NTNU

Norwegian University of
Science and Technology

Combining offline and online learning in developing an adaptive controller for a simulated car racing environment

Snorre Christoffer Corneliussen
Magnus Westergaard

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The goal is to implement an artificial intelligence car controller to be entered in an annual simulated car racing competition. By combining offline and online learning the controller should be able to adapt to new tracks in unknown environments. The controller will be compared to previous entries that have performed well, both in terms of the methodologies used and racing performance.

Assignment given:	17 January, 2011
Supervisor:	Keith L. Downing, IDI

Abstract

This report presents the work done to develop an autonomous driver for the Simulated Car Racing Championship (SCRC), a competition in computational intelligence based on The Open Racing Car Simulator (TORCS). Autonomous race driving based only on local sensory data is a complex problem, and previous SCRC entries' work show a wide variety of approaches taken to address it. We describe CRABCAR, a controller that combines offline learning prior to the competition with online learning during the competition to optimize its performance in the SCRC context. The presented approach extends and builds on track modelling and racing line optimization techniques introduced previously, addressing known problems said techniques have with noisy sensory input and non-perfect track information. CRABCAR's performance is compared to previous entries from the SCRC, with results showing CRABCAR at a performance level similar to the others. We conclude that a system for online adaption is essential when pre-learned strategies are applied to discretely segmented and non-perfect track models in the SCRC context.

Acknowledgements

It is a pleasure to thank the many people who have made this thesis possible. First and foremost we are grateful to our supervisor, Professor Keith L. Downing. With his encouragement, knowledge and guiding advice we were able to complete the task we set out to do. In addition, we thank him for the financial support to attend GECCO 2011.

We also wish to thank the student colleagues closest to us, working alongside us in the robotics lab at IDI, NTNU. You provided the fun and stimulating environment we looked forward to be in and work in every day for the last year. Your feedback and enthusiasm has lifted our spirits and kept us going. Thank you Jannik Berg, Camilla Haukenes Karud, Erlend Hamberg, Petter Westby, Johannes Høydahl Jensen, Tiril Anette Langfeldt Rødland, Pedro León and Even Bruvik Frøyen.

Lastly we wish to extend our gratitude to our parents for their continued love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

International simulated racing car competitions have recently become a popular testbed for computational intelligence researchers. They provide a platform where methods and algorithms can be investigated and compared, and have the added lure of competing and winning against others in the same field of research. In particular, they invite researchers to use and combine artificial intelligence (AI) techniques in novel ways and use them in a more complex testbed than those normally used to illustrate performance in scientific publications. The competitions in many cases represent something more similar to a real-world problem.

The Simulated Car Racing Championship (SCRC) was first held in its current form as part of IEEE Computational Intelligence and Games Symposium (CIG) and IEEE World Conference on Computational Intelligence (WCCI) in 2008. The simulator TORCS, modelling advanced physics and race car dynamics in a nice 3D graphics visual package, was the basis for the competition. Relying only on local sensory data from a car, participants were tasked with designing a controller that drove the race car around a unknown tracks with the possibility of multiple opponents racing at the same time. The championship has had largely the same format in recent years, with two important changes in added in the 2010 version. An extra challenge presented itself with the introduction of noise on the sensors used in distance measurements, and a warm-up stage was added. In this stage, controllers were allowed to drive alone on the tracks - prior to the part of the competition where they would be scored on performance - for a limited amount of time, providing an opportunity to obtain knowledge about the track layouts. This knowledge can be utilized in the later stages when the controllers are competing against each other. The SCRC is made up of several individual competitions, each of which happens at different conferences and with different racing tracks.

Although there is little time for *online* learning (i.e. during a competition) without

risk, there is no limit to the amount of time available for *offline* learning prior to competitions. This can be exploited to learn general behavior applicable to any racing track environment presented in the competitions. The warm-up stage of the competitions can then possibly be used for adapting and optimizing behavior specially for that particular track.

Beside providing a competitive testbed for researchers, the competitions have two goals. Firstly, commercial computer racing games can benefit from the research. Most commercial games utilize controllers hand-coded by the developers and domain experts, which is a hard and time-consuming process. Such controllers do most likely have access to more of the game state than controllers in the SCRC.

The other goal of these competitions is to encourage development of techniques applicable for real world scenarios. The development of autonomous cars is an important field of research. Predicted benefits of having fully autonomous cars are improved safety and comfort and reduced fuel consumption and emissions. Over the last few decades many different approaches to this problem have been studied, and successful prototypes have been demonstrated (see Section 2.2). General Motors expect semi-autonomous cars on the roads by 2015, i.e. cars that need a driver to navigate complex junctions and busy city streets but are autonomous on the highway. They envision fully autonomous cars being available by 2020 [1].

The motivation for doing the work described in this thesis is two-fold. Firstly, we wish to get a good understanding of this particular field of research by seeking out knowledge about the SCRC competition setting and how AI techniques can be applied to solve the problem it poses. In particular, we wish to address the problem of using an abstract model of a race track produced under noisy conditions, and the potential in utilizing driving strategies learned offline prior to the race to improve racing performance. Secondly, we are to implement a car controller system of our own and participate in the SCRC hosted at the Genetic and Evolutionary Computation Conference (GECCO) in Dublin, Ireland 2011. This system will in this thesis be referred to as CRABCAR.

The SCRC introduces many interesting AI challenges. Being able to map local sensor information to actions that make the car race fast is not a trivial task, and many different approaches have been studied, using techniques such as artificial neural networks, evolutionary algorithms and fuzzy logic (see Section 2.7). Since the race tracks used in the competitions are unknown, it is difficult to plan ahead without first learning the track. There have been attempts at building abstract models, usually by abstracting the track as a set of segments with properties concerning length, curvature and more.

Our primary research questions, which we will investigate in this thesis, are:

Whether or not it is possible to build and effectively use a track model in the noisy

environment, and how offline learning can can aid performance when there is very limited time for online learning.

In this thesis we develop a car controller system that communicates with TORCS using the SCRC framework. The controller combines online and offline learning techniques to create a model of its environment and exploit this information in attempt to optimize racing performance (i.e. driving as quickly as possible while remaining on the track and avoiding collisions).

Evolutionary learning is utilized offline to optimize parameters related to general driving (e.g. when to gear) and handling of different turn types, such as preferable speed and positioning on the track prior to the turn. Online, we apply a turn classification technique inspired by Quadflieg et al. [2], but modified to handle noise, to create an internal model of the track to allow planning ahead. While the model is created during the first lap of a race, the controller is able to adapt its behaviour in subsequent laps to allow for adjustments to increase performance and decrease error. The track model is updated with these adjustments if they prove to produce a faster driver.

This thesis is structured as follows: Chapter 2 begins with a description of the problem. After that, a brief history of the research field of making autonomous vehicles, both in the context of real prototypes and simulations, is presented. The differences between the two are briefly discussed. This is followed by an introduction to the related work from the recent past and what is currently being done in the field.

Chapter 3 describes the methods and design behind the CRABCAR system of this report. Chapter 4 presents the experiments that have been conducted and lists and discusses the results of testing the system. Chapter 5 summarizes the work, concludes the thesis and discusses implications for future work.

Chapter 2

Background

In this chapter we introduce the problem area of the work presented in this thesis. The history of autonomous driving is briefly explored, before going into details regarding the simulated car racing environment which our system will focus towards. Some necessary concepts are described, and we take a look at other systems that have been used to solve similar problems.

2.1 The problem

Designing a controller for a car robot that performs well with only limited local information available is a complex task. The problem is producing a good mapping between the noisy input sensory signals and the actions of the vehicle, such that the car is able to drive as fast as possible while staying on the track, and at the same time dealing with dynamic changes in the environment in the form of opponents. This includes managing speed and position in arbitrary turns, handling S-shaped chicanes at the end of high-speed straights, avoiding collisions and overtaking opponents. It should be able to do so on any racing track without any a priori knowledge of the layout, except from what is gathered through the sensors during the time-limited warm-up stage (see Section 2.3) of a competition. This makes it difficult to produce an internal model of the track for planning purposes. Coupled with the facts that there was no warm-up stage or noisy sensors prior to the 2010 championship, the majority of the work done regarding the racing car bot problem has focused on reactive controllers with little or no internal state (see Section 2.7).

