square.

The resulting behavior of the best individuals after evolution consisted in starting to scan the visual scene with the retina until a shape is found. Then the retina slides back and forth along one of its vertical edges to discriminate a square with a straight edge from a triangle, or makes use of another strategy to scan the corners of the shape. Mostly a sampling filtering method was used to enhance to contrast between the shape and its background and the zooming factor changes the retina resolution continually to be able to recognize all variant sizes of shapes.

In a comparing study to let stationary multilayer neural networks (provided with the whole visual scene to its input layer) learn the same task by a supervised learning algorithm, no neural architecture deprived of Active Vision seemed to be capable of learning the discrimination task. The resulting performances were at chance level, instead of the 80% correct performance of an Active Vision system.

Car Driving

In this set of experiments, the same co-evolutionary method and architecture are applied for driving a simulated car over roads in the Swiss Alps and is shown that Active Vision is exploited to locate and fixate simple features while driving the car (see Figure 2.6) [20].

The neural network is provided with a 5×5 retina, two proprioceptive neurons encoding the vertical and horizontal position of the retina in the visual scene, which is the view through the windscreen of the car. Furthermore, two vision behavior output units regulate the horizontal and vertical displacements of the retina, and two others the filtering method and zooming factor. Two output neurons of the system behavior control the motor commands, steering and forward/backward acceleration. A population of 100 individuals was evolved for 150 generations to drive the car on three different circuits with a fitness function rewarding the total traveled distance of an individual across all circuits during trials of maximally 60 seconds. When an individual drives the car off road, the trial is truncated and consequently gets assigned less fitness, as it cannot drive further.

The best-evolved individuals behave according to two different strategies. The first strategy tries to keep the retina, tracking constantly the vanishing point of the far edge of the road in the visual scene, at a constant vertical position in the visual scene. A vertical displacement of the tracking retina (meaning an approaching edge of the road) is correlated with a correcting steering movement. The other strategy consists of a correlation of shifts of the edge of the road within a totally zoomed out retina covering one side of the visual scene, with appropriate motor output.

The performances of these individuals resemble or outperform the performances of well-trained human drivers tested on the same circuits.

Robot Navigation

In this set of experiments, again the same co-evolutionary method and architecture are applied to navigation of an autonomous robot equipped with a pan-tilt camera. The behavior of a real mobile robot is evolved in order to let it navigate collision-free as far as possible in a small square arena surrounded with walls.

A 5×5 fixed retina received input from its visual fields with a fixed zoom factor in the middle of the camera image. As the pan-tilt camera is able to move itself, it is not necessary to move the retina in it, and the zooming behavior is omitted to reduce the number of evolvable parameters and thus shorten evolutionary time on the physical robot. Two proprioceptive neurons encode the actual pan and tilt angles of the camera. Two visual behavior output neurons control the pan and tilt displacement speeds of the camera and another neuron sets the filtering method. The system behavior neurons include two neurons controlling the motor forward/backward rotational speeds of the two wheels. The robot is two-wheel driven and can therefore turn on the spot. A population of 40 individuals was evolved for 15 generations on the real robot, each generation taking approximately one and a half hours. The fitness function selected individuals that could maintain the highest forward speed with the lowest difference between the speeds of the two wheels during the whole trial of 60 seconds. Individuals were thus encouraged to navigate as fast and straight as possible. Of course it is not possible not to turn at all, as it should avoid hitting a wall, therefore a collision triggers immediately a truncation of the current trial and adds zero fitness to the remaining trial time.

The evolved strategy of the robot consists in maintaining the detection of the edge between the dark floor and the white walls by pointing its camera towards an approaching wall. The expanding amount of white in the visual fields is used to slow down one of the wheels to turn away from the approaching wall.

Evolved individuals in all experiments exploit Active Vision and simple features to direct their gaze at invariant features of the environment and perform the appropriate system behavior. All used features are linearly separable categories of the input vectors, which is hardly surprising because visual neurons project directly to output neurons without intermediate hidden layers. The recurrent connections at the output layer provide time-dependent dynamics at the behavior level and make it possible for the individual to relate its conceptualization and response to the current action context.

Chapter 3

Research Objectives

Active Vision is able to select and analyze only the relevant parts of the total available visual scene. This study will apply the Active Vision approach to the domain of perception of the environment through an omnidirectional camera. An omnidirectional camera can provide visual information from every possible direction in the same time and is therefore potentially advantageous over normal pan-tilt or static camera's that cannot access the surrounding environment that easily. However, the advantage cannot be exploited as long as the excessive amount of information is not filtered to reveal the useful properties of the total supply. Co-development of Active Vision and feature selection seems pre-eminently appropriate for this purpose.

> *The objective of this study is to co-evolve Active Vision and feature selection in a robotic scale car equipped with an omnidirectional camera in a simulated environment, evaluated to drive a circuit at high speed without going off-road.*

The mobility of a racecar enables us to investigate computationally effective visual processing during its high-speed navigation and to conduct experiments in relatively large terrains. As stated, Active Vision operating on omnidirectional images allows evolved artificial retinas to select important features in a broad field of view without mechanical camera control and is therefore a nice test bed for development of sensory-motor coordination.

The omnidirectional camera consists in this case of a standard camera facing up towards a hyperbolically curved mirror. Hence the obtained omnidirectional image yields a geometrically distorted representation of the whole environment around the car, with the car itself big in the middle and the farther surroundings further pressed together around it. To match the circular distortion of the image with the receptive fields selecting retina, this retina will correspondingly be defined in circular polar coordinates. Since Active Vision has to deal with a new kind of visual representation,

> (1) we will investigate how the resulting sensory-motor coordination deals with the omnidirectional visual information, and what kind of strategies are developed to accomplish the given task of driving the circuit, and
>
> (2) we will compare the emerged solutions with the solutions a car with a normal pan-tilt camera will yield in the same situation, to examine the possible advantages of the use of the omnidirectional camera over the use of the pan-tilt camera.

In contrast with the biological inspired modeling and methods of the used approach, the use of an omnidirectional camera does not seem biologically plausible. Natural evolution favored a vision strategy using two eyes with a limited field of view. The way in which we are using this by human engineering constructed sensor is not totally biologically inconceivable though. One could well imagine such a kind of visual sensor could, once arisen in the evolutionary process, be exploited by natural systems. Moreover, the mechanisms we can find in biology are not necessarily optimal for a given task and/or do not always represent the only way of achieving a given functionality [16]. Following the same principles and dynamics that form and affect nature, the use of currently non-existing but imaginably possible mechanisms can be well defended, bearing in mind the stochastic character of evolutionary solutions.

Furthermore, to quote Christopher Langton, the founder of Artificial Life: "Only when we are able to view

life-as-we-know-it in the larger context of life-as-it-could-be will we really understand the nature of the beast. Artificial Life is a relatively new field employing a synthetic approach to the study of life-as-it-could-be. It views life as a property of the organization of matter, rather than a property of the matter which is so organized." [14] Therefore, studying the visual systems that could be possible can serve us in a deeper understanding of the visual systems that we already know.

The evolutionary process takes place in simulation. However, the simulated car is modeled after its equivalent in reality, in order to make possible a future transfer of the behavior evolved in simulation to a real car in a real environment. Especially since we have stressed the importance of the environment, with which evolved intelligent behavior is closely coupled, the realistic modeling of the environment and its interactive physical properties is cautiously taken care of, both for yielding realistic results and for leaving the possibility for a future transfer to reality.

Chapter 4

Methods

4.1 Vortex Simulation Toolkit

Evolutionary processes take a long time. For this purpose and because of the technical constraints, the Active Vision systems are evolved in a simulation based on the Vortex Simulation Toolkit (CMLabs Simulations Inc., *http://www.cm-labs.com*). Vortex is a real-time[1] physics-based 3D visualization and simulation toolkit developed by CMLabs, providing a set of libraries for robust rigid-body dynamics, collision detection, contact creation, and collision response [12]. By using the Vortex libraries for building a simulation in which the controllers of the robotic scale car can be evolved, physical interactions between the environment and the robot can be realistically simulated. It ensures the realistic behavior of modeled objects following the laws of physics, including natural movement and preventing models from passing through each other. Moreover, the view from a robot's vantage point can be rendered well, based on OpenGL [11].

4.1.1 Architecture

Vortex is a cross-platform set of C++ libraries for IRIX, Windows, PS2 and Linux platforms, providing the physics engine for to be developed applications. Developers can easily add and integrate their simulation specific C++ code. Vortex consists of four different modules: static geometrical definitions and dynamic behaviors of objects are separately described in the collision detection module and the dynamics module, the simulation module serves as a higher layer bridging the two, and the rendering module is responsible for the visualization.

Collision Detection Module

The Collision Detection Module (MCD) contains collision detection and contact creation algorithms as well as related utility functions. For this purpose, the geometrical definitions of all entities in the simulated world have to be defined. Vortex supports a number of primitive geometry types, meshes and a few special types. The available geometrical primitives are: a box, a cone, a cylinder, a plane and a sphere. The mesh geometry types allow you to specify every arbitrary geometry shape as a set of polygonal meshes. The special types include a Regular-Grid Height Field representing a rough terrain as a set of z-values varying in a regular grid of locations in the x-y plane, and the composite type in which different types can be combined to create a unique collision type. Each object in Vortex is defined by a *collision model* containing the *collision geometry types*. Such a *collision model* holds a transform that defines the position and orientation of that shape in the global coordinate system and is the fundamental type in Vortex to which dynamics and user-defined properties are attached. It also defines the kind of material of the model, by referring to the material table in the simulation module.

Simpler geometry types require less memory and simpler algorithms that operate on them. In choosing the geometry types to define an object with, one should base the complexity on the trade-off between performance, memory and geometrical accuracy. When it is important that a simulation takes place in real-time, an approximate modeling should be chosen, consisting of composite models of geometrical primitives.

[1] Real-time means the simulation runs at the frame rate of a visual display, around 30-60 Hz.

Dynamics Tools Module

The Dynamics Tools Module (MDT) is a library providing complete specifications of the physical interactions between objects and joints and their environment. Every *collision model* can be combined with a dynamics definition, the *dynamics body*, to make a complete representation of a rigid body. This *dynamics body* provides the parameters necessary for physical movement, like mass, center of mass, applied forces and impulses, velocities and accelerations.

This module also contains the constraints to limit the freely moving of the rigid bodies. Bodies interact with other freely moving bodies through contact constraints. Vortex automatically generates contacts between bodies by the collision detection module and passes those to the dynamics solver. *Contact constraints* restrict the motion at one or more points where two geometrical objects have points or lines or planes that are within a distance tolerance, i.e. they have an unrealistic overlap. *Joint constraints* are constraints that attach two bodies to each other or one body to the world, creating articulated bodies. They restrict the position and orientation of a body related to another body. Some possible joint constraints are: angular joints, hinge joints, prismatic joints, ball-and-socket joints, motorized joints, spring joints, and car wheel joints. The car wheel joint is modeling the behavior of a car wheel, with steering and suspension.

Simulation Tools Module

The Simulation Tools Module (MST) is a high-level library providing the main entry point to Vortex and a bridge between collision detection and dynamics tools. This module provides an API which is the preferred interface to Vortex, integrating the lower-level functions of the dynamics and collision modules and managing the data flow between collision detection, contact creation and dynamic response. Furthermore, MST maintains the material table, defining the unique way of interacting between every possible pair of materials that exists in the simulation. It contains dynamic contact properties like friction and restitution between the two materials.

Vortex Viewer

The Vortex Viewer (MeViewer) is the visualization tool, using OpenGL or DirectX for rendering. It is a basic 3D rendering and interactive viewing tool that is independent of the simulation part of Vortex. It is possible to use any other renderer for visualization of the dynamics and collisions. For every *collision model* a *graphical object* has to be created. Such an object contains also information about the color and texture of the rendered object. Colors are specified as RGBA, that uses relative intensities varying between 0 and 1 for the primary colors red, green, and blue, and a transparency value alpha. An alpha value of 0 represents complete transparency regardless of the color values, and an alpha value of 1 complete opacity. The Viewer supports a maximum of 25 different textures. These should be $128 \times 128 \times 24bpp$ Windows .bmp files. It also supports 256×256 images, but as this takes four times the memory, it reduces the maximum possible amount of textures proportionally.

Like most graphics applications, MeViewer uses a left-handed Cartesian coordinate system, with x increasing from left to right in the screen, y from bottom to top, and z into the screen. The point of view of the simulated world, can be changed by setting the look-at point of the camera to a specified world position. The camera position can be interactively changed during the simulation using spherical polar coordinates to specify a position relative to the specified look-at point. It is possible to zoom in and out, to pan, tilt and elevate the camera.
Lighting in the simulated universe is possible in three ways: one ambient light source, two directional light sources, or one point light. Furthermore, there are several possibilities for interacting with the simulation by menu management, mouse and joypad controls, and a displayed performance bar.