2.2 A brief history of autonomous driving

The first major step in the field of automous vehicles can allegedly[1] be dated back to 1977, when Japanese researchers at the Tsukuba mechanical engineering laboratory produced a driverless car which was able to follow a specially made course. It used specialized hardware to follow white markers on the road, and achieved speeds of up to 20 mph. Several major research projects that would advance the field were undertaken in the 1980s.

Ernst Dickmanns and his team at the Bundeswehr University of Munich (UniBwM) from Germany collaborated with Daimler Benz, and modified a Mercedes-Benz van so that it was possible to control throttle, brakes and steering via a computer [3]. It was fitted with a UniBwM image processing system and was in 1987 able to achieve speeds of 96 km/h over a 20 km stretch on a blocked highway.

In the United States, the Defense Advansed Research Projects Agency (DARPA) funded the Autonomous Land Vehicle (ALV) project [4]. The result was a 15,200 lb, eight-wheel vehicle fitted with a TV camera for computer vision and a laser scanner for range detection. Limited by processing power, it used 2-3 seconds to process an image and decide what to do next. Later iterations of the ALV was able to navigate off-road through terrain with obstacles.

In 1987, the Pan-European "PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety" (PROMETHEUS) started [5]. The EUREKA-funded project was a collaboration between several universities and car manufacturers, and among them Dickmanns and his team from UniBwM. In 1994, their re-engineered Mercedes 500 SELs VaMP and VITA-2 were used in the final presentation of PROMETHEUS. On a three-lane highway near Paris, France, they drove more than 1,000 km in normal traffic with a safety-driver onboard. They were capable of overtaking others, an action requiring them not only to interpret the world in front of them, but also behind them. This was the first of several long distance experiments to follow.

In 1995, a 1758 km trip from Munich to Odense in Denmark was made by VaMP, where a total of 95% of the distance covered was done autonomously [6]. The car's controller was based on saccadic computer vision, and it achieved speeds exceeding 175 km/h on the German Autobahn.

The "No hands across America" project from the Carnegie Mellon University (CMU) drove their vehicle NavLab5 from Pittsburgh, Pennsylvania to San Diego, California the same year [7]. Using a combination of computer vision, gyroscope and GPS, it achieved 98.2% autonomous driving (by distance).

[1] http://en.wikipedia.org/wiki/Driverless_car

In 1998, VisLab of the University of Parma's ARGO project completed a 2,000 km journey throughout Italy [8]. The fact that the vehicle was only fitted with two cheap black-and-white cameras to perform stereovision in order to navigate makes this achievement particularly notable. The journey was done 94% autonomously.

DARPA launched their first Grand Challenge in 2004 as the first of its kind in the world [9]. With the promise of a $1,000,000 first prize, all businesses and organizations from anywhere in the world were welcome to enter the contest, given that at least one US citizen was on the team roster. The first Grand Challenge was set in the Mojave desert, where contestants were to complete a 240 km route. None did. The next year, however, five vehicles successfully completed the Grand Challenge and nearly all the finalists surpassed the distance covered in the previous year's challenge [10]. The top contestants were teams from CMU and Stanford University, with Stanford as the winner. In 2007, the Grand Challenge was set in an urban environment [11]. This time, CMU was victorious, with Stanford coming in second place.

Two more recent accomplishments have earned attention from mainstream media. In October 2010, Google revealed that they had been working on driverless cars for several years, and that their fleet of test cars had travelled a total of 230,000 km without any incidient [12]. The project is lead by Sebastian Thrun, director of the Stanford Artificial Intelligence Laboratory, who was involved in the team that won the 2005 DARPA Grand Challenge. With him are several key personnel previously involved with successful DARPA Challenge teams. The goal of the Google project is better highway safety and a lower impact on the environment.

The second recent event was the VisLab Intercontinental Autonomous Challenge (VIAC) [13]. The 2010's World Expo in Shanghai, China had the theme "Better City - Better Life". VIAC's goal was to demonstrate that autonomous, environment friendly vehicles could cross extreme distances under normal traffic conditions. Over a three month period, four vehicles divided into two leader-follower pairs, journeyed from Milan, Italy to Shanghai. The leader vehicles provided routes for the follower vehicles to follow. Human interaction was only needed in special cases in the leader vehicles - the followers were fully autonomous. The journey totalled 20,000 km.

2.3 The Simulated Car Racing Championship

2.3.1 Origin and history

The first version of the simulated car racing competition was held in 2007, at the IEEE Congress on Evolutionary Computation (CEC) and the IEEE Symposium on

Computational Intelligence and Games (CIG) (changed name to IEEE Conference on Computational Intelligence and Games in 2010). The competition used a simple driving game based on a 2D physics model called simplerace. Two cars were pitted against each other, and the goal was to pass through as many randomized waypoints as possible within a certain time frame. Three waypoints existed at any given time of the game, of which only two were observable by the cars, and the cars had to pass through them in the correct order. Once one of the cars passed through a waypoint it would disappear. This meant that if a car was able to predict that its opponent would reach a waypoint first, it could plan accordingly and head straight for the second waypoint which would "activate" on the car's way there. The majority of the participants incorporated some form of decision module in their systems, which purpose was to figure out which of the two observable waypoints was worth pursuing. There was a broad spectrum of AI techniques used for navigation: recurrent multi-layer perceptron (RMLP) neural networks, genetic programming, continuous-time recurrent neural networks (CTRNN), a collection of fuzzy systems and force fields [14].

Several aspects of the competition were changed for 2008. TORCS was introduced as the replacement for the simple 2D driving game used in the competition's first year. The motivation behind this change was to introduce more complexity for the controllers to tackle, with the ability to have more than two cars on the same track simultaneously. It would demonstrate that computational intelligence could be used for more than academically conceived benchmarks, and hopefully attract more attention with its advanced 3D graphics [15].

Several new rules were added to the competition, most importantly a set interval between the issuing of sensory data from the simulation to the controller. This meant that controllers would have to process the information and react to it quickly if they were to rely on the most recent sensory data. The limit came as a result of a lesson learned the year before: because they had relied on opponent-modelling, the top-scoring controllers had been very computationally expensive, and would take hours to run and evaluate [14].

In 2009, several simulated car racing competitions were organized into one big simulated car racing championship, the SCRC. There were three competitions, each consisting of three races, at three different conferences: IEEE CEC 2009, IEEE CIG 2009 and the 2009 ACM Genetic and Evolutionary Computation Conference (GECCO 2009) [16]. The rules of the championship were similar to those of the previous year. Thirteen teams participated, several of which were updated versions of controllers submitted to the 2008 competitions. The best controllers performed significantly better than the best controllers of 2008.

The 2010 SCRC was organized largely in the same fashion as the year before (see

Appendix A.1). The directions of the distance sensors on the cars were made customizable to the controllers, and noise was added to the distance sensors' signals. A warm-up stage was added, allowing controllers to drive alone on the tracks for a set amount of time without being scored on performance. This enabled them to potentially learn about the track layout and properties before the competitive qualifying and race stages. The new rules also introduced control of the clutch, adding even more complexity to the gear shifting and use of the brake and accelerator.

The only change made for the 2011 version of the SCRS was a decrease in the noise applied to sensors detecting opponents, and an increase in the noise applied to the sensors detecting the track edges. For a complete description of the competition setup, rules and regulations, see Appendix A.

2.3.2 Real vehicles vs. SCRC controllers

The projects mentioned in Section 2.2 concerning real cars have fundamentally different goals from the controllers developed for the SCRC. They focus on reliability, driver comfort, adhering to traffic laws and navigating difficult terrain instead of fast driving. All of them are based in part on computer vision techniques, which are not applicable to the SCRC in its current format. The more recent projects also include GPS as part of their navigation systems. The use of lasers and radars to locate obstacles and measure distances are real-world relatives of the range finding sensors of the simulator, but all in all, techniques used by these autonomous vehicles in the real world do not seem directly transferable to the SCRC controller context. The main focus of the literature study in this report has therefore been on previous work done in simpler, simulated environments.

2.4 The software environment

2.4.1 TORCS

The simulator used throughout the work done, both for experiments and the final competition, is The Open Racing Car Simulator (TORCS) [17]. Development first started in 1997, and was done by Eric Espié and Christophe Guionneau. At first it was a 2D simulator, which eventually evolved into simple 3D shapes. Engines, textures and more sophisticated physics have since been added, and although the original creators no longer oversee the project, an active community led by Bernhard Wymann have continued development [18].

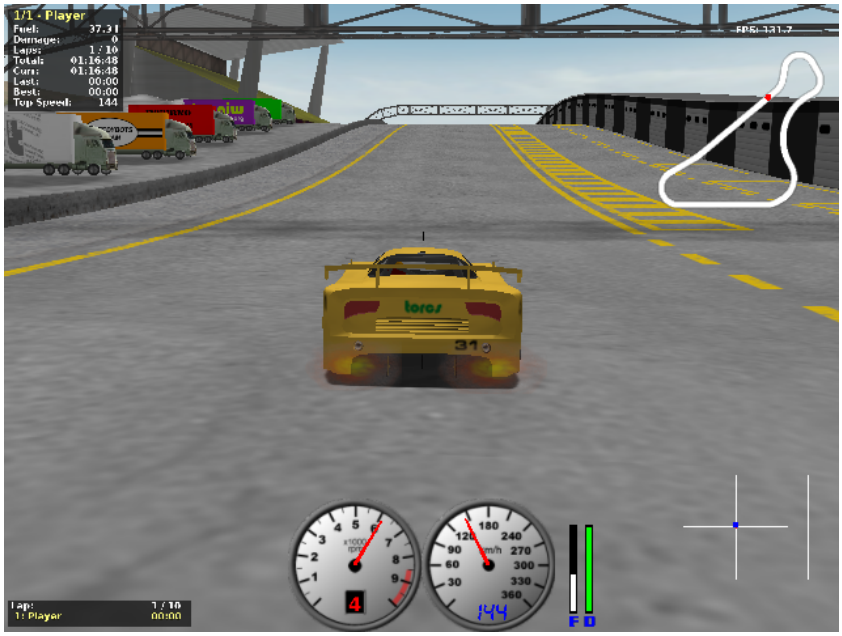


Figure 2.1: Screenshot from TORCS.

TORCS is licensed under GNU GPL [19] and Free Art License [20], and is currently available for all major platforms, including OS X, GNU/Linux and Microsoft Windows. It is written in C++ and uses OpenGL to produce the 3D graphics shown in Figure 2.1. Although it can function as a simple race car driving game, users are encouraged to create their own AI controlled "bots" and have them race against each other. The online community TORCS Racing Board [21] serves as a platform for such competitions, where anyone can participate and let their bot compete against those of other participants. Two other popular competitions are the Simulated Car Racing Championship and Demolition Derby. They both use modified versions of TORCS as platforms for the competitions.

2.4.2 The competition software

The Simulated Car Racing Championship committee provides a software package that, when installed, patches TORCS to the state used in the championship. TORCS provides users with a way to create, test and use their own bots. However, to control and limit the way user bots may interact with the simulator, the patch wraps a server-like entity around the original bot interface. Participants of the competition have to interact with TORCS through this server by implementing clients that adhere to the communications protocol defined by the SCRC committee. During a competition, every bot running in the simulator is a server with a

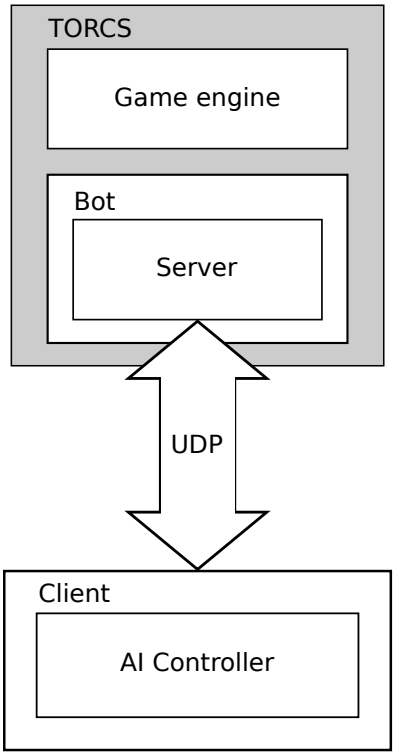


Figure 2.2: Competition software architecture. Communication between the race controller and the SCRC software incorporated with the TORCS framework.

corresponding client.

All communications happen via UDP. The communication flow between the controller and TORCS is shown in Figure 2.2. The serverbot sends the current state of sensors in the simulator to its client once every game tick, which corresponds to 20 ms of simulated time. The client then has 10 ms to respond to the server with what actions the bot should take. If the server receives no response, the actions of the previous game tick are repeated.

Decoupling the client from the simulator in this way means that as long as the controller client adheres to the server communications protocol defined by the committee, the client implementation can be done in a number of programming languages. Example implementations of very simple clients are provided for Java and C++.

The clearly defined interface and time limit, coupled with the freedom of language agnostic controller support, result in little overhead in the development of a controller. Good documentation of both the installation process of the TORCS patch and the client-server interface allow participants to focus their time on developing

the AI controller in the client. A short description of the sensors available to the controllers is listed in Table 2.1. The full description of sensors and acutators can be found in Appendix B.

The server also provides a few mechanisms for the user and client to control the race settings, such as race stop conditions, number of races and data output from the simulator. These mechanisms are designed to let the user run races in batch mode under defined conditions, a necessity when training and testing controllers. A text-only mode of the simulator makes it possible to run simulations at much faster speeds than real-time. This mode makes AI methods that require a lot of time, such as evolutionary computation methods, feasible.

2.5 Evolutionary computation

2.5.1 Evolutionary algorithms

Evolutionary algorithms (EAs) [23] seek to solve search problems by means of mechanisms inspired by those found in natural evolution. An initial population of randomly generated genotypes that encode candidate solutions (phenotypes) is evolved through many iterations, similar to natural generations. A fitness function developed from the criteria of the problem to be solved is used to evaluate phenotypes. The fitness is a measure of how well adapted a phenotype is to solve the problem, and the more fit individuals are given a bigger chance to reproduce and have their properties survive to the following generations. During reproduction, evolutionary operators such as mutation and crossover introduce diversity into the population, potentially creating novel and better solutions. The cycle of fitness evaluation and reproduction can continue as long as is needed, illustrated in Fig. 2.3.

The genetic representation, describing the elements of a genotype and how they are mapped to the phenotype, is problem dependent. A good genetic representation should produce meaningful phenotypes that are likely to produce better individuals in the reproduction process, and all the possible genotypes should cover as much of the solution space as possible. Genotypes are commonly encoded as bit strings, but can also be real values or tree structures.

The initial size of the individual population is often problem-dependent. If the fitness evaluation process is very time-consuming, for example with evolution of robots, relatively few individuals are used to be able to iterate over many generations. If the fitness landscape is such that small mutations are likely not to improve fitness, larger populations of hundreds or thousands of individuals are used.

Sensor name	Range	Description
angle	$[-\pi, +\pi]$	Angle between the car direction and the direction of the track axis.
damage	$[0, +\infty)$ (point)	Current damage of the car.
distFromStart	$[0, +\infty)$ (m)	Distance of the car from the start line along the track line.
distRaced	$[0, +\infty)$ (m)	Distance covered by the car from the beginning of the race.
opponents	$[0, 200]$ (m)	Vector of 36 opponent sensors: Each sensor covers a span of $\pi/18$ (10°) within a range of 200 meters and returns the distance of the closest opponent in the covered area.
rpm	$[2000, 10000]$ (rpm)	Number of revolutions per minute of the car engine.
speedX	$(-\infty, +\infty)$ (km/h)	Speed of the car along the longitudinal axis of the car.
track	$[0, 200]$ (m)	Vector of 19 range finder sensors: Each sensor returns the distance between the track edge and the car within a range of 200 meters.
trackPos	$(-\infty, +\infty)$	Distance between the car and the track axis. The value is normalized w.r.t. the track width: It is 0 when the car is on the track axis, -1 when the var is on the right edge of the track and $+1$ when it is on the left edge of the track. Values greater than 1 or smaller than-1 mean that the var is outside of the track.

Table 2.1: Some of the sensors available to the controllers.

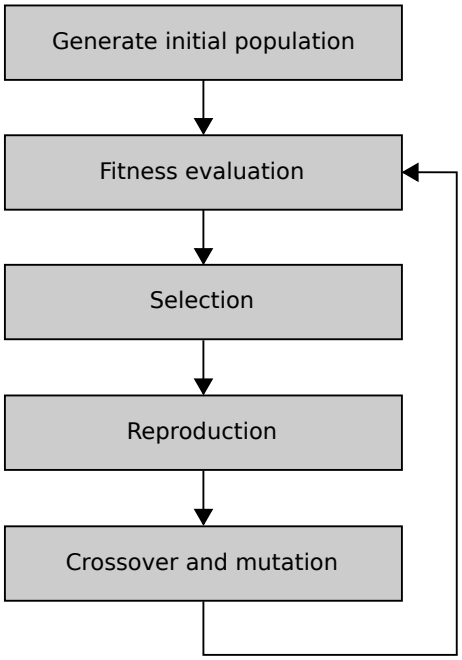


Figure 2.3: Cycle of operation for an evolutionary algorithm.