4.1.2 Conventions

There is no standard convention of units in Vortex, the developer can choose any system of units, provided the consistency of values in the system. In the case of the simulation for the robotic scale car a centimeter-grams-seconds unit system is chosen. Vortex uses Cartesian coordinates for virtually all vector quantities. The position and orientation of a body can be adjusted by applying a relative transform between the body and the global reference frame. This transform consists of a translation vector and a 3×3 orthonormal rotation matrix. Equivalent to this, it is possible to use a 4×4 transformation matrix, containing both rotation and translation information.

4.1.3 Process flow

After an initial setup, Vortex calls a *stepper* function to update the simulation. As already stated, MST takes care of managing the data flow between collision detection, contact creation and dynamic response. Every time step it updates the transformation matrices of each *collision model* and its corresponding *dynamics body*. Then data about intersections and contact points are obtained for each collision event. Vortex will first perform a rough culling by building lists of pairs of objects that are close to each other. Afterwards geometrical overlap is determined and contacts are generated. Subsequently the dynamical response is computed based on the collision contacts and the dynamical information appropriate to the given pair of models that are in contact. Finally, the positional information of every *dynamics body* is passed to the *graphical objects*, which are rendered by the Viewer.

A time step is an interval of time and should be equal or shorter than the frame rate of your monitor to create a real-time simulation in which everything seems to move in the same rate as in the real world. The simulator uses current velocities and forces and extrapolates them to compute an approximation of the state into the future. Although the approximation is more accurate with a small time step, the simulation will also take more memory. Note that Vortex is able to compute a collision detection between two time steps, which ensures physical objects not to be able to pass through each other even when the time step is high.

4.1.4 Vortex XML File Format

To make it easier for the developer to define a simulated world, Vortex provides the possibility to describe the four kinds of Vortex data (i.e. dynamics bodies, constraints on those bodies, collision geometry and collision models) in an XML structured `.me` file [13]. When compliant to the appropriate structure, the Vortex loader[2] will take care of making instances of the described objects in the Vortex modules and setting all their properties.

Core Structure

The XML format of the `.me` file requires always the outer namespace `<ME>`. With the `<IMPORT>` element it is possible to include any number of other `.me` files, allowing a hierarchical construction of atomic units assembling a greater whole and multiple use of a definition in a single simulation. Within the `<CORE>` element the dynamics data (`<MDT>`) and the geometry and collision detection data (`<MCD>`) can be specified. In the `<MST>` element the material table is defined. The `<ME_APPS>` element contains non-core Vortex data like the Vortex Viewer data in `<RENDER>` and the `<USER_APPS>` element specifies all non-Vortex data. All of these sections are entirely optional.

A basic overview is given below.

```
<ME>
   <IMPORT>
      <ME_FILE> ... </ME_FILE>
   </IMPORT>

   <CORE>
      <MDT>
         <WORLD> ... </WORLD>
         <BODIES> ... </BODIES>
         <CONSTRAINTS> ... </CONSTRAINTS>
      </MDT>
      <MCD>
         <GEOMETRIES> ... </GEOMETRIES>
         <MODELS> ... </MODELS>
      </MCD>
      <MST>
         <MATERIAL_TABLE size="n"> ... </MATERIAL_TABLE>
      </MST>
   </CORE>

   <ME_APPS>
      <RENDER> ... </RENDER>
```

[2]To load the specified world from the XML file, the existing Vortex based simulator code uses the parameter `-file` *name*`.me`

Figure 4.1: The original real car.

Figure 4.2: The modified real car.

Figure 4.3: The resulting car object in simulation.

```
    </ME_APPS>

    <USER_APPS>...</USER_APPS>
</ME>
```

Rendering

In the RENDER section it is possible to define the textures of the graphical objects using the <TEXTURE> element. The contents of this element should be a 128 × 128 or 256 × 256 × 24*bpp* Windows .bmp file, located in the Resources directory. The rendering of the surrounding background landscape can be defined with a similar <TEXTURE> element within the <R_SKYDOME> element. The camera settings (look-at point, distance, angle, elevation) can be set in the <R_CAMERA> element in the <RENDER> section.

World.me

To define the world with the models appropriate for this study, the World.me XML file sets the global physical parameters (e.g., gravity, the material table) as in the real world, defines a background rendering which is natural and not distracting (i.e., low contrast differences) and imports the circuit objects and the car model object, which are described in the next sections.

4.2 Car Modeling

4.2.1 The car

The real robotic scale car is a modified commercially available radio controlled electric powered 4WD car model (scale 1:10), equipped with an embedded CPU and omnidirectional camera. Its length is 370 mm, its width 195 mm and its height 85 mm. It has a wheelbase of 260 mm and a ground clearance of 15 mm. The gear ratio is 7.6:1 and it weighs approximately 1600 grams. On top a tube providing the convex mirror for the omnidirectional camera is mounted, 80 mm high with a radius of 10 mm [2]. See Figure 4.1 and Figure 4.2 depicting the real car. See for more information about the real car, including technical details, control procedures, and C++ code for joypad control over a wireless connection the car report [22].

4.2.2 The model

As stated earlier, in modeling a complex object, the trade-off between simulation performance (i.e. time and memory consumption) and geometrical accuracy should be considered, regarding the constraint of real-time performance. Accordingly, as real-time performance is very important in the case of the simulation of the behavior of the robotic scale car, the modeling of the car consists of simple geometrical primitives together forming a composite model. A simplification in geometrical shapes causes few problems, because the important basic physical features and interactions (e.g. collision, gravity, acceleration, etc.) on which the evolutionary algorithm should base a robust solution still possess similarity in both the simplified simulation and the reality. The car is built up out of three boxes, a wheelbase, a main box and a drivers compartment (see Figure 4.3). The four torus formed wheels are connected by four axes to the wheelbase. The camera tube is a cylinder on top of the car. This approximate model allows the simulation to run fast enough, without loosing the fundamental features intrinsic to the car. See Figure 4.4 for the exact specification.

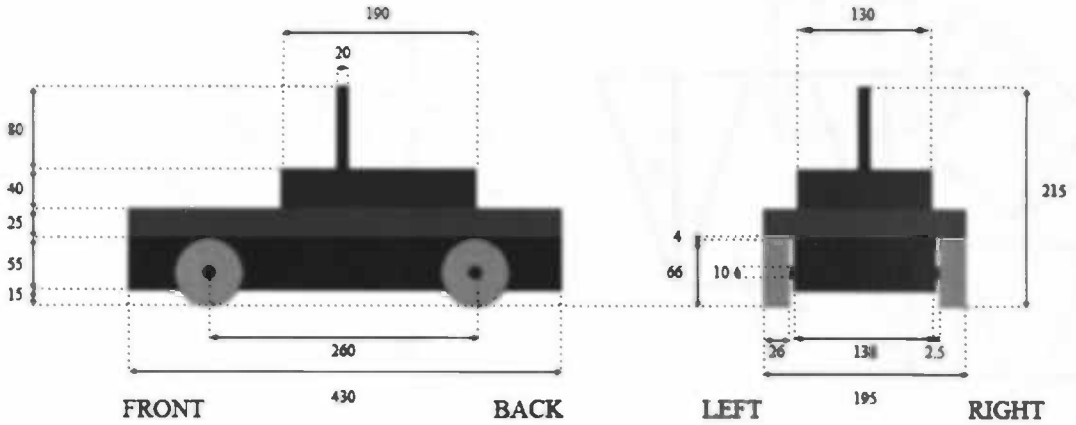


Figure 4.4: Specifications of the simulated car object, sizes in mm.

In the dynamics section of the XML file defining the modeling of the robotic scale car, the body is defined by specifying its mass and the joint constraints for the car wheels. In the collision detection part the primitive geometry types and sizes, and the corresponding collision models are described, resulting in a composite model combining all primitive elements. Finally the rendering is defined. The resulting rendered car object is depicted in Figure 4.3.

4.3 Circuit Modeling

4.3.1 Prerequisites

The behavior of the robotic scale car will be evolved in simulation to drive the car on a circuit without going off-road. This requires a sufficiently broad circuit which can be visually differentiated in a low-quality greyscale image. In former research it turned out that simple reactive behavior is obtained quite easily by the evolutionary process, as in correlating compensatory motor output in a driving task directly for small shifts in the position of the white sideline of the road in the visual input. However, when the environment is less predictable, e.g. with discontinuities in the sideline contrasting the edge of the road, evolution is forced to find a solution which consists of less reactive, more robust behavior. For this reason it is important to be able to create a non-trivial environment, with different types and directions of curves and non-permanent or changing contrast and sidelines between road and environment. To check whether the car drives off-road, one possibility is to omit the off-road ground. Thus, driving off-road will result in the car falling down, simply measured by the change of position on the z-axis.

4.3.2 Conceptualization

As Vortex does not provide a standard way to define a road, the circuit also has to be constructed out of primitive geometrical elements. A straight piece of road can be made by using a very flat box, a curve needs one or more triangular shapes. However, simple triangular shapes do not exist in Vortex, but there is a way to construct a trapezoid shape, which can serve as a curved piece of road too. The trapezoid shape is constructed placing two mirrored parallelogram shapes and a rectangular shape on top of each other. The parallelogram shapes are constructed by transforming the rectangular shape in the horizontal plane. After the deformation, this shapes are translated to yield together a trapezoid shape.

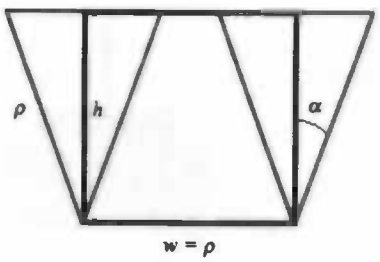


Figure 4.5: Trapezoid shape consisting of rectangle and parallelograms.

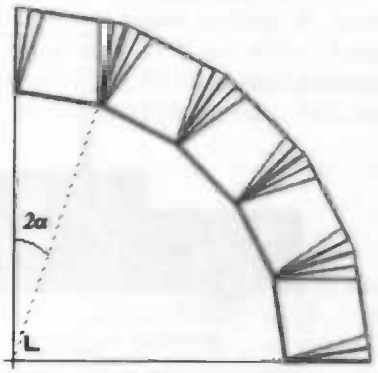


Figure 4.6: 90 degrees curve, with $N = 5$.

4.3.3 Realization

Trapezoid shape

As stated, the trapezoid shape is constructed out of three rectangular shapes, two of them deformed to yield parallelograms. The shape of the trapezoid depends on the width of the road and the angle of the wished curve to be acquired with the trapezoid shape (see Figure 4.5). The angle α of the trapezoid sides is derived from the number N of trapezoids constituting a curve of 90 degrees (see Figure 4.6), which equals $\frac{1}{2}\pi$ radians:

$$\alpha = \frac{\frac{1}{2}\pi}{2N} \tag{4.1}$$

As the trapezoid should fit the rest of the road to yield a smooth curve, the diagonal sides should be the same length of the width of the road ρ. The height h of the trapezoid can then be calculated as follows:

$$h = \rho * cos(\alpha) \tag{4.2}$$

The base rectangle of the trapezoid can now be defined as a flat box with height h and width ρ. Two rectangles of the same size are deformed to two opposite parallelograms by moving all the points on the horizontal plane with respect to the origin in the middle of the rectangle. As a deformation value D of 1 in the transformation matrix results in a parallelogram with a side angle of 45 degrees ($\frac{1}{4}\pi$ radians), the deformation value resulting in an angle α is defined by:

$$D = \frac{\alpha}{\frac{1}{4}\pi} \tag{4.3}$$

Subsequently, the parallelograms are translated in order to fit together with the base rectangle and construct as a whole a trapezoid shape. The translation on the x-axis is given by:

$$\Delta_x = D \cdot \frac{\rho}{2} \cdot \frac{h}{\rho} \tag{4.4}$$

Composite construction

Every possible shape of a circuit can be created by combining the trapezoid and rectangle road building blocks. In the construction of the trapezoid shape, the length of the diagonal sides are taken into account to be the same length as the width of the rectangle road. To fit the parts together, the following x- and y-translations should be calculated, for each possible transition: a trapezoid following a rectangular block (BT), a trapezoid following a similar trapezoid (TT), and a right bending trapezoid following a left bending trapezoid (LR). See Figure 4.7 depicting the corresponding geometrical relations.