A selection strategy is used to determine which and how many individuals in an evaluated population that are allowed to reproduce. The selection pressure describes the percentage of the population that are allowed to produce offspring. High selection pressure means that few individuals produce the next generations of individuals, and vice versa. There are many selection strategies with different properties, working differently to maintain diversity in the population while at the same time maintaining the selection pressure.

The two evolutionary operators mutation and crossover ensure diversity and exploration of novel solutions in the reproduction process. Crossover makes offspring inherit features from both parents. Fig. 2.4 illustrates how three variants of crossover can be applied to real value-based genotypes. Mutation emulate the sudden changes that can occur in natural genetic material due to radiation, errors in transcription or other causes. In EAs it introduces random changes in genotypes, normally with very low probability. For bit string genotypes, a bit or more can be flipped; for real valued genotypes a random number can be added or subtracted.

When a new generation is generated, there are several ways of replacing the old one. Complete generational replacement means no individuals survive to the next generation, only their offspring. Gradual replacement can be enforced by replacing only a number of the least fit individuals with new offspring. It is also possible to

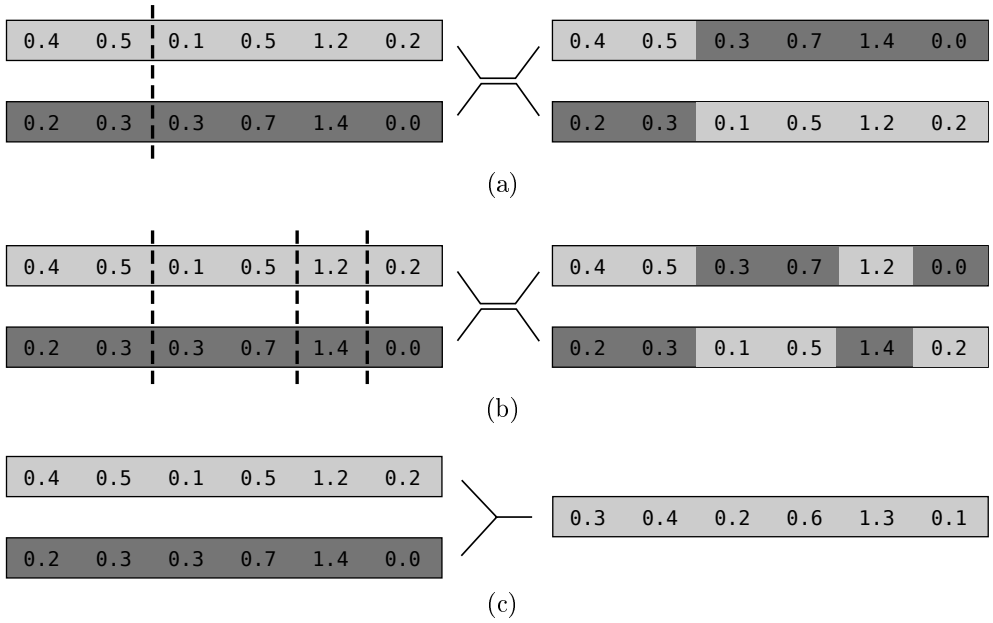


Figure 2.4: Crossover on real valued genotypes. (a) one-point crossover, (b) uniform crossover, (c) arithmetic crossover.

enable elitism, meaning that the best individual(s) are guaranteed to survive to the next generation.

2.5.2 Evolutionary programming

Evolutionary programming (EP) [22] is a variant of evolutionary algorithms. Instead of operating at the genotype level, it manipulates the parameters that define phenotypes directly. Mutations are based on adding random values from zero-mean gaussian distributions, and tournament-based selection is often the selection strategy used [23].

Tournament-based selection, illustrated in Fig. 2.5, works by drawing k random individuals from the population. The most fit individual of a tournament is allowed to reproduce, so the total number of tournaments equals the number of individuals that are to be produced for the next generation. Selection pressure can be adjusted by changing the tournament size. Bigger tournaments make it less likely that the less fit individuals are allowed to reproduce.

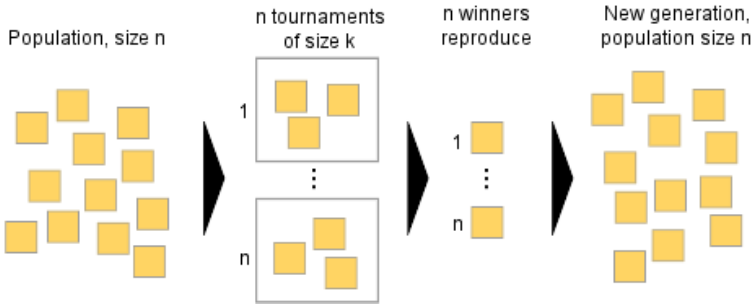


Figure 2.5: Tournament-selection as a selection strategy.

2.6 Behavior-Based Robotics

Behavior-based robotics (BBR) is a methodology to design and build robots to operate in real world environments [24]. Situatedness and embodiment is of importance in this approach, meaning that the robot control system should experience the world and make desicions based on its own sensory information in the actual environment, as opposed to dealing with abstract descriptions and/or simulations of the environment [23].

A behavior-based architecture consist of several independent modules of intelligence/behavior. Based on sensory input, each module calculates the appropriate behavior according to its goal, and the final action can depend on behaviors suggested from the different intelligence modules (see Figure 2.6). An example of this kind of architecture is the subsumption architecture, introduced by Brooks in 1986 [25]. In this approach, behavior modules are organized in layers, each implementing a goal of the agent, and the higher levels subsume the decision proposed by the early layers.

2.7 Related work

The Simulated Car Racing Championship has been a very transparent competition from the beginning. Information about previous submissions are available in several forms through the websites and proceedings associated with the various conferences that host the competition. Many of the participants release the source code of their projects, allowing previous and future competitors as well as anybody else interested to explore their systems in detail. There has also been published a number of papers on the subject, directly related to the major methods and techniques used in the submissions for the competition. As a result there has been

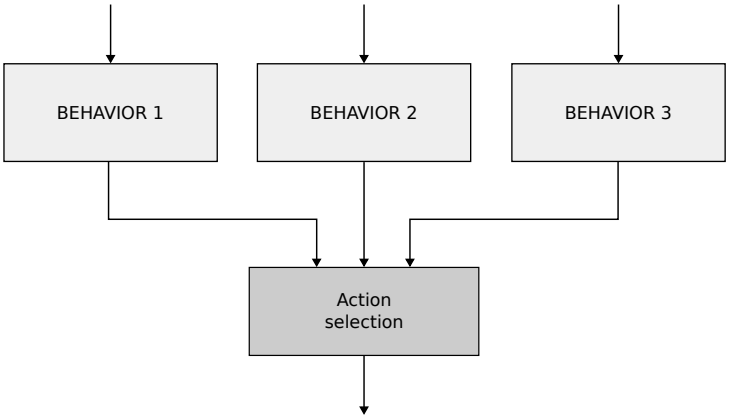


Figure 2.6: Behavior-based architecture. Different behavior modules suggests actions to take based on their goals and sensory information, and a decision is done based on these suggestions. Different architectures use different approaches in making this decision, i.e. interpolating suggestions, choosing the one with highest priority, etc.

noticeable "feature diffusion" [14], where participants have been inspired by each other's work and emulated seemingly successful features.

The different approaches investigated over the lifetime of the competition have ranged from using AI methods to develop controllers entirely, to hand-coding rulesets for the controller (there is no rule that forbids this), and different combinations of the two. In Section 2.7.1 we briefly describe many of the approaches taken and their most notable results. Sections 2.7.2 and 2.7.3 contain more detailed descriptions and results of the two approaches that have been the main inspiration for the approach proposed in this thesis.

2.7.1 Notable previous SCRC submissions

Matt Simmerson and Luigi Cardamone produced controllers based on Artificial Neural Networks (ANNs) developed using Neuro Evolving Augmented Topologies (NEAT) [15, 26], a technique that evolves both the structure and the weights of an ANN. It starts out small, and grows as nodes and links are added. Simmerson's controller intitially used three random sensory inputs. By the end of the evolutionary process, it used 9 of the available 36. The controller won the competition at IEEE WCCI 2008.