When a trapezoid shape follows a straight road block:

$$\Delta_x^{BT} = \frac{\rho}{2} + (\frac{\rho}{2} \cdot cos(\alpha)) + (\frac{h}{2} \cdot sin(\alpha)) \tag{4.5}$$

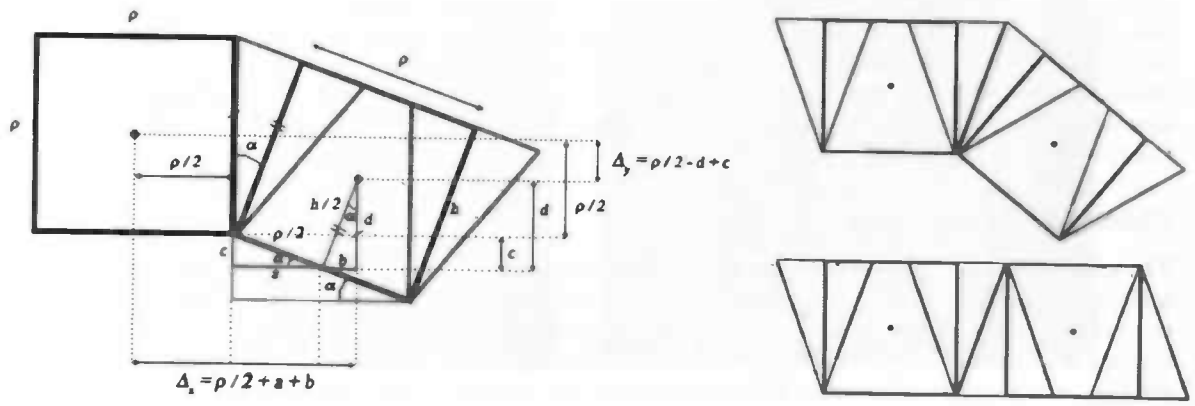


Figure 4.7: Compositional calculations of a trapezoid shape following a square block (BT). At the right: a trapezoid shape following a similar trapezoid shape (TT) and a trapezoid shape following a flipped trapezoid shape (LR).

$$\Delta_y^{BT} = \frac{\rho}{2} - (\frac{h}{2} \cdot cos(\alpha)) + (\frac{\rho}{2} \cdot sin(\alpha)) \tag{4.6}$$

When a trapezoid shape follows a similar trapezoid shape:

$$\Delta_x^{TT} = \frac{\rho}{2} + (\frac{\rho}{2} \cdot cos(2\alpha)) + (\frac{h}{2} \cdot sin(2\alpha)) \tag{4.7}$$

$$\Delta_y^{TT} = \frac{\rho}{2} - (\frac{h}{2} \cdot cos(2\alpha)) + (\frac{\rho}{2} \cdot sin(2\alpha)) \tag{4.8}$$

When a trapezoid shape follows a flipped trapezoid shape:

$$\Delta_x^{LR} = \rho + h \cdot tan(\alpha) \tag{4.9}$$

$$\Delta_y^{LR} = 0 \tag{4.10}$$

4.3.4 XMLCircuitGenerator

To ease the burden of making all calculations for placing the different elements together to create a total circuit, a C++ program is written to automatically generate every possible shape of a circuit using adjustable parameters for the appearance of the road. Different types of circuits can be generated to meet the requirements for a non-trivial environment. The program generates the needed XML circuit definitions to be imported to Vortex. It creates a circuit consisting of rectangular and trapezoid building blocks, which are defined in `circuit_trapezoid.me` and `circuit_ block<`n`>.me`. The composition of N building blocks is defined in the XML files `circuit_cpiece0.me` up to `circuit_cpiece<`N`-2>.me`.

Usage

The generation of any arbitrary circuit is dependent on the width of the road, the angle of the curves (depending on the number of trapezoids making a curve of 90 degrees) and the composition of the elements (the design). The program can be called specifying these parameters:

```
./XMLCircuitGenerator <road width> <# trapezoids making 90 degrees> <design [B()|B|L|R]>
```

Every possible circuit can be made using a design string describing the circuit's composition of every component relative to the former: a straight road block (B for a squared shape of size *road width* × *road width* or B(*length*) for a rectangular shape), a left bend (L) or a right bend (R). For example, a circle is defined by either twelve left bends or twelve right bends, provided that three trapezoids make together 90 degrees. An ellipse shape is defined as the design string `"LLLLLLLLBBBBBLLLLLLLLBBBBB"` when four trapezoids make 90 degrees.

Furthermore, a few pre-defined shapes are available: a circle, an ellipse, an eight shaped circuit[3] and a banana shaped circuit. These designs are made for a configuration with 4 trapezoids making 90 degrees and a road width of 50. When you want to generate a pre-defined circuit, just type in stead of the design string *circle*, *ellipse*, *eight* or *banana*. For example, calling: `./XMLCircuitGenerator 50 4 banana` generates the banana shaped circuit. In the header file the constants `ROAD_THICKNESS` and `BMP_ROAD` are set, defining the height of the flat box and the texture used for a road block.

Compositional XML files

The compositional XML files are responsible for positioning the basic elements at the right place to create a circuit out of them. The generator writes XML files for the definition of the base shape definitions in the composition of the design string. The design is interpreted as the composition of every component relative to the former, from which the compositional XML definitions `circuit_cpiece0.me` up to `circuit_cpiece<`N`-2>.me` are created. Each compositional XML file imports the current building block of the design to position it at the current relative origin of the coordinate axes and imports a next compositional XML file while rotating and translating the origin of that compositional XML file according to the compositional constraints of the next building block in the design. The last compositional XML file imports after placing the current building block the transformed last building block directly instead of another imported compositional XML file. In this way rotation and translation of each building block is always calculated relative to its predecessor and every arbitrary design can be generated without difficult absolute calculations from the starting point.

4.3.5 Circuit Properties

The four pre-defined shapes mentioned above have been generated and imported to the Vortex simulation. The resulting modeled circuits and the car object are shown in Figure 4.8. In generating the circuits the possibility of realization in reality is taken into account. This is, with the chosen sizes of a circuit, the same circuit will fit into the 8 × 8 m robot arena room.

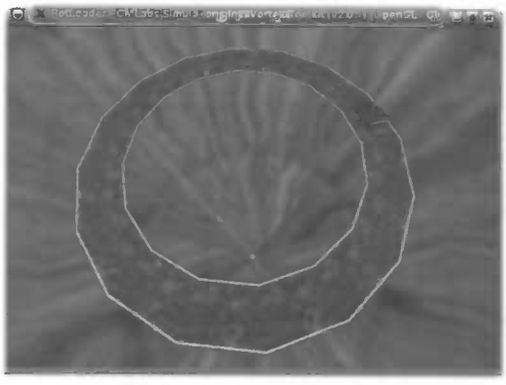
With the automatic circuit generator, any circuit can be made, meeting the experimental prerequisites of the Active Vision and feature selection study with the robotic scale car with omnidirectional camera. The parameters defining the appearance of the environment are adjustable and the pre-defined banana and eight shaped circuits meet the requirements for non-trivial environments, i.e. these circuits contain both right curved bends and left curved bends and in the case of the eight shaped circuit a discontinuity in the guiding sidelines appears in the middle where two tracks cross. See for more information about the modelling and the corresponding XML and C++ code the Vortex Modeling Report [23].

4.3.6 Simplified Circuit

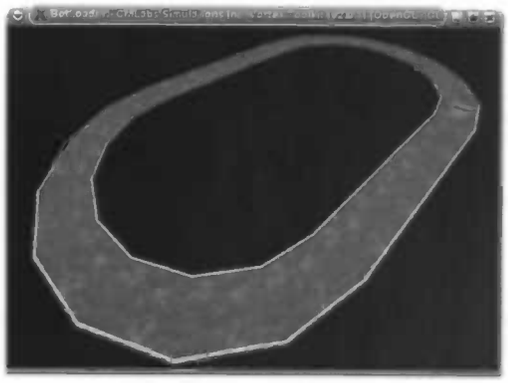
The described way of constructing the circuit out of trapezoid shapes is the neatest solution to create a bending road within the constraints Vortex introduces by the limited amount of possible primitive shapes of objects. However, although Vortex renders the circuit created in this way well (see Figure 4.8), its collision detection and dynamics tools do not function unfortunately. The collision detection mechanism of Vortex cannot deal with the deformation of the rectangular shapes in order to create the parallelogram shapes. The mathematically justified deformation by applying a corresponding transformation matrix to an object does result in proper rendering, but lets Vortex fail in making a collision model, because it checks whether the transformation matrix is orthonormal to ensure the consistency of the object and as a constraint for its collision detection algorithms. The applied deformation is naturally not orthonormal on purpose and is thus not accepted. It results in missing collision models for the parallelogram shapes, which will let the robotic scale car fall on its side as soon as it drives over a bend in the road.

To overcome this problem, a less optimal way of creating the circuit is applied. Instead of using trapezoid building blocks that exactly fit together to make a curve, curves can also be constructed out of only rectangular shapes, rotated with respect to the previous rectangular shape and having a small overlap in one corner (see

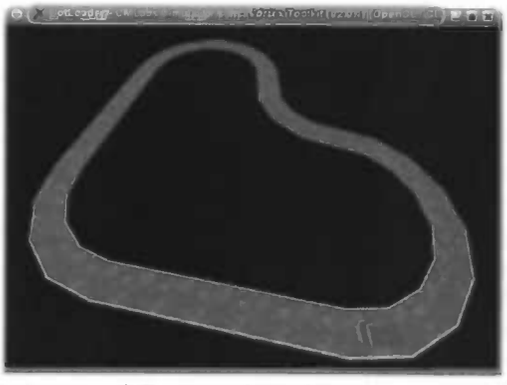
[3]The generator uses the same texture for every road piece. However, when a texture containing two white sidelines is used, this is not applicable for the road piece in the middle of the circuit, where the two tracks cross. As the *eight* design is defined as "`B(50)B(50)B(28.5)RRRRRRRRRRRRB(50)B(50)B(28.2)BB(50)B(50)B(28.5)LLLLLLLLLLLLB(50)B(50)B(28.2)`", where the sole squared `B` corresponds with this crossing road piece, it is simple to change the used texture for that part in the corresponding XML file manually. The definition of the squared `B` building block is always defined in `circuit_block0.me`



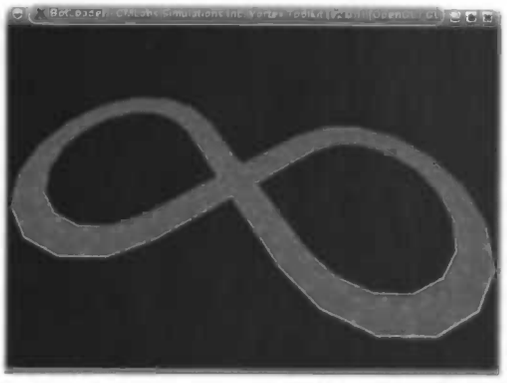
a) Circle shaped circuit



b) Ellipse shaped circuit



c) Banana shaped circuit



d) Eight shaped circuit

e) Eight shaped circuit (close-up)

f) Eight shaped circuit (close-up car)

Figure 4.8: Resulting modeled circuits and robotic scale car in Vortex based simulation.

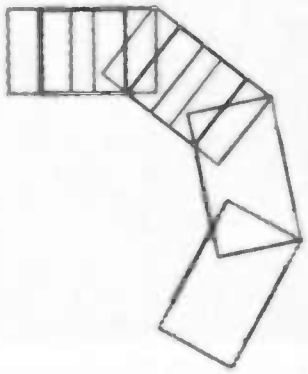


Figure 4.9: A bending circuit constructed out of partly overlapping rectangular shapes instead of trapezoid shapes.

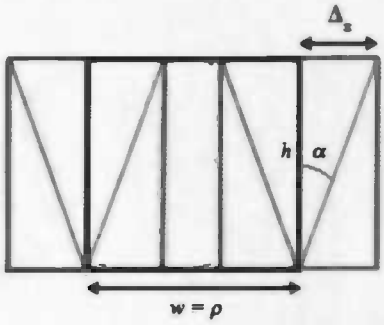


Figure 4.10: The rectangular curve component consists of three rectangular shapes: a base block with two other rectangular blocks translated over the x-axis.

Figure 4.9). In this approach no deformation is needed, and by placing the rectangles at the same height in the world the overlaps are not generating any effect on the flatness of the surface of the road. Vortex has no problems modeling objects that have overlap in space. The drawback of this approach is that it is not compatible with other road textures than uniform textures, e.g. with two white sidelines, because the transition is interrupted. By using deformation to bend the road, the texture would also deform in the right manner, assuring a smooth transition of sidelines from one building block to another. Furthermore, in rendering two objects that have their surface at the same place, Vortex cannot choose which object is the upper one and renders by chance one or the other. This results in a slight flickering of the overlapping pieces in the road. The flickering is of no concern when the textures of the pieces are uniform and similar, but will be obvious when they differ, which is the case in using textures with sidelines. To maintain the clear contrast between the road and the environment, instead of by using sidelines, the brightness of the road color is made sufficiently discernable from the brightness of the environment.

The automatic circuit generator is modified for the simplified approach. The **XMLSimpleCircuitGenerator** still makes the same geometrical calculations for the trapezoid shapes, but the trapezoid shapes are rectangular instead (see Figure 4.10). Such a rectangular curve component is similarly constructed out of three rectangular shapes, the same base rectangle and two other rectangles that are translated respectively to the left and the right of the base rectangle. The translation on the x-axis is given by:

$$\Delta_x = h \cdot tan(\alpha) \tag{4.11}$$

which equals the size of one of the triangular parts of the curve's outer side of the original trapezoid shape.