Butz and Lönneker [27] would challenge the NEAT approach with their COBOSTAR controller in 2009. Inspired by Braitenberg vehicles they devised strategies for a sensor-action mapping for both on-road and off-road racing, and op-

timized the paramaters of these strategies using a covariance matrix adaption evolution strategy. The optimized strategies were tested for their generality on tracks not used in the evolutionary process, and the most general parameters were used in the final controller which won the competition at IEEE CEC 2009.

Several controllers have utilized fuzzy logic [14]. Perez et al. [28] created a controller based wholly on a fuzzy logic . Their approach was to take an already competent controller that was able to navigate tracks in a simple way and use it as a seed for an evolutionary process which optimized its fuzzy sets.

Learning by imitation has been used as an approach to impose human-like behavior on artificial intelligence. This approach can generally be divided into two categories: Direct and indirect methods [29]. Direct methods try to extract behavior from training data sets describing the target behavior. Indirect methods use the same data to guide the development towards the target behavior through methods such as neuroevolution. Togelius et al. [29] attempted to model human behavior in car controllers as part of their effort to auto-generate racing tracks that a human would consider entertaining, as subjecting a real human player to the thousands of tests needed to determine the quality of a track was unfeasible. Early efforts were based on data recorded when a human player drove a car. First, a neural network controller was trained using backpropagation. Even though the error rate of the network after training was very low, they achieved very little success with this approach. Using the same training data, another controller used a k-nearest neighbor classifier to determine which output in the training data best matched the current sensory input. This controller performed well, but was prone to missing turns and crashing as time went on. Having no recovery mechanism, it got stuck.

van Hoorn et al. [30] used TORCS to investigate the question of whether there would be a trade-off between driving like a human and driving as good as possible in the game. To make controllers display human-like behavior, keyboard input logs were recorded from a human player and used to train them. These were then compared to controllers developed to perform as good as possible. They found it hard to produce controllers that were as human-like as they desired, but had good results with the controllers only focusing on being good at the game.

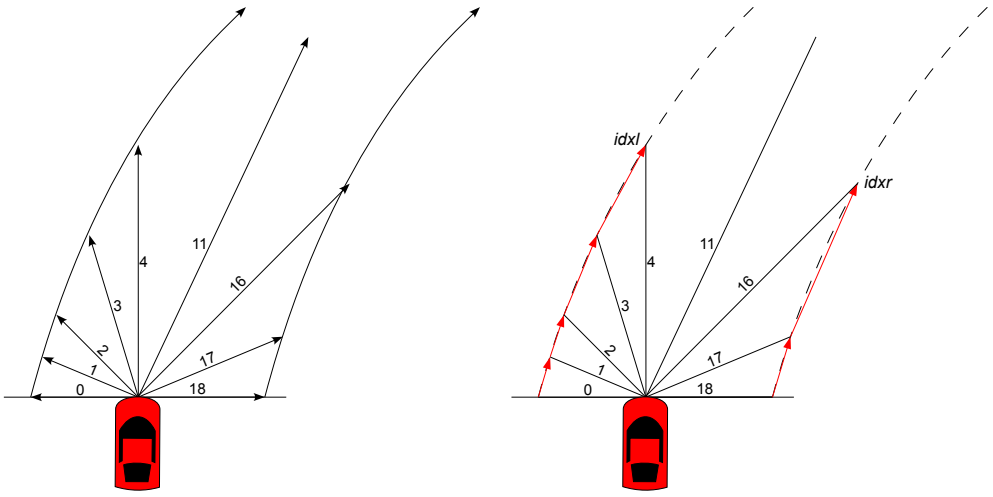
Cardamone et al. [31] extended the approach of using a direct method to train a car controller and applied their approach to TORCS. By using high-level information about a track and predicting high-level actions, they were able to produce a controller that achieved performance 15% lower than the best TORCS bot available at the time. The high-level information was acquired through a specially made sensor that detected the curvature of the track ahead, making the approach impossible to use in the SCRC context. Controllers that used the traditional sensor

setup from the SCRC behaved much like the controllers of Togelius et al. [29].

The issue of *perceptual aliasing* has been presented as a cause for the poor performance of controllers trained using direct methods [31]. This occurs when the target behavior is inconsistent in its mapping of sensory input to actions. If a TORCS controller with the sensors and actions available in the SCRC is to be trained using a data set obtained from a human driver or another controller, an example of inconsistent behaviour can occur on straight sections of the track prior to a turn. The human player will in many cases be able to see the turn up ahead, and position herself on the right for a left-hand turn and vice versa, allowing higher speed turns. The sensors of a controller are not able to detect turns beyond their range. If, on two different straight sections that produce the same sensory input to the controller, the human decides to position herself left on the first straight and right on the second in preperation of turns outside of the controller sensors' range, the same sensor values would map to two different actions in the data set being recorded. The SCRC rule changes from 2009 to 2010 included an increase in the range of the track edge sensors from 100 m to 200 m. Such an increase in range can reduce the problem of perceptual aliasing and improve performance of controllers trained for imitation through direct methods [31].

No information about a racing track's layout is known by the car controllers in SCRC before the competitions. With the inclusion of a warm-up stage in 2010, learning the layout and building a track model to use for planning became a more viable approach to producing a competitive controller.

Muñoz et al. [32] deviced a method to build a track model by analyzing the speed of the rear wheels of the car. Assuming that the two wheels' trajectories were on the radii of concentric when the car is moving circles and knowing the distance between the two rear wheels, they calculated the difference in angle between consecutive sampling points all along the track. All the edge detection sensors were pointed perpendicularly to the car's axis out on both sides of the car. The average provided an estimate for the distance to either side of the track from the car. Knowing the angle between the car's axis and the track direction, they were able reconstructed a track model. Comparing the reconstructed tracks with the originals revealed som discrepancies, with 4 out of 5 track models being discontinuous. All the turns of the original tracks had a corresponding turn in the track models, but the shapes were, in general, less sharp and less detailed than the original. The end result was a track model described by the angles between cross-sections of the track at 4m intervals. The track was used in an attempt to produce a human-like controller, employing several neural networks to predict the trajectory of humans. The pure racing performance results were not impressive as a result. However, their entry in the WCCI 2010 leg of SCRC (presumably tweaked since the publishing of their results) won the competition.



(a) (b)

Figure 2.7: (a) Illustration of track edge sensors s_i, which return the distance to the edge of the track at different angles v_i with respect to the car axis. $-90° \leq v_i \leq 90°$. (b) The red vectors illustrates the vectors $\mathbf{l}_i$ and $\mathbf{r}_i$, which define the outline of the track. Not all sensors and vectors are drawn in the illutrations for the sake of readability.

2.7.2 Segment-based track modelling

Quadflieg et al. [2] proposed a track segment classification system capable of modelling a race track by processing sensory data recorded over one lap on any unknown track. The track modelling process is based around p, a property derived from the sensory data to describe the curvature of the track in front of the car. The method takes use of the 19 *track edge* sensors s_i, which return how far the car is from the track edges at the different angles v_i (see Fig. 2.7a). First, the track edge sensors are used to construct two-dimensional vectors $\mathbf{s}_i$ using a local coordinate system with the car in the origin. The vectors are representations of the sensor signals themselves, incorporating their directions as well as their length. This is done using the formula shown in 2.7.1.

$$\mathbf{s}_i = \begin{pmatrix} -\cos\left(v_i \cdot \frac{\pi}{180}\right) \cdot s_i \\ \sin\left(v_i \cdot \frac{\pi}{180}\right) \cdot s_i \end{pmatrix}, \; i \in \{0, \ldots, 18\} \tag{2.7.1}$$

As seen in Fig. 2.7a, some sensors will return the distance to the left side of the track and some to the right side of the track, while some may, due to limited sensory range, return the maximum value of the sensor which is 200.0. The next step is finding $idxl$ and $idxr$, where $\mathbf{s}_i$, $i \in \{0, indxl\}$ represents the sensors returning a

distance measure to the left side of the track and$\mathbf{s}_{18-i}$, $i \in \{0, idxr\}$ to the right side. As can be seen in the figure, the vectors $\mathbf{s}_i$, $i \in \{0, idxl\}$ are increasing in length as i increases, so $idxl$ is found by first setting it to 0 and then iterating through the original sensor values from left to right and setting $idxl = i$ if $s_i > s_{indl}$ and $s_i < 200.0$. The $idxr$ is found in the same way, by being initialized to 18 and iterating from right to left.