4.4 Robot Simulator

In order to run the evolutionary experiments in simulation, a robot simulator has been made, based on the already described Vortex Simulation Toolkit, which is responsible for the realistic physical dynamics in the simulated world. The Robot Simulator is initially developed by Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA, *http://www.idsia.ch*) to provide tools to facilitate the creation of a robot simulation in the context of the Swarm-bots project in the European Commission Future and Emerging Technologies program (*http://www.swarm-bots.org*) [5]. After that the simulator is modified multiple times to meet the actual needs in the Active Vision and Feature Selection project of the Laboratory of Intelligent Systems (EPFL, *http://lis.epfl.ch*) [19, 21].

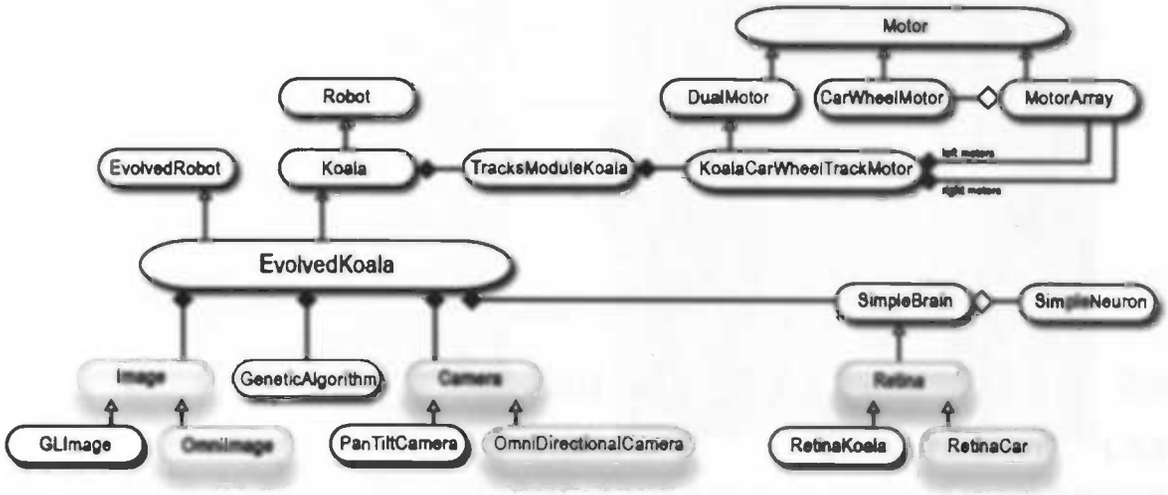


Figure 4.11: UML Diagram of the classes in the Robot Simulator and the relationships amongst them. The class hierarchy is depicted by arrowed inheritance links, pointing towards the superclass. Aggregation, the association in which one class belongs to a collection, is depicted by a link ending in an open diamond at the part containing the whole. Composition, the strong association in which the part can belong to only one whole and cannot exist without the whole, is depicted by a link ending in a filled diamond at the whole end. The highlighted classes have been added in the context of this study to enable the use of an omnidirectional camera.

4.4.1 Structure

The Robot Simulator consists firstly of a module responsible for the communication with the Vortex layer. It transfers the location of the XML files defining the simulated world, to the Vortex XML parser to let Vortex create the defined models and set the desired parameters. It creates links from the C++ objects in the Robot Simulator to the corresponding physical Vortex objects constituting the robot in simulation. All control and evolution processes interact via this module with the robot objects in the Vortex world to get informed about their properties, to modify and to update them.

Every Vortex time step the `action()` method, and every Active Vision time step the `evolution()` method of the robot object in the Robot Simulator are called. These methods reside in the module that controls the evolutionary robot. This module is the central module in initiating and controlling the robot's sensors and actuators, neural network and evolutionary process. At every `action()` step (currently set to 100 ms) the motor commands are given to the robot, and at every `evolution()` step (currently set to 30 ms) a sensory-motor loop is made that consists of updating the input neurons with visual information, computing the output values of the neural network to determine the new motor commands, and supervising the evolutionary process. The actual neural network and genetic algorithm classes can be found in the Active Vision module. The sensors and actuators form the devices module. Another module defines the modular objects a robot can consist of, i.e. robot configurations with predefined devices, like a wheelbase specifying a fixed number of motorized wheels. All these objects are eventually activated by the central robot module mentioned before, controlling the robot at the highest level.

The class structure is shown in Figure 4.11. The simulation starts with creating a new instance of the evolutionary robot (EvolvedKoala[4]), which gets its evolutionary functionality inherited from the EvolvedRobot class, and its robotic functionality, including the motorized wheels, from the Koala class. Furthermore, it consists of individual instances of its neural network, genetic algorithm, and camera characteristics, including the acquired camera image properties. In the context of this study, six classes are added to enable the use of the omnidirectional camera, see the next sections.

[4]In adapting the existing robot simulator to the needs for the experiments with the robotic scale car, some old class names reminiscent of the former work have remained unchanged. In reading the robot name Koala, the actual class or function refers to the robotic car instead.

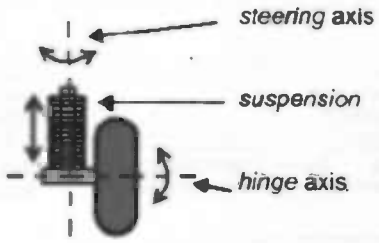


Figure 4.12: The used car wheel model, with steering and suspension.

4.5 Modifications for the Car Robot

4.5.1 Motor Modeling

While in the existing simulator steering of the robot was achieved by different speeds of the left and right wheel of a two-wheel driven robot, the four-wheeled robotic car steers by rotating its two front wheels around their steering axes. Vortex provides a CarWheel joint modeling the behavior of a real car wheel, with suspension, a steering column for steering, and a driving axle for the rotation of the wheel (see Figure 4.12). The motor section of the robot simulator has been modified to enable the front wheels to rotate around their steering axes. The motor call from the evolutionary robot involves now both a speed assignment and a steering angle assignment for the two front wheels. Furthermore, since the robotic scale car is four-wheel driven, all four wheels will receive the same speed command.

4.5.2 Camera Modeling

In the Vortex Viewer the point of view of the simulated world can be changed by setting the look-at point of the camera to a specified position in the world. In modeling a pan-tilt camera, the point of view in the renderer is just set to the desired position, pan and tilt values, corresponding with the robot's point of view. In order to determine the current robot's point of view, Vortex is called for the physical position of the robot's camera in the simulated world. In the XML definition of the robot is taken into account that one specific object in the composite robot model represents the camera's mounting point (on top of the tube a `CAMERA_OBJ_MODEL` object is defined) to enable referring to this specific object in the Vortex call for its position in the world.

In modeling the omnidirectional camera however, a standard camera view cannot be used. Figure 4.13 depicts a resulting image a real omnidirectional camera typically grabs by using a hyperbolically curved mirror to see all around. Firstly, the image contains more information than a standard camera view (i.e. in all directions), and secondly, the view is circularly distorted by the mirror. These two properties of an omnidirectional image have been modeled in the camera section of the robot simulator to enable the use of an omnidirectional camera. A resulting omnidirectional image in simulation is depicted in Figure 4.14.

The result is achieved by mapping the information of five images (all directions, see Figure 4.15) on a fish-eye coordinate system (circular distortion). The current physical position of the robot's camera is requested, and the look-at point of the Vortex Viewer is set five times from this position to grab four images in all wind directions together with one top view, looking down from the top of the tube to the roof of the robotic car. Subsequently, the indices for the fish-eye transformation are determined. The coordinates in all five images get their corresponding coordinates in the omnidirectional image assigned, by mapping their Cartesian coordinates to corresponding circular polar coordinates (see Figure 4.16) in the omnidirectional image.

The process of grabbing a camera image of the simulated omnidirectional camera takes significantly more time than for the pan-tilt camera, because it has to compose the image out of five different images in a computational time-consuming way. As a result the simulation's speed is affected, and sensory-motor cycles of 30 ms cannot be guaranteed anymore. A solution to speed up the simulation without affecting the sensory-motor cycle time is to enlarge the Vortex time step. With a Vortex time step of 200 ms instead of 100 ms, the physical integrator is a bit less accurate, but the simulation runs at double speed.

Figure 4.13: An omnidirectional camera image grabbed from the real omnidirectional camera (the car situated on an office desk).

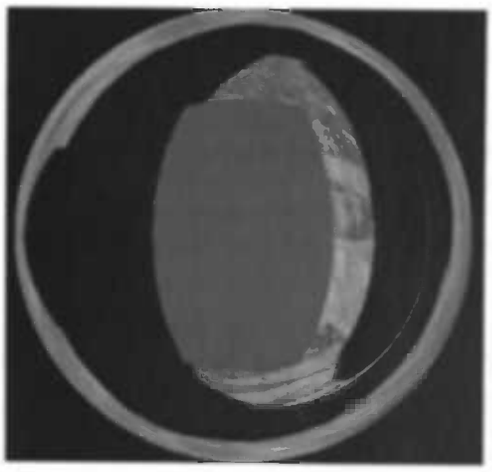


Figure 4.14: An omnidirectional camera image grabbed from the simulated omnidirectional camera (the car situated on a circuit).

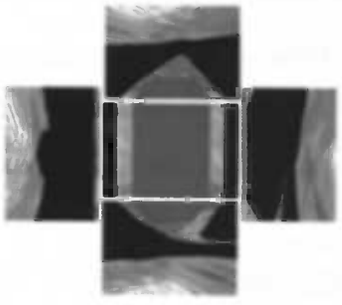


Figure 4.15: The five images an omnidirectional image is composed of: a top view image and four images in all wind directions).

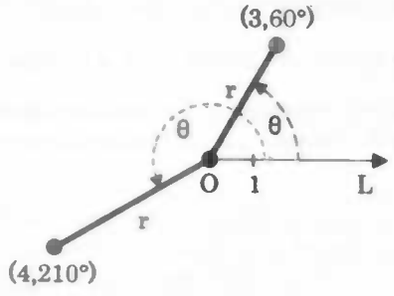


Figure 4.16: Circular polar coordinates define a point in space instead of in the corresponding x- and y-values, in the radius r (the distance from the origin to the point) and the azimuth θ (the angle between the positive x-axis and the line from the origin to the point).

Every pixel in the original five images is treated as follows:

- The x- and y-coordinates (x_{ori}, y_{ori}) in the original image are reset to belong to a Cartesian (x, y) coordinate system with its origin in the center of the original image, instead of its origin in the upper left corner of the image:

$$x = x_{ori} - \frac{width}{2} \tag{4.12}$$

$$y = y_{ori} - \frac{height}{2} \tag{4.13}$$

  where *width* and *height* are respectively the width and height of the original image.

- The polar coordinates (r, θ) corresponding to the Cartesian coordinates (x, y) in the original image are calculated:

$$r_{ori} = \sqrt{x^2 + y^2} \tag{4.14}$$

$$\theta = \arctan 2(\frac{y}{x}) + (y < 0 \ ? \ 2\pi \ : \ 0) \tag{4.15}$$

If the resulting radius is larger than the maximum radius r_{max} in the original image constrained by its height, the pixel is omitted.

$$r_{max} = \frac{height}{2} \tag{4.16}$$

- The minimum and maximum viewing tilt angles of the hyperbolically curved mirror are calculated:

$$\alpha_{min} = \frac{1}{2}\pi - \frac{1}{2}FOV \tag{4.17}$$

$$\alpha_{max} = \frac{1}{2}\pi + \frac{1}{2}FOV \tag{4.18}$$

where FOV equals $\frac{1}{2}\pi$ radials, the total field of view of a camera shot.

- The actual viewing tilt angle α, corresponding to this pixel, is calculated:

$$\alpha = 2\arcsin\left(\sqrt{\frac{1}{1.4 \cdot r_{max}^2 \cdot r_{ori}^2}}\right) \tag{4.19}$$

where $1.4 \cdot r_{max}^2$ is the estimated squared radius of the circular mapping in the resulting omnidirectional image (the full range of the mirror).

Subsequently, only for the top view image:

- If the actual viewing tilt angle α is larger than $1.4 \cdot \alpha_{min}$, the pixel is out of the mirror range and omitted.
- The pixel is mapped to the resulting omnidirectional image, back to Cartesian coordinates (x_{omni}, y_{omni}), using the radius r_{omni} of the omnidirectional target image:

$$r_{omni} = \frac{height}{2} \cdot \tan(\alpha) \tag{4.20}$$

$$x_{omni} = \frac{width}{2} + r_{omni}\cos(\theta) \tag{4.21}$$

$$y_{omni} = \frac{height}{2} + r_{omni}\sin(\theta) \tag{4.22}$$

- If the resulting x- and y- coordinates exceed the width or height of the image, the pixel is omitted.