Based on the indices $idxl$ and $idxr$ two new sets of vectors are computed as shown in equations 2.7.2 and 2.7.3.

$$\mathbf{l}_i = \mathbf{s}_{i+1} - \mathbf{s}_i,\ i \in 0, \ldots, idxl - 1 \tag{2.7.2}$$

$$\mathbf{r}_i = \mathbf{s}_{17-i} - \mathbf{s}_{18-i},\ i \in 0, \ldots, 17 - idxr \tag{2.7.3}$$

The vectors $\mathbf{l}_i$ and $\mathbf{r}_i$ define the outline of the left and right side of the track, respectively (see Fig. 2.7b). Next, we seek to find p, which is the measure of the curvature. This is done by summing together the angles between two consecutive vectors in the left and right vector set:

$$p_l = \sum_{i=0}^{idxl-2} \left[\arccos\left(\hat{\mathbf{l}}_i \cdot \hat{\mathbf{l}}_{i+1}\right) \cdot \frac{180^\circ}{\pi}\right] \cdot sgn\left(\hat{\mathbf{l}}_{i,x} \cdot \hat{\mathbf{l}}_{i+1,y} - \hat{\mathbf{l}}_{i,y} \cdot \hat{\mathbf{l}}_{i+1,x}\right) \tag{2.7.4}$$

$$p_r = \sum_{i=0}^{16-idxr} \left[\arccos\left(\hat{\mathbf{r}}_i \cdot \hat{\mathbf{r}}_{i+1}\right) \cdot \frac{180^\circ}{\pi}\right] \cdot sgn\left(\hat{\mathbf{r}}_{i,x} \cdot \hat{\mathbf{r}}_{i+1,y} - \hat{\mathbf{r}}_{i,y} \cdot \hat{\mathbf{r}}_{i+1,x}\right) \tag{2.7.5}$$

$$p = p_l + p_r \tag{2.7.6}$$

The notation $\hat{\mathbf{v}}$ indicates that the vector is normalized, resulting in simplified expressions for equation 2.7.4 and 2.7.5. The $sgn\left(\ldots\right)$ term is introduced to give a differentiation between left and right turns, resulting in $p > 0$ denoting a left turn and $p < 0$ denoting a right turn.

Based on p, the track was divided into segments belonging to one of six possible classes. Each class had a speed parameter associated with it, dictating how fast the car should drive when going through a segment of that particular class. They employed the model and strategies in a simple controller, and were able to outperform COBOSTAR on several of the standard TORCS tracks in their experiments. However, they did not take into account the noisy sensors that were introduced in the competition in 2010. This resulted in the classification system breaking down, and an overall poor performance in the 2010 SCRC [33].

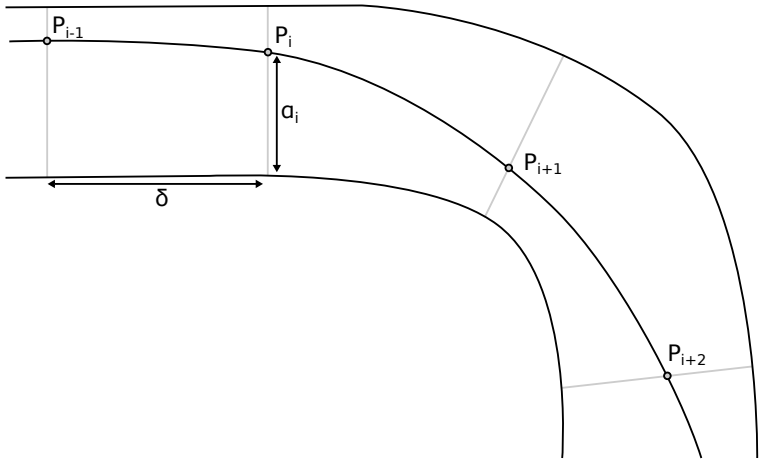


Figure 2.8: Racing line defined as a series of points $P_1 \dots P_n$.

2.7.3 Optimizing the racing line

If a model of a track is available, it is possible to search for the optimal racing line for a car. Braghin et al. [34] suggested optimizing a compromise between the minimum curvature path (MCP) and the shortest path (SP). The car's dynamics affect its ability to accelerate, brake and handle turns, and they proposed an algorithm that, taking such dynamics into account, would identify the optimal path and speed profile. Cardamone et al. [35] adapted and extended this method to the TORCS environment. The method described relies on perfect knowledge of the track layout, and their experiments and results were based on extracting this information from the files used by TORCS to store tracks. It allowed them to accurately calculate both MCP and SP of any track in the simulator.

A racing line is defined as a series of points $P_1, \dots, P_n$ along the racing track, with the last point P_n connecting to the point P_1 to complete the circuit. Assuming the points are equally spaced along the track by some distance δ (see Fig. 2.8), the whole racing line can be described by the vector $\boldsymbol{\alpha} = \langle \alpha_1, \dots, \alpha_n \rangle$ where α_i describes the distance of the point P_i from the edge of the track. With both the MCP and the SP described using such vectors, the goal of the experiments was to optimize a combination of these that was better than both. The first experiment assumed that the optimal racing line could be described as a linear combination of the two (first proposed in [34]), as described in Equation 2.7.7, using the parameter ε as a weight.

$$\boldsymbol{\alpha} = (1 - \varepsilon) \cdot \boldsymbol{\alpha}_{MCP} + \varepsilon \cdot \boldsymbol{\alpha}_{SP}, 0 \leq \varepsilon \leq 1 \tag{2.7.7}$$

A grid search where lap times were evaluated with ε ranging from 0 to 1 with step size 0.01 on 11 tracks showed that the best ε was in all but two cases 0 or very

close to 0. This means that the optimal racing line was very close to $\boldsymbol{\alpha}_{MCP}$, which was consistent with the findings of Braghin et al.[34].

The method was then extended to use $\boldsymbol{\varepsilon}$, a vector of weights. The track was divided into segments wherever the MCP and the SP crossed, and one weight ε_i was associated with each such segment. A genetic algorithm was used to optimize $\boldsymbol{\varepsilon}$ so that the weighting of the MCP and the SP was locally optimal in every segment of the track. The results showed an improvement in lap time on all 11 tracks over the MCP racing line, with all but one track having an improvement of $0.5 - 1.3$ seconds. Given that this controller used information not available in the SCRC by accessing TORCS track files directly, it is not directly comparable to any SCRC submission.

Chapter 3

Methods and design

This chapter describes the detailed design of the CRABCAR system. A general overview of the system given, before going more in depth on the key components and methods.

3.1 CRABCAR overview

The CRABCAR system is implemented in Java, and uses the networking framework provided by the SCRC organizing committee. A slightly modified[1] version of the Watchmaker framework [36] is used to perform evolutionary computation. The database that is used for persistant storage of track models and strategies uses SQLite.

CRABCAR is designed around a behavior-based architecture similar to motor schemas [24], illustrated in Fig. 2.6. Several modules with different functionality all work together to produce the final behavior of the system. The modules all receive all or some of the sensory information available from the simulator, and have the ability to override the default behavior given that certain conditions are met. This allows behaviors like crash recovery and opponent avoidance to be separated from the core driving behavior of the system, only taking control when necessary. A description of the architecture and the workings of the different modules, including crash recovery and opponent avoidance, is presented in Section 3.2.

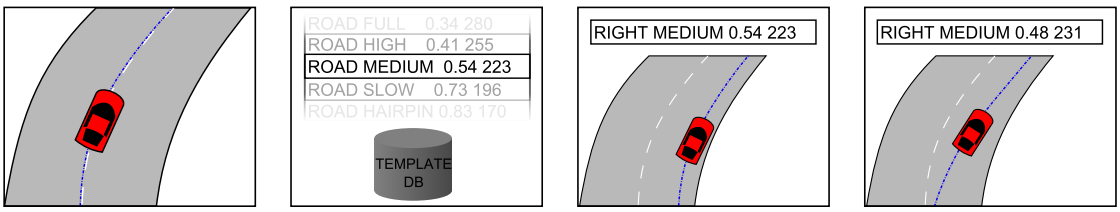
[1] Functionality was added to extract more information concerning fitness statistics of populations of individuals during evolution. The unmodified version provides only maximum and average fitness information for a population.