Finally, for all four side view images:

- If the actual viewing tilt angle α is smaller than α_{min} or larger than α_{max}, the pixel is out of the mirror range and omitted.
- The pixel's polar coordinates from the original image are mapped to the corresponding polar coordinates $(r_{omni}, \theta_{omni})$ in the omnidirectional target image, and transformed back to the corresponding Cartesian coordinates (x_{omni}, y_{omni}) of the omnidirectional target image:

$$\theta_{omni} = \frac{1}{2}FOV - ((\theta + \frac{1}{2}FOV) \bmod FOV) \tag{4.23}$$

$$\alpha_{omni} = \alpha - \frac{\alpha_{min} + \alpha_{max}}{2} \tag{4.24}$$

$$x_{omni} = \frac{width}{2} + \frac{1}{2} \cdot height \cdot \tan(\theta_{omni}) - 1 \tag{4.25}$$

$$y_{omni} = \frac{height}{2} + \frac{\frac{1}{2} \cdot height}{\cos(\theta_{omni})} \cdot \tan(\alpha_{omni}) - 1 \tag{4.26}$$

- If the resulting x- and y- coordinates exceed the width or height of the image, the pixel is omitted.

In order to let the robot simulator work with the omnidirectional camera, the camera initiation by the evolutionary robot is modified and classes are added for the implementation of the omnidirectional camera and for the properties of its generated omnidirectional image (see Figure 4.11). To maintain the possibility to switch easily between the omnidirectional and the pan-tilt camera, in both cases a virtual superclass is created that ports the general camera and image calls from the evolutionary robot either to the one or to the other subclass, dependent on the used type of camera. To clear the view of the omnidirectional camera in all directions, it was needed to switch off rendering of the camera tube in the XML robot model definitions, because this object interfered with the top view. In the real omnidirectional camera the tube holding the hyperbolically curved mirror is also transparent.

In addition to the OpenGL window of the Vortex Viewer, which renders the simulated world from the distal perspective of the Vortex camera settings, a second window showing the omnidirectional camera view has been created. In using the omnidirectional camera image as input for the neural network of the robot, the visual information from this second window is used and any camera position can be chosen in the Vortex Viewer window to display the car driving on the circuit from a spectator's perspective. In the case of the pan-tilt camera, the information is grabbed directly from the Vortex Viewer camera view, and restricts therefore the visualization of the robot behavior to the perspective of the robot camera itself (the pan-tilt camera's `seeThroughThis` function should be called). In the case of the omnidirectional camera it is, in addition to the possible static distal view on the robot's starting position, made possible to call the omnidirectional camera's `seeThroughTrack` function, which moves the camera looking at the driving car respectively to the moving position of the robotic car. This results in a helicopter-like tracking camera behavior, which follows the car everywhere over the circuit from above.

4.6 Evolutionary Active Vision Set-up

4.6.1 Neural Network Architecture

To equip the car with Active Vision and analyze the developed strategies in learning how to drive a circuit without going off-road, a deliberately simple discrete-time feed-forward network with recurrent connections at the output layer and evolvable thresholds and connection weights is used (see Figure 4.17).

The input layer consists of a set of 25 visual neurons (the retina), receiving the grey-level information of the pixels corresponding to their non-overlapping receptive fields arranged on a 5×5 grid in the image, and two or possibly three proprioceptive neurons, giving feedback about the current state of the vision system. In the case of the use of the pan-tilt camera, two proprioceptive neurons encode the actual pan and tilt angles of the camera. In the case of the omnidirectional camera, these two neurons encode the absolute vertical and horizontal position of the retina in the visual scene, and when zooming of the retina is activated (to dynamically change the resolution of the receptive fields), a third neuron encodes the current zooming factor. In the case of the pan-tilt camera the resolution of the receptive fields is fixed, and the retina is permanently positioned in the middle of the camera image (as the pan-tilt camera is able to move itself).

The output layer consists of two neurons generating the motor behavior and three or possibly four neurons generating the retina behavior. Two motor output neurons control the steering angle of the two front wheels and the forward/backward rotational speeds of all four wheels. Two vision behavior output units regulate either the pan and tilt displacements of the camera in the case of a pan-tilt camera, or the horizontal and vertical displacements of the retina in the case of an omnidirectional camera; in both cases it represents the displacement relative to the current absolute position of the camera or the retina. Furthermore, another vision behavior neuron sets the used filtering method of the pixels in the receptive fields (averaging: processing the average grey-value of all corresponding pixels within each of the 25 receptive fields, or sampling: taking one sample pixel in the middle of each receptive field as fully representing each receptive field). In the case of zooming, a fourth vision behavior output neuron determines the change in resolution of the receptive fields relative to the current absolute zoom factor. The used activation function is the sigmoid function, or in the case of an architecture with zooming neurons the equivalent $\tanh(\frac{1}{2}x)$ function.

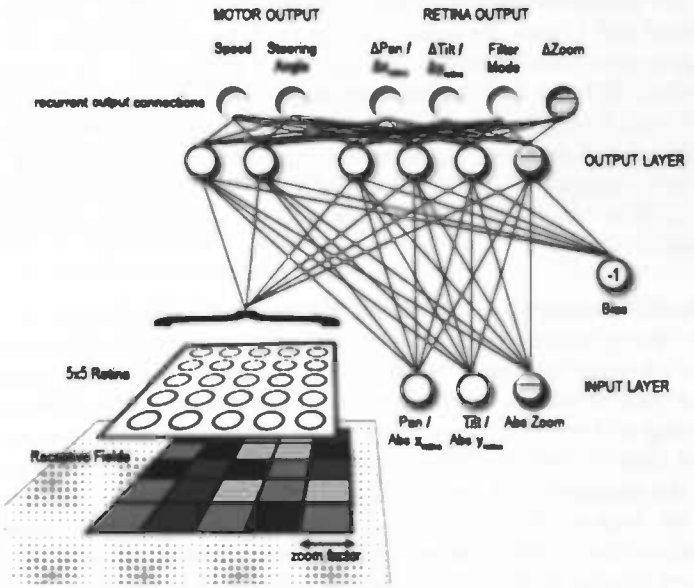


Figure 4.17: The neural architecture for the Active Vision system. The highlighted neurons are only used in an Active Vision set-up with a retina capable of zooming.

4.6.2 Genetic Algorithm

The synaptic development in the described neural network is done by an evolutionary process. The thresholds and connection weights are encoded in a binary genotype string to be evolved using a genetic algorithm. A population of 100 individuals is evolved for 20 generations to drive the car on a circuit with high speed as straight as possible. Every individual is evaluated three times for 500 sensory-motor loops (every sensory-motor loop takes 0.3 seconds). When an individual drives the car off-road (detected by the change of height of the car, since the car is falling), the trial is truncated and consequently gets assigned less fitness, as it cannot drive further.

The fitness function rewards high forward speed and punishes for superfluous steering:

$$Fitness = \frac{1}{E \cdot T \cdot s_{max}} \sum_{e=0}^{E} \sum_{t=0}^{T'} s_t \cdot \left(1 - \sqrt{\frac{|d_t|}{d_{max}}}\right) \tag{4.27}$$

where E is the number of trials per individual (3), T the maximal number of time steps for each trial (500), T' the effective number of time steps the individual survived, s_t the speed of the wheels during the current time step, s_{max} the maximum speed possible, d_t the steering direction angle during the current time step, and d_{max} the maximum steering angle (45 degrees in the current simulations).

The fitness value is integrated over every time step in all trials and normalized. The square root over the steering punishment part yields a little punishment for small steering movements close to a straight course, but big punishment for steeper steering movements. Note that the fitness of an individual can never reach 1.0, since it always has to steer in the corners to keep driving on the circuit. Truncation selection is used as the selection method, i.e. the best 20% individuals are selected for reproduction, crossed over with a probability of 0.1 and mutated with a probability of 0.01. A copy of the two best genotypes of the previous generation is inserted in the new population (elitism).

In order to let the evolved behavior be as general as possible, the starting position of the car should not be the same for every individual. Evolution could then develop a blind solution that is based on and will always work from this specific starting position. To force evolution to develop a robust solution, based on the visual input it selects, every individual starts differently positioned on the circuit, facing the circuit clockwise

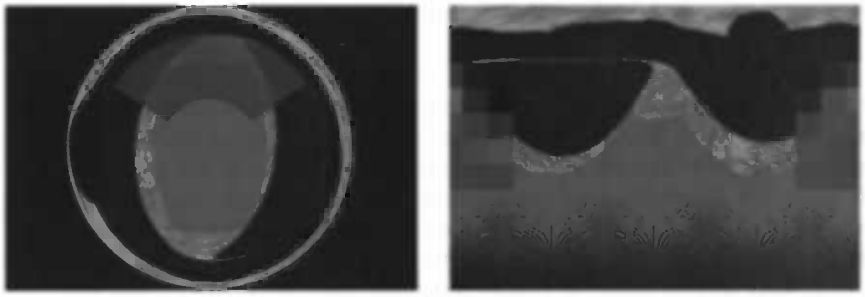


Figure 4.18: The retina defined in polar coordianates in the omnidirectional image (left). For computational simplification the activation from the receptive fields is computed from the corresponding polar map (right), visualizing the azimuth θ on the horizontal axis and the radius r on the vertical axis.

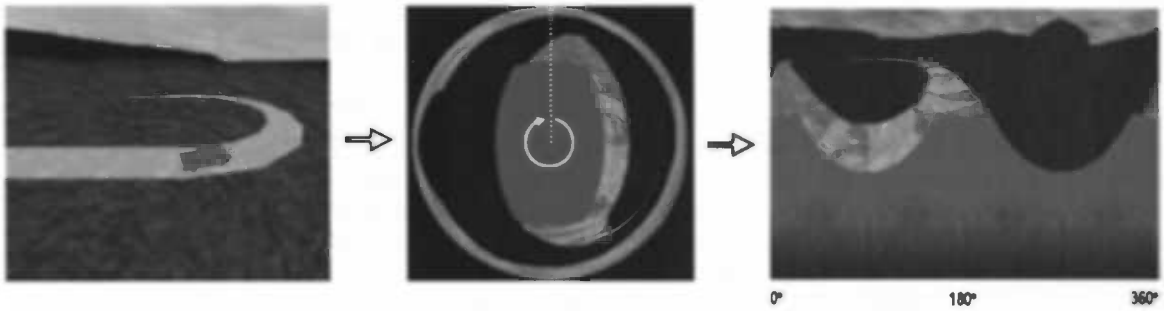


Figure 4.19: The polar map (totally right) represents the stretched panoramic view grabbed by the omnidirectional camera. The azimuth θ is plotted on the horizontal axis and the radius r on the vertical axis.

or counterclockwise by chance, with a slight random deviation from the middle of the road, and a rotational deviation of maximally 30 degrees.

4.6.3 Retina Movement

In the case of the pan-tilt camera the retina is positioned fixed in the middle of the camera image and the visual scene is actively accessed by moving the camera itself. However, in the case of the omnidirectional camera the image provided by the camera already represents the total visual scene, and selective access must be made by a retina moving through the image. Since the omnidirectional image is circularly distorted, the information about neighboring pixels is correspondingly distributed circularly over the image. To match the information distortion, the retina's grid of receptive fields has been defined in circular polar coordinates (r, θ) in the case of the omnidirectional camera (see Figure 4.18).

To ease the computation of the retina in polar coordinates, the omnidirectional image is first stretched to a panoramic view on a coordinate system plotting the azimuth θ on the x-axis, and the radius r on the y-axis. In the resulting polar map (see Figure 4.18, 4.19) a square grid of receptive fields can move through the polar space by displacement over the two axes. Since the angular azimuth displacement lacks boundaries, exceeding the boundary of the polar map image on the x-axis should yield appearance of the retina at the other boundary. The radius has its limits however, and correspondingly a retina that tries to move out of the image on the y-axis is kept at its maximal position. The polar map can therefore be thought of as cylindrical.

4.7 Analytical Tools

In order to be able to analyze and understand the evolved behaviors of successful individuals, several analytical tools are provided by the simulator. Firstly, a fitness graph depicting the development in average and best fitness over generations can be extracted from a log file. Furthermore, every generation the genotypes of the individuals selected by truncation selection are saved to enable loading an individual with a specific genotype

produced in the evolutionary process. This makes it possible to visually evaluate the behavior of e.g. the best-evolved individual of all generations.

For visualizing such an evaluative run, three separate analysis windows are added in the context of this study. One contains the distal view (possibly with helicopter-like tracking of the driving robot), another shows the omnidirectional camera view (possibly with the sizes and position of the retina depicted in the image), and a last window visualizes the polar map (possibly with depiction of the retina, including the grey-values of its corresponding receptive fields). During an evaluative run of a specific individual, the activation values of all neurons in the neural network are saved in a neural activation file. Furthermore, the saving of the position of the robot at every time step is added to enable reconstruction of the car's trajectory.

Chapter 5

Experiments and Results

5.1 Pan-Tilt Camera Experiment

5.1.1 Experimental Set-up

In this experiment a controller for the robotic scale car equipped with the standard *pan-tilt camera* is evolved in simulation, in order to compare these results with those obtained by a car equipped with an omnidirectional camera in the same situation. The individuals are evaluated to drive an *ellipse shaped circuit* as straight as possible with high speed. The ellipse shape is a simple circuit with curves as well as straight parts, and is therefore appropriate for comparative analysis of the both camera types. The maximum rotational speed of the car's wheels is set to 2.7 radians per second, the maximum steering angle to 45 degrees.

The previously described neural network architecture is used, receiving input from a 5×5 retina, together with two proprioceptive neurons encoding the absolute pan and tilt degrees. The retina has a fixed zooming factor resulting in a total size of 240×240 pixels, and is permanently located in the middle of the camera image, which has a total size of 640×448 pixels. The output layer consists of two motor neurons controlling the speed of the four wheels and the steering angle of the two front wheels, two neurons controlling the pan and tilt displacement of the camera, and a neuron setting the filter mode. The output layer has fully recurrent connections and is connected to the bias. All neurons in the output layer use the sigmoid activation function.

A population of 100 individuals is evolved over 20 generations and each is evaluated during three trials, lasting maximally 500 sensory-motor cycles. The fitness function rewards high speed and punishes for steering. A trial is immediately truncated when an individual either falls down from the circuit (negative change of position on the z-axis), or bounces too high due to its suspension (a significantly high positive change of position on the z-axis). The latter measure was needed, because evolution cheated in exploiting the fitness assignment assumption that the wheels are always on the circuit as long as the car is not falling down. This resulted in acrobatic individuals jumping to fall on their back, in order to let their wheels spin at high-speed in the air. Individuals are thus prevented from bouncing too high. Truncation selection is applied to select the 20 best individuals for reproduction in the next generation, together with elitism insertion of the two best individuals. Every individual starts from a fixed position on the circuit (at the start of a curve), but every trial with a random orientation and deviation from the middle of the road. Since the individuals can get positioned clockwise as well as counterclockwise on the circuit, both types of curves (left and right bending) have to be learned by the individuals.

5.1.2 Evolved Behavior

Best-evolved individuals reach a fitness around 0.6 (see Figure 5.1) and are able to drive the circuit with full speed (i.e. 2.7 radians per second, which equals 8.9 cm/s) without driving off-road. The evolved visual behavior consists of turning the camera immediately downwards and always to the right (see Figure 5.4), independently of the car's position on the circuit (resulting in looking towards the inner side of the circuit in some trials as well as to the outer side in other trials). The filtering of the receptive fields is always done by the averaging method.

The resulting strategy consists of keeping track of the circuit's edge with the lowest layer of the 5×5 retina and using this horizontal feature to align (see Figure 5.3). As soon as the second to lowest layer starts perceiving a part of the circuit too, the car reacts with a sudden steering of its wheels to align to the circuit's sideline again (see Figure 5.5). The trajectory of the car on the circuit is shown in Figure 5.2.

5.2 Omnidirectional Camera Experiments

5.2.1 Ellipse Shaped Circuit

Experimental Set-up

In this experiment a controller for the robotic scale car equipped with the *omnidirectional camera* is evolved in simulation to drive teh same *ellipse shaped circuit* as straight as possible with high speed. The maximum rotational speed of the wheels is set to 2.7 radians per second, the maximum steering angle to 45 degrees.

The previously described neural network architecture is used, receiving input from a 5×5 retina, together with two proprioceptive neurons encoding the absolute vertical and horizontal position of the retina in the visual scene. The retina has a fixed zooming factor (zooming is not activated) resulting in a total size of 100×100 pixels, and can freely move within the whole omnidirectional polar map with a maximal displacement of 20 pixels per time step. The polar map has a total size of 320×224 pixels. The output layer consists of two motor neurons controlling the speed and the steering angle of the wheels, two neurons controlling the horizontal and vertical displacements of the retina, and a neuron setting the filter mode. The output layer has fully recurrent connections and is connected to the bias. All neurons in the output layer use the sigmoid activation function.

A population of 100 individuals is evolved over 20 generations and each is evaluated during three trials, lasting maximally 500 sensory-motor cycles. The fitness function rewards high speed and punishes for steering. A trial is immediately truncated when an individual either falls down from the circuit, or bounces too high due to its suspension. Truncation selection is applied to select the 20 best individuals for reproduction in the next generation, together with elitism insertion of the two best individuals. Every individual starts from a fixed position on the circuit (in the middle of a straight part), every trial with a random orientation and deviation from the middle of the road.

Evolved Behavior

Best-evolved individuals reach a fitness around 0.45 (see Figure 5.6) and are able to drive the circuit without driving off-road, provided a good start, which happens about half of the times. Once on the way the behavior is very robust in staying on the road. The car drives at full-speed (8.9 cm/s), with tiny slow-downs while steering in the curves (see Figure 5.10). The evolved visual behavior consists of looking to the road in front of the car, in order to anticipate on an approaching turn. The retina stays at zero degrees in the omnidirectional image (looking forward) and increases the radius to see the road (see Figure 5.9). The filtering of the receptive fields is always done by the averaging method.

The resulting strategy consists of looking ahead to the approaching piece of circuit and correlating the discernible difference of an approaching curve with the appropriate motor commands. It is able to discriminate a curve by the emerging brightness of a piece of road at the right or left half of the retina, and the simultaneously diminishing road perception at the other half of the retina (since the road bends away to the other side). In order to enhance this feature, it drives always close to one side of the road (preferably the left), so one half of the retina cannot see the road that much as the other half (see Figure 5.8). Since it looks ahead, it can see a turn coming in time and is thus able to correct its heading direction gradually, without sudden steering movements. The trajectory of the car on the circuit is therefore quite smooth, and follows the fastest track in the circuit by taking the turns on the inside (see Figure 5.7).

5.2.2 Banana Shaped Circuit

Experimental Set-up

In this experiment a controller for the robotic scale car equipped with the *omnidirectional camera* is evolved in simulation to drive a *banana shaped circuit* as straight as possible with high speed. The banana circuit has

a characteristic that both type of curves appear during a single trial and prevents thus individuals to learn to steer always in the same direction during a trial. The maximum rotational speed of the wheels is set to 2.7 radians per second, the maximum steering angle to 45 degrees.

The same neural architecture is used, providing a 5×5 retina with a fixed size of 100×100 pixels exploring the polar map of the omnidirectional image. It has visual proprioception, speed and steering motor output and visual output for retina displacement and filter mode. The output layer has fully recurrent connections and is connected to the bias. All neurons in the output layer use the sigmoid activation function.

A population of 100 individuals is evolved over 20 generations and each is evaluated during three trials, lasting maximally 500 sensory-motor cycles. The same fitness function is used. A trial is immediately truncated when an individual either falls down from the circuit, or bounces too high due to its suspension. Truncation selection is applied to select the 20 best individuals for reproduction in the next generation, together with elitism insertion of the two best individuals. Every individual starts from a fixed position on the circuit (in the middle of a straight part), every trial with a random orientation and deviation from the middle of the road.

Evolved Behavior

Best-evolved individuals reach a fitness around 0.3 (see Figure 5.11), which is rather low. However, best-evolved individuals are also in this case perfectly able to drive the circuit without driving off-road when they survive the starting period. Many times an 'in principle' successful individual drives off the road in the starting phase, because it is still looking for the location of the useful features. The low fitness seems thus not due to a bad general performance, but to the unfortunate random starting position and orientation of the car on the circuit and of its retina in the polar map. Once on the way, the behavior of these individuals is very robust in staying on the road. The car drives at full-speed (8.9 cm/s) all the time and wiggles a lot with its wheels (see Figure 5.15). The evolved retina movement consists of looking at the road in front of the car, in order to anticipate an approaching turn. The retina also wiggles a lot around an azimuth of zero degrees (looking forward) and the radius position that represents the location of the upcoming road (see Figure 5.14). The filtering of the receptive fields is in this case always done by the sampling method.

The resulting strategy consists again of looking ahead to the approaching piece of circuit, but the retina does not fixate at a maximal radius. It is focusing on the exact location in the polar map where the upcoming road is located and wiggles constantly around this point. At the straight parts of the circuit it wiggles more towards a higher radius (looking more ahead) and wiggles simultaneously slightly on the azimuth axis (scanning for either right or left bends). In a curve the retina wiggles more towards a lower radius (looking near the car) and keeps its azimuth at zero degrees (looking forward). The retina movements in the direction of the radius are correlated with the steering commands, which results in a likewise constant wiggling of the wheels (see Figures 5.13, 5.14 and 5.15). The car thus constantly perceives its environment actively, prepared for a possibly appearing curve, and corrects its heading correspondingly. The sampling method emphasizes the contrasts and yields quite abrubt big corrections (even on straight parts), resulting in a less smooth trajectory (see Figure 5.12).

5.2.3 The Ellipse-Evolved Individual On The Banana Circuit

To examine to what extent the banana shaped circuit environment affects the evolved behavior (with its curves of both types), the best-evolved individual from the evolutionary run on the ellipse shaped circuit is evaluated on the banana shaped circuit to compare the performance and used strategies. The ellipse-evolved individual also performs well on the banana shaped circuit and drives at high-speed without going off-road. After a long scanning movement of the retina in the visual scene, it finds its permanent location looking forward at a maximum radius (see Figures 5.18, 5.17). It constantly corrects its heading, mainly in the curves (see Figure 5.19) and follows a smooth trajectory (see Figure 5.16). The filtering of the receptive fields is always done by the averaging method.

A noticeable fact is the survival strategy at the disorientated starting period, when the retina is searching its desired location. The car does not drive randomly around during this phase, but makes a short backward movement and drives very slowly until the appropriate heading direction is found. Subsequently it speeds up performing full-speed navigation on the banana circuit.

The ellipse-evolved individual and the banana-evolved individual use both different strategies. The ellipse-evolved individual drives much more sensitive, while the banana-evolved individual is actively scanning for possible curves bending to both sides of the road. The banana-evolved individual seems more prepared for sudden curves to either side of the road, but wiggles very wildly while doing that. Obvious is that the ellipse-evolved individual can survive on the banana shaped circuit and can switch steering direction within one trial too. It should be noted that the difference in the developed strategies could also simply be the stochastically different outcome over different evolutionary runs, unrelated to the type of circuit a strategy has been evolved on. To examine this, more evolutionary runs are needed.

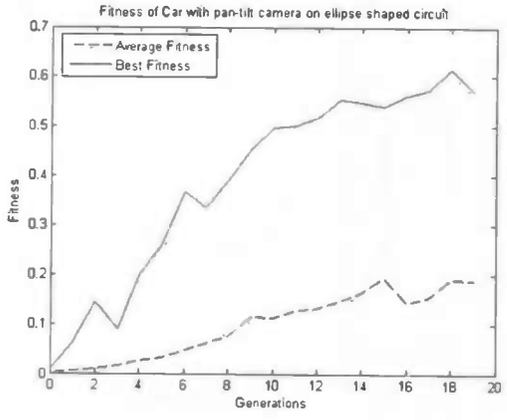


Figure 5.1: The fitness graph depicting the development of the population average and the best individuals over 20 generations of evolved robotic cars equipped with the pan-tilt camera, driving the ellipse shaped circuit.

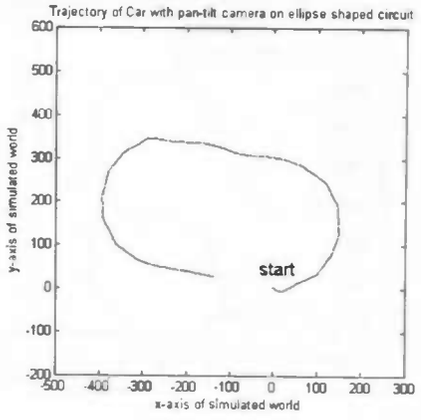


Figure 5.2: The trajectory of the best-evolved car equipped with the pan-tilt camera on the ellipse shaped circuit.

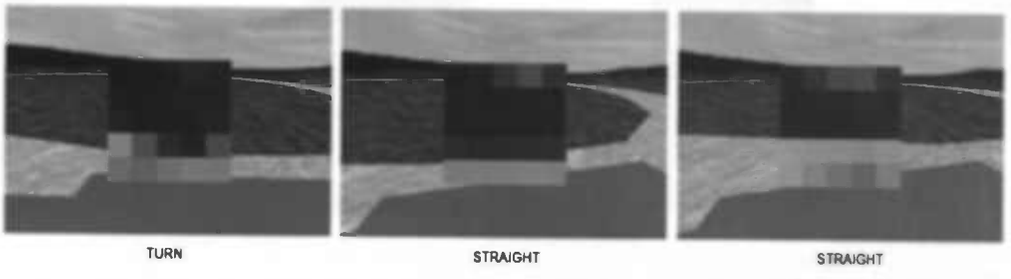


Figure 5.3: The evolved strategy in a curve of the circuit. The car is driving to the left in the image, with its camera looking to its right side. The car aligns with the lowest layer of its 5×5 retina and correlates the perception of the circuit in its second to lowest layer with a correcting steering action.