With the SCRC introducing the warm-up stage of the competition in 2010, the possibilites for learning the track layout, building an abstract track model and using it to plan ahead greatly increased. CRABCAR tries to build such a track model during the warm-up stage. After classifying the turns on the track, it can benefit from strategies learned offline prior to the competition. The method used to build the track model is an extension of the approach proposed by Quadflieg et al. [2], described in detail in Section 3.4. The main caveat of their approach is that it does not handle noisy sensors well. This became apparent in the 2010 SCRC when they had performance difficulties due to wrongly classifying turns [33]. CRABCAR introduces filters that work both on the noisy sensory input and the proposed classifications produced during the warm-up stage of the competition.

When discretely classifying turns and applying strategies concerning speed and positioning, the main weakness is that the strategies have to be optimized on the slowest and tightest turns of every class. If not, the car risks entering a turn too fast, going off-road and crashing. The car can only take a certain amount of damage during the SCRC before it is disqualified. This means that in many cases, depending on the granularity of the classification, the car will drive much slower through a turn than what is actually possible, losing important time in the competition context. Quadflieg et al. [2] optimized the speed at which the car entered turns for their six classes. CRABCAR extends the strategies by also optimizing the car's position on the track when entering a turn, affecting the racing line. This is inspired by the approach used by Cardamone et al. [35] to describe an optimal racing line, maximizing the speed at which the car can complete a track. The optimization process of strategies is described in Section 3.5.

To address the problem of underperforming when using strategies optimized for the slower and tighter turns of each turn class, an adaptation system with roots in reinforcement learning is applied to further optimize the strategies. It starts working as soon as a track has been classified online, and does so on an individual turn basis. This way, turns that are originally found to be of the same class can end up using different strategies that are more suited to their individual properties. These new strategies are kept in the track model and used throughout the later stages of the competition.

Fig. 3.1 illustrates the major steps in the process used by CRABCAR during the warm-up stage. First building a track model, then employing strategies that have been learned offline previously, beofre adapting the strategies on an individual turn basis. After the warm-up stage, very little is done to the strategies and track model. In the second and third stage of the competition, every second counts. As the the car is potentially driving at the limit of its abilites after the warm-up stage, further modification of the strategies has a high likelihood of sending the car off-track and ruining its chances of qualifying for and winning the competition.



(a) (b) (c) (d)

Figure 3.1: CRABCAR during the warm-up stage of the competition. This is the phase where classification, the use of offline learning and most of the online learning happens. (a) The first lap, where the car drives drives safely around on the track, positioning itself in the middle of the road, and utilizes range sensors to sample the curvature for every part of the track. (b) Directly after the first lap, curvature samples are used to create discretized turns, which are are initialized with prelearned strategies fetched from a database. (c, d) In the following laps, adaptivity is used to optimize the turn strategies on individual basis.

3.2 System architecture and module functionality

3.2.1 The architecture

The situations a racing car has to deal with are many and varied. Designing a controller capable of handling them all in a satisfactory manner is a complex task. The BBR architecture (see Section 2.6) divides the functionality into logical units that have different responsibilities and affect actions only when it is called for. In the racing car setting we have identified four different behaviors that are the reason behind the architecture design. Fig. 3.2 shows an overview of the architecture.

The main module is the drive module, responsible for keeping the car on the track and at the highest speed allowed at all times. The warm-up module is responsible for controlling the car during the first lap of the warm-up stage of a competition, keeping the car at low speeds and as centered in the track as possible to create the best possible conditions for building the track model (see Section 3.4). The stuck module takes control of the car any time it ends up off-track, and the opponent module decides the best course of action in scenarios where opponent cars are nearby. By default, the drive module is controlling the car entirely. Based on sensory input, one of the other modules may override some or all of the decisions made by the drive module.

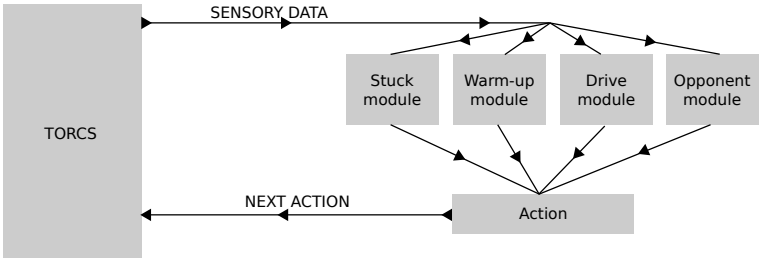


Figure 3.2: CRABCAR architecture.

3.2.2 Warm-up module

The warm-up module is designed to be used in the first lap of the first stage in a competition. This is the one lap where the track is completely unknown to the controller (i.e. no model of the track exists yet). During this lap the classification of turns and track modelling happens, the driving forces behind this module's behavior. The warm-up module is in complete control of the car for the duration of the entire first lap.

The primary objective of the warm-up module is to facilitate turn classification. Due to the noise present in the sensors that are used for this, the results of the classification are best when the car drives as close to the center of the track as possible, i.e. having equal distances to both sides of the track. The classification process also benefits from smooth movements, avoiding sudden changes in the car's direction and velocity.

Making a controller that keeps the car in the center of the track for a lap is trivial. The challenge comes from the time limit imposed on the warm-up stage of the competition: the track model must be generated as accurately as possible while still leaving as much time as possible for adaption of the car's behavior to optimize performance on that particular track after the first lap. To compromise, the warm-up module does rudimentary analysis of the sensor data and sets its speed accordingly. This means that if the stretch of track ahead of it is straight, it will accelerate to high speed. If it detects an approaching turn, it will slow down and try to match its speed to the strength of the turn, all the while steering towards the center of the track. The changes in speed fit well with the sampling of sensory data for the turn classification process, which is happening concurrently. Since the sampling of sensory data is time-based, the slower speeds effectively provide higher resolution in the sensor data collected in turns, and lower resolution when driving fast on the straight sections that need no classification.

The result of this compromise is a car that navigates the track relatively quickly, but still defensively. To be on the safe side, it never takes any chances and rather

drives slower than it needs to to ensure that it makes it around any track without going off-track. Acceleration and braking are done gradually to avoid sudden changes in the car's velocity.

3.2.3 Drive module

The drive module is the main controller of the car, being responsible for controlling it at all times except when overridden under special circumstances by the other modules. Given a track model its purpose is to drive the car as fast as possible, keeping it on the road by adhering to the limitations imposed by the strategies associated with turns on the track. This is the most complex controller.

The module is designed to rely on a track model which is provided when the warm-up module is done and hands the control of the car back over to the drive module. The track model describes where turns are on the track, and their associated strategies. The drive module's task is to make sure the car lines up ahead of turns and adjust its speed according to the information in the track model.

On straight sections of the track, i.e. sections that are not classified as turns and where the car is not close to an upcoming turn, the car accelerates as much as possible while aligning with the track direction. If a turn is preceded by a straight section of track, the module will accelerate until the point where it must start to brake in order to reach the target speed for the coming turn.

Throughout the turn, the car is kept at constant speed, namely the speed dictated by the strategy associated with the turn. Steering is done by heading in the direction of the distance sensor reading the highest value, an example being sensor 11 in Fig. 2.7a. When approaching a turn, just before the car needs to brake, the drive module aligns the car in the position dictated by the associated strategy of the upcoming turn. In right-hand turns, the car will enter the turn somewhere to the left of the center of the track, and vice versa. Combined with the trivial steering scheme which directs the car towards the inside of the turn, this increases the radius of the car's racing line through the turn, thus making the car able to take complete it at higher speeds compared to hugging the inside of the turn throughout.

Racing tracks often have more complex layouts than a straight section followed by a turn, followed by a straight section, followed by a turn etc. Chicanes are sections of the track where two or more turns are chained together directly after each other, often designed to force cars to slow down after long straight sections. The turns are often short and relatively sharp. In some instances, a turn is followed by a sharper turn, in which case the car cannot drive through the first turn at the speed

it would do if the turn was followed by a straight section. The track model allows CRABCAR to look beyond the next turn and detect such cases.

3.2.4 Stuck module

Rather than adding the complexity of being able to handle crashes and incidents where the car drives off the track and gets stuck to the warm-up module and the drive module, we opted to separate it into its own module whose sole responsibility it is to get the car back on the track and facing in the correct direction as fast as possible. Firstly, it has to detect when it should take control of the car. This is done by continuously monitoring the sensory input concerning the car's position relative to the center of the track, the angle it is facing compared to the direction of the track, and its speed. We define a stuck state for the car. By default it is NOT_STUCK, meaning that the sensory data shows no indication that the car is hindered in any way, and the stuck module does not override the drive module.