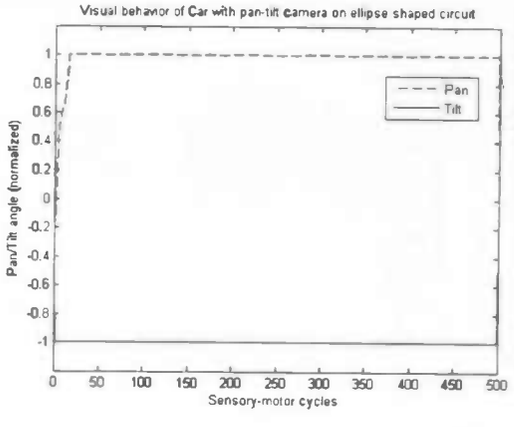


Figure 5.4: The pan and tilt behavior of a successful evolved individual equipped with the pan-tilt camera, driving the ellipse shaped circuit. The camera is immediately set to look down and to the right.

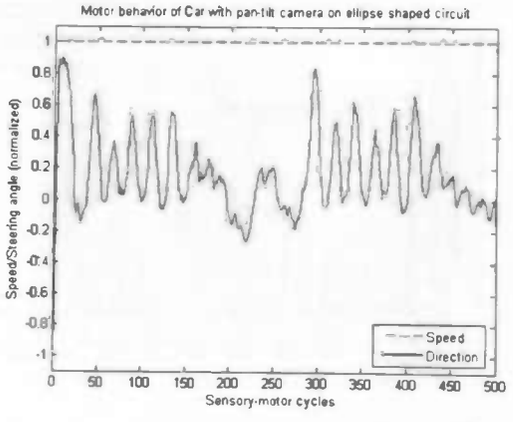


Figure 5.5: The speed and steering behavior of a successful evolved individual equipped with the pan-tilt camera, driving the ellipse shaped circuit. The car is driving at full-speed and aligning to the sideline of the road with sudden steering corrections.

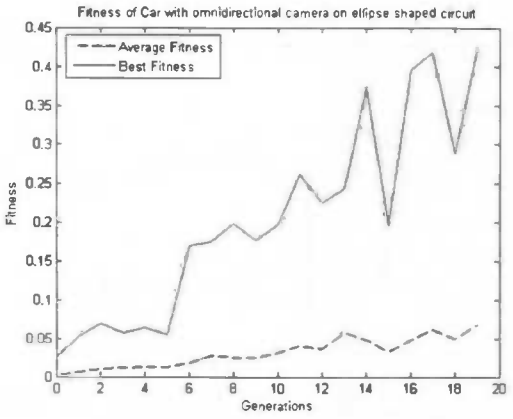


Figure 5.6: The fitness graph depicting the development of the population average and the best individuals over 20 generations of evolved robotic cars equipped with the omnidirectional camera, driving the ellipse shaped circuit.

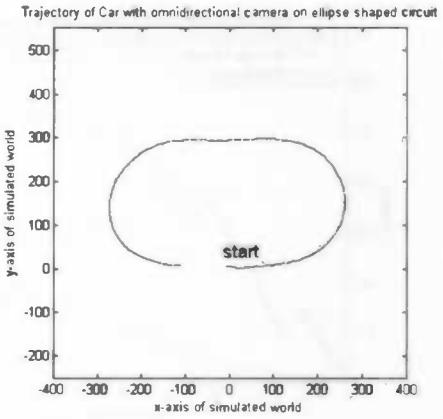


Figure 5.7: The trajectory of the car equipped with the omnidirectional camera on the ellipse shaped circuit.

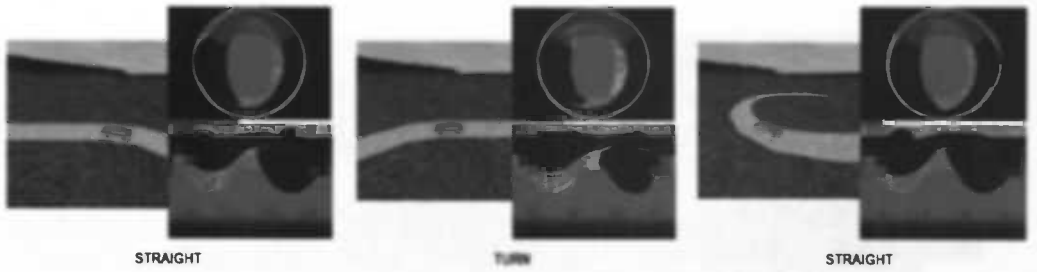


Figure 5.8: The strategy of the best-evolved individual driving counterclockwise in a left bending curve of the circuit. The car keeps much road at its right side to perceive better the diminishing of the road when a curve starts. It correlates activation and contra-activation in the left half and the right half of its retina respectively, with steering corrections to the left.

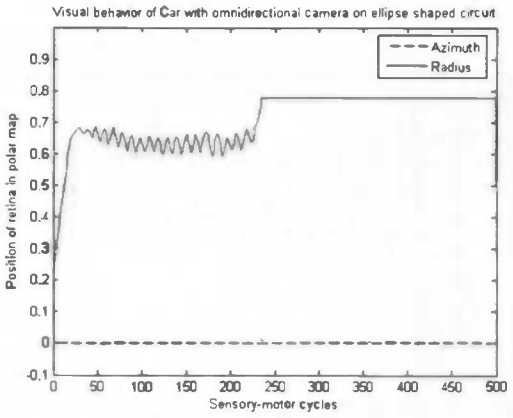


Figure 5.9: The retina behavior of the best-evolved individual on the ellipse shaped circuit. The retina looks forward (azimuth of zero degrees) to the road ahead. The radius grows to the maximal value (near 0.8 with the used retina size).

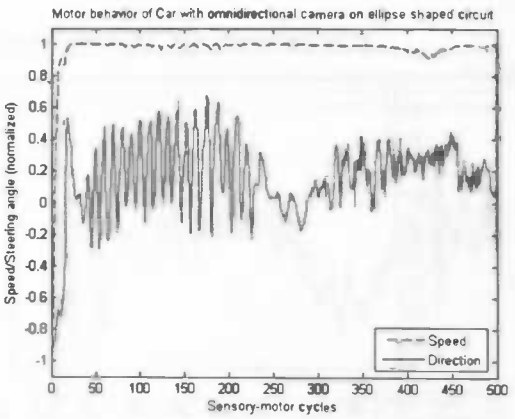


Figure 5.10: The speed and steering behavior of the best-evolved individual equipped with the omnidirectional camera, driving the ellipse shaped circuit. The car is driving at full-speed almost all the time, with little breaking during steering. It constantly corrects its heading, mainly in the curves.

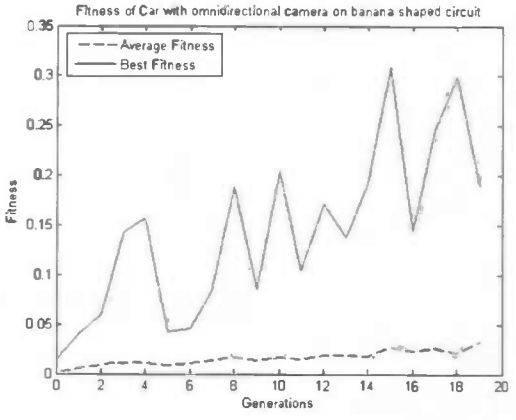


Figure 5.11: The fitness graph depicting the development of the population average and the best individuals over 20 generations of evolved robotic cars equipped with the omnidirectional camera, driving the banana shaped circuit.

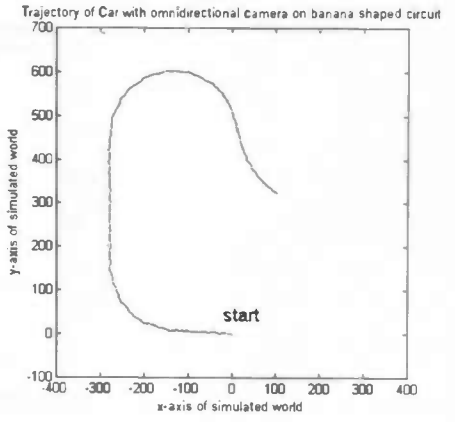


Figure 5.12: The trajectory of the car equipped with the omnidirectional camera on the banana shaped circuit.

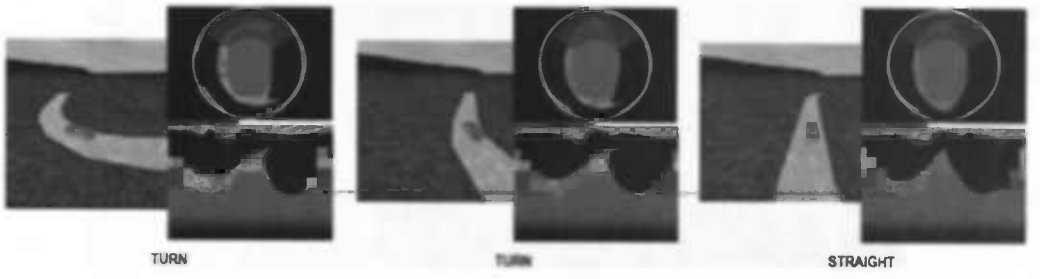


Figure 5.13: The evolved strategy of the best-evolved individual driving clockwise in a right bending curve of the circuit. The retina looks just in front of the car in a curve and looks further ahead in straight parts. The car correlates wiggling of the retina with steering corrections.

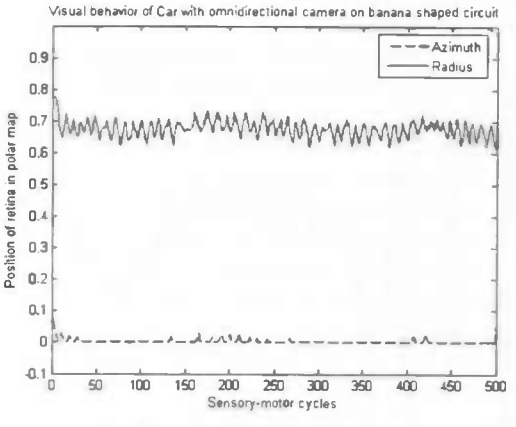


Figure 5.14: The retina behavior of the best-evolved individual equipped with the omnidirectional camera, driving the banana shaped circuit. The retina looks forward (azimuth of zero degrees) and wiggles around the location of the road ahead.

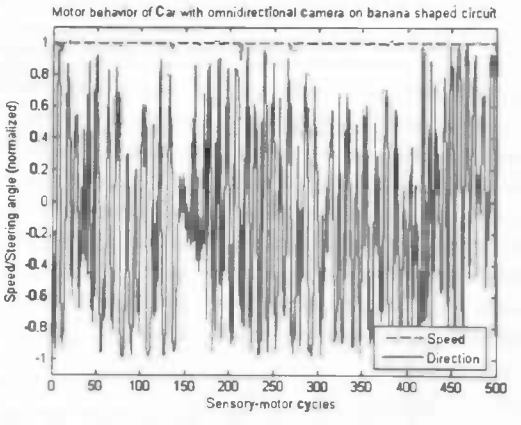


Figure 5.15: The speed and steering behavior of the best-evolved individual equipped with the omnidirectional camera, driving the banana shaped circuit. The car is driving at full-speed all the time. It constantly wiggles in its heading, in the curves as well as on the straight parts.

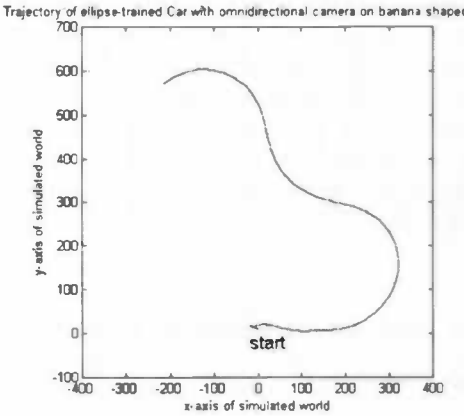


Figure 5.16: The trajectory of the ellipse-evolved car equipped with the omnidirectional camera on the banana shaped circuit. At the start, the car drives a litlle backwards and first scans the environment with its retina to localize the right heading direction. Once it finds the road, it starts driving at full-speed.

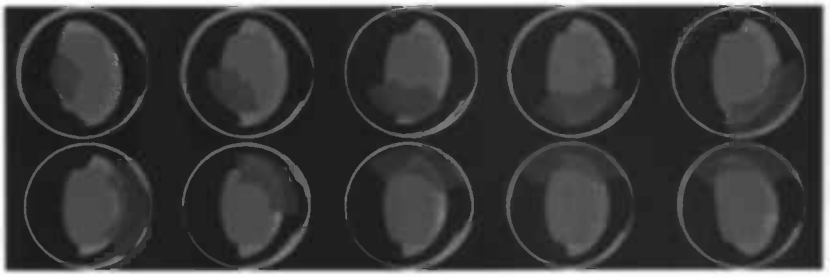


Figure 5.17: The retina scanning behavior in the starting phase of the trial, corresponding with the slow backward movement shown in Figure 5.16. The retina takes a long time to end up at the location with the appropriate features, and in the mean time the car is susceptible to driving off-road.