The most common way of being stuck is where the car is facing close to perpendicularly into a wall and the only option is to reverse. When this happens, the car's stuck state is changed to FRONT_STUCK, meaning that the car is hindered by something in front of it. Using the names of the sensors as defined in Table 2.1, the conditions for being FRONT_STUCK are low speed, defined as $speedX < 1\,km/h$ *and* one or both of the following:

- being nearly or completely off the track, defined as $\|trackpos\| \geq 0.8$ *or*
- high absolute value for the angle between the car's direction and the track direction, defined as $angle > \pi/8$.

If such a situation persists for 50 game ticks, about 1 s of simulated time, the car is considered FRONT_STUCK and the stuck module assumes total control of it. The default behavior is then to reverse while turning until the car's direction lines up with the track's direction. It then brakes, steers towards the track and accelerates. Once the car is back on the track the control is given to either the drive module or the warm-up module, depending on which was in control before the stuck module took over. The routine of reversing and driving back on track is illustrated in Fig. 3.3.

Another situation can happen if the walls enclosing the track are close to the edge of the track itself. If the car at some point is considered FRONT_STUCK and it is reversing to get back on the track, the car may reverse into the wall on the other side of the track, in which case further reversing will get it nowhere. The car is

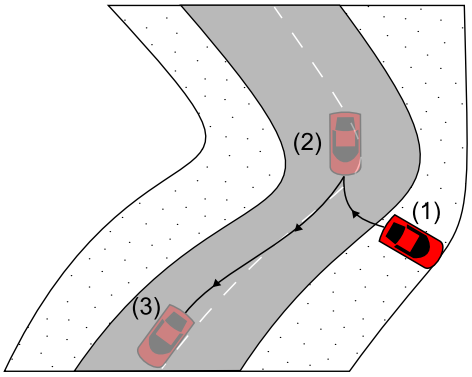


Figure 3.3: Illutration of the car being FRONT_STUCK. After a second in state (1), where the car has driven off track and crashed into a wall, the stuck module takes control of the car. It then reverses while turning to get aligned on the track as shown in state (2). Control is handed over to the drive module, which resumes normal driving, see state (3).

now BACK_STUCK, which is detected in the same way as FRONT_STUCK, the only difference being that the car can only become BACK_STUCK if it is already in the FRONT_STUCK state. To get back on track, the car will attempt to drive forwards and turn in the direction of the track driving direction.

3.2.5 Opponent module

The opponent module is responsible for avoiding opponent cars. It utilizes the opponent sensors, which return distances to opponents at different angles with regard to the car's axis. There are 36 such sensors in total, each covering a span of 10°, and returning the distance to the car nearest in their respective sectors within a range of 200m. Only 18 of these sensors are utilized by our opponent module, namely the ones pointing forward and directly to the side (i.e. the sensors covering the angles $[-90°, 90°]$ with regard to the car axis). As such, CRABCAR does not monitor the opponents behind the cars, but focuses on avoiding those in front and to the sides of the it. The reason no particular attention is paid to cars not ahead of us, is that the drive module is then utilized to drive as fast as possible, indirectly leading to the car avoiding cars coming from behind using speed.

The 18 opponent sensors used are separated into two categories, *nearbyLeft* and *nearbyRight*, containing sensors in the range $[-90°, 0]$ and $[0, 90°]$, respectively. If an opponent is sensed to be only in *nearbyLeft*, the opponent module suggests that our controller place itself to the right in the track, to avoid bumping into the opponent as well as hopefully getting enough room to overtake the opponent. The

same is done when opponents are only sensed in *nearbyRight*, but searching for room on the left side. If opponents are found to be in both categories, it can mean two things. Either multiple cars are registred by the sensors, or a car is directly in front of us. In both cases, the following suggestion from the opponent module is to steer towards the opposite site of where our controller is placed on the track. This effectively scans for openings where no opponents are straight ahead of the car, or only on one side.

It should be noted that opponent handling has not been a priority in this work, which is reflected in the simple opponent handling module. More advanced behavior, such as blocking opponents coming from behind and planning for overtaking on the inside, is saved for future work.

3.2.6 Shifting gears, clutching and braking

The car has a total of 7 gears, including reverse. An important component of driving as fast as possible is to make timely gear changes. By gearing up too quickly, acceleration is inhibited in two ways: the engine does not exploit the higher end of the RPM (revolutions per minute) register where it is at its strongest, as well as starting out in the lower RPM register on the higher gear, where it is weak. Gearing up too slowly also inhibits acceleration. There is an upper limit to the RPM an engine can do. If it does not shift gears, its speed will be limited by this upper limit and the ratio of the gear it is currently in. For instance, it is impossible for CRABCAR to drive faster than 278 km/h in fifth gear on a flat track. To drive faster, it is necessary to shift gears up to sixth. This enables it to exceed speeds of 300 km/h.

The gear shifting scheme for CRABAR is simple: for every forward gear, one through five, there is a number. If the engine's RPM reaches this number, the car shifts gear up to the next. Similarly, for gears two through six, there is a lower bounding number. If the engine's RPM dips to this number, the car shifts gears down. The reverse gear is only used in special circumstances by the stuck module (see Section 3.2.4), and neutral is never used.

The lower bound for gears is derived from the built-in automatic gearbox in the TORCS simulator that CRABCAR is running in. These numbers are not as crucial as the upper bounds, as the car mainly shifts gear down when braking to slow down before a corner, in which case it uses the clutch to brake with the brakes and not the engine.

The set of numbers used when gearing up are optimized to provide the best acceleration possible (see Section 3.5.2). Especially in the start of a race, where the

car starts from a standstill, these gear shifts are crucial to keep up with and get ahead of opponents. Another thing that is very important at the start of the race is the use of clutching.

If the clutch is not carefully used at the start of a race, the car will spin for several seconds while moving slowly forwards before gaining enough traction to accelerate properly. The power of the engine in first gear results in this suboptimal start where precious seconds are wasted, allowing the opponents to take an easy lead. It is therefore necessary to use clutching when starting from a standstill, allowing the car to put as much power as possible to the wheels without them losing traction. The clutching scheme used is inspired by Butz et al. [27], which applies alot of clutch from the start of the race, and slowly decreases it as the car accelerates in while still in first gear. As the car gains speed, the clutch is decreased faster until no clutch is applied.

Effective braking allows the car to remain at higher speeds for longer, resulting in an increase in average speed. In the case of an upcoming turn, the car should not brake until it is absolutely necessary to reach its target speed for that turn. The naive approach of letting off the gas and braking as hard as possible results in the tires locking up and the car skidding for as long as it takes to bring it down to the target speed.

The laws of physics dictate that the force needed to get an object moving from a standstill position is greater than that which is required to keep the object moving. This applies to the tires when braking as well. If a tire rotating fast enough to keep up with the car's speed, so that each "patch" of tire is static in relation to the ground surface it comes in contact with, the traction is much greater than if a car tire is locked up and sliding across the ground surface. To exploit this using brakes, one has to perform a pumping action where brakes are first applied, then let off to allow the tires to regain traction and rotate, then applying the brakes again and repeating. The automation of this act exists as ABS (anti-lock braking system) in modern cars. On most surfaces and for most drivers, ABS provides superior braking distances and vehicle control while braking. CRABCAR uses a simple ABS scheme, repeatedly letting off the brakes as soon as the wheels lock up under braking and waiting a small amount of time before applying them again.

3.3 The database

The three stages the competition is comprised of are three discrete events. If one wants to carry information (learnt behavior, track information etc.) over from one stage to another, or even use information acquired before the competition, persistent storage is needed. CRABCAR is dependent on both the general strategies

learned ahead of the competition, as well as the track information it gains during the warm-up stage throughout the rest of the competition. CRABCAR uses a database to store both its template strategies learned using evolution (see Section 3.5) and the track information with updated strategies that is used in the competition. When it classifies turns in the warm-up stage, it uses the classifications to build a track model by extracting matching template strategies from the database. This track model is then saved to the database for further manipulation. In the qualifying and racing stage of the competition, the track model is loaded in its entirety from the database, carrying over all the information learned over the course of the previous stages.

3.4 Building a track model

The championship CRABCAR will be entered in contains a warm-up stage which takes place prior to the actual competition. In this ~30 minute time slot