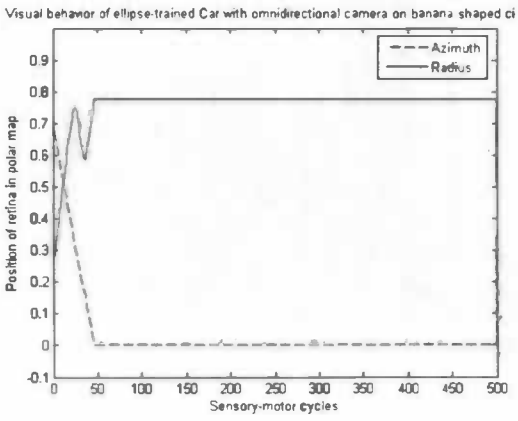


Figure 5.18: The retina displacement of the total trial shows a scanning phase at the start of the trial, before ending up at a fixed position used during the rest of the trial.

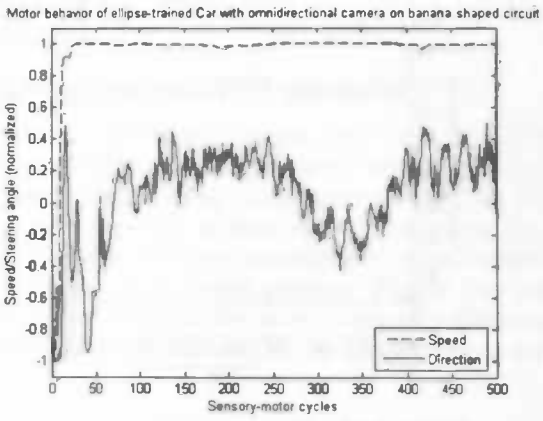


Figure 5.19: The speed and steering behavior of the ellipse-evolved, driving the banana shaped circuit. The car is driving at full-speed almost all the time and uses rather sensitive steering corrections.

Chapter 6

Discussion

6.1 Evolved Omnidirectional Strategies

The evolved active perceptual behavior of the robotic scale car achieves good performance on the task of driving the two circuits at high-speed. All successful individuals are able to drive at the maximum speed and cope with the corresponding physical forces during steering.

A common characteristic emerging from two selective benefits is the tendency of the car to drive closely to the inner side of the circuit. Firstly, as already mentioned, this strategy enhances the perceptual difference between a straight part and a curve, and simplifies thus the discrimination task. Secondly, it is as well supporting the car to drive at high-speed, since it reduces the centrifugal force pushing it out of the circuit, when the steering is done in small gradual steps.

The visual strategy consists generally of selecting the part of the visual scene that provides the information of the approaching road. This is naturally the best part to select in the context of the given task, as it contains the features useful for anticipation of an upcoming curve. Because of this straightforward place of looking, the resulting retina behavior is not needed to be very active; it can do with staying at the same place. Once this location is discovered by evolution and co-evolved with the appropriate receptive fields development to use the right features, active retina displacement becomes rather unnecessary. However, in the case of the banana shaped circuit, more active retina movements are evolved. This seems beneficial when more anticipation is needed, i.e. when the environment is less predictable. In the case of the best-evolved individual on the banana shaped circuit, the retina looks farther ahead on a straight part to see a curve coming in time, and focuses on the specific curve during the turn by looking nearer to the car.

Moreover, even when one fixed location of the retina in the visual scene can provide all features, the retina still needs to find this location at the start of a trial. Since the car is positioned randomly, it has to scan first the environment to find out where the useful information is, before fixating its retina at the right location in the omnidirectional visual scene and start heading the right direction. During this disorientated phase the car is susceptible to drive off the circuit, and that is the reason why even the best-evolved individuals do not make successful trials every time. The car is moving without visual appropriate guidance until the retina arrives at the appropriate location. Individuals equipped with the omnidirectional camera gain remarkably lower fitness than the individuals with a pan-tilt camera. This is not surprising, because the pan-tilt camera set-up has a much smaller search space in this task to find the location with the appropriate features, as its selection range is already directed to the front of the car. In the case of the omnidirectional camera the retina can be very unlucky to be initially positioned looking behind the car, so it takes a long risky time to move to the front in the visual scene.

The best-evolved individuals make often a back and forth movement or a slight turn in this period, followed by an abrupt impulsive correction as soon as their brain knows where to go. As the beginning of each trial is the most difficult period, evolution will favor the individuals capable of quickly detecting the right retina location. This is nicely shown in the trial with the ellipse-evolved individual on the banana shaped circuit. It slows-down its motor behavior while its visual behavior is still initializing.

Almost all individuals make use of the averaging filter method, which points out that the different pixels within each receptive field of the retina have all informative value and cannot be represented by one pixel in the center of the receptive field. This is a clue that activation of the zooming capability of the retina can be helpful to select more precisely, since it allows to change the resolution of the retina to let each receptive field have the size of the distinct features in the image. A zooming retina is able to select much more precisely the currently needed features in the visual scene, and more active displacement of the retina through the omnidirectional image can be expected. Furthermore, it can be helpful for the car in the disorientated starting period, because when it is allowed to enlarge its retina in this phase, it might find faster the indications in the visual scene to drive to the right direction.

6.2 Advantages of Active Vision with the Omnidirectional Camera

As stated, the use of an omnidirectional camera provides the possibility to access the visual scene in any direction fast and easily, provided the useful features can be selected from the excessive amount of information. As shown in the presented results, co-evolution of Active Vision and feature selection successfully achieves good performance in interactively selecting the task-relevant features during high-speed navigation.

When we compare the developed strategies on the ellipse shaped circuit from the individuals equipped with an omnidirectional camera and the individuals equipped with a pan-tilt camera, the first thing to be noticed is that the pan-tilt camera yields rather reactive and rough behavior, while the omnidirectional camera yields more anticipatory and sensitive behavior. This is due to the fact that the car can look further ahead by using the more informative omnidirectional image, while the pan-tilt camera only looks to the side and cannot gradually adapt its heading until a shift in the orientation of the side of the road appears. The results show therefore a smoother trajectory with less sudden steering in the case of an omnidirectional camera. The development of this strategy is only possible when the retina is able to find the right location fast enough in the disoriented starting period, which stresses the importance of the capability of the omnidirectional camera to quickly scan the whole environmental scene and find appropriate features.

Because of the larger freedom in selecting certain parts of the visual scene, a robot equipped with the omnidirectional camera can adapt faster to the situation, as it is able to virtually jump from one place to another in the visual scene. In this study only a maximal displacement per sensory-motor cycle of 20 pixels is allowed, but in principle no constraint is necessary. As a side note should be noticed that when the size of displacement is not restricted, it could become hard for the system to associate information from one sensory-motor cycle to the other, as the neighboring information is not inherent to the sequence of cycles anymore, so a modified neural architecture capable of dealing with that might be needed. Nevertheless, the possibility emphasizes that the omnidirectional camera is a perfect test bed for Active Vision to operate on.

6.3 Future Work

To extend the current promising results of this study further, multiple suggestions could be made. First the already stressed possible improvement of a zooming retina would probably yield better results. Furthermore, the robustness of the evolved behaviors can be investigated by diminishing the contrast of the road with the environment, or by using other textures. Additionally a guiding white sideline can be created as the single feature to discriminate the road from the environment with, and more active anticipatory visual behavior can be expected by creating discontinuities in the guiding features, e.g. a dashed sideline. Finally, the resulting behavior can be examined by letting another car drive the same circuit in the same or opposite direction. For these non-reactive, multi-factor behaviors, it might be interesting to explore the use of two retinas selecting simultaneously features from the visual scene.

While modeling the car, the omnidirectional camera and the environment, a possible transfer to reality is taken into account. Successful experiments in simulation can in principle be repeated in reality, with the real robotic scale car on a real circuit. However, since we made some simplifications in simulating the real world, the behavior might not have exactly the same success in the real world. To overcome this problem, individuals that are to be transferred to reality should also be capable of some simple ontogenetical learning, to be able

to fine-tune the basic regularities learned in simulation with respect to the actual environmental characteristics during life-time.

Chapter 7

Conclusion

This thesis has described the generation of active perceptual behavior of robotic scale cars equipped with an omnidirectional camera navigating two different shaped circuits at high-speed. By co-developing Active Vision and feature selection in a simulated evolutionary process, the synaptic weights of deliberately simple neural controllers have been set to yield successful behavior that is able to select only the needed task-dependent features in parts of the total available omnidirectional visual scene.

A 3-D robot simulator based on a realistic physics simulator has been modified and extended in the context of the aim of this thesis to meet the needs of the automation of an evolutionary process yielding succesful navigation strategies, in which the sensory-motor coordination of the individuals can be tuned to realistic environmental interactions and real-time constraints. Furthermore, an existing robotic scale car has been modeled in simulation, together with the characteristics of an omnidirectional camera, and different kinds of circuits. Subsequently, an artificial retina, able to move through the visual scene to select the appropriate features, has been made possible to operate on the omnidirectional camera input.

In this thesis it has been shown that co-evolution of Active Vision and feature selection in behavioral robots achieves good performance in sequentially and interactively selecting useful visual features in the information-abundant omnidirectional visual scene, with respect to the circuit navigation task. Successfully evolved individuals managed to drive the circuits efficiently at high-speed, without going off-road. Developed strategies used the retina to look forward at the approaching road, to be able to correlate an upcoming curve with anticipating steering corrections.

The use of the omnidirectional camera turned out to be advantageous over a standard pan-tilt camera, because it provided a more informative visual scene, which could be fast and easily accessed without mechanical control. This resulted in selection of more useful features and a more sensitive and anticipatory behavior. The drawback of the use of omnidirectional input was encountered in the initial random position of the retina, which can cause in unfortunate cases a long period of disorientation before the appropriate location of the useful features is found, due to the large omnidirectional search space. Evolution has thus favored individuals that could cope the best with this difficulty, and some best-evolved individuals showed initial scanning behavior before starting a full-speed decisive heading.

Bibliography

[1] K.E. Adolph. Specificity of learning: Why infants fall over a veritable cliff. *Psychological Science*, 11:290–295, 2000.

[2] A. Béguin. Active vision with omnidirectional camera, February 2005. Semester Project, Laboratory of Autonomous Systems 2, Swiss Federal Institute of Technology Lausanne.

[3] R.A. Brooks. Intelligence without reason. In J. Mylopoulos and R. Reiter, editors, *Proceedings of 12th International Joint Conference on Artificial Intelligence*. San Mateo, CA:Morgan Kaufmann, 1991a.

[4] A. Clark. *Being There. Putting Brain, Body, and World Together Again.* Cambridge, MA:MIT Press, 1997.

[5] F. Mondada et al. Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17:193221, 2004.

[6] D. Floreano, T. Kato, D. Marocco, and E. Sauser. Coevolution of active vision and feature selection. *Biological Cybernetics*, 90:218–228, 2004.

[7] K. Gurney. *An Introduction to Neural Networks.* UCL Press, London, 1997.

[8] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–464, 1990.

[9] S. Haykin. *Neural Networks. A comprehensive Foundation.* Englewood Cliffs, NJ:Macmillan, 1994.

[10] J.H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI:University of Michigan Press, 1975.

[11] CMLabs Simulations Inc. *Vortex Viewer Developer Guide*, 2001. Montreal, Canada.

[12] CMLabs Simulations Inc. *Vortex Developer Guide, version 2.0.1*, 2002. Montreal, Canada.

[13] CMLabs Simulations Inc. *Vortex File Format Developer Guide, version 2.0*, 2002. Montreal, Canada.

[14] C. Langton. *Artificial Life.* Addison -Wesley, 1989.

[15] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[16] S. Nolfi and D. Floreano. *Evolutionary Robotics. The biology, Intelligence, and Technology of Self-Organizing Machines.* Cambridge, MA:MIT Press, 2000.

[17] K. O'Regan and A. Noë. A sensorimotor account of vision and visual consciousness. *Behavioral and Brain Sciences*, 24:939–1031, 2001.

[18] R. Pfeifer and C. Scheier. *Understanding Intelligence.* Cambridge, MA:MIT Press, 1999.

[19] O. Rutti. Evolutionary active vision for all-terrain robots. Master's thesis, Laboratory of Autonomous Systems 2, Swiss Federal Institute of Technology, Lausanne, Switzerland, 2003.

[20] E. Sauser. Evolutionary car game, June 2002. Semester Project, Laboratory of Autonomous Systems 2, Swiss Federal Institute of Technology Lausanne.

[21] M. Suzuki. Adaptive neural controller for autonomous outdoor navigation. Master's thesis, Graduate School of Science and Engineering, Waseda University, Tokyo, Japan, 2004.

[22] J.D. van der Blij. 1:10 scale car with omnidirectional camera, April 2005. Internal Report, Laboratory of Intelligent Systems, Swiss Federal Institute of Technology Lausanne.

[23] J.D. van der Blij. 1:10 scale car with omnidirectional camera: Vortex modelling report, June 2005. Internal Report, Laboratory of Intelligent Systems, Swiss Federal Institute of Technology Lausanne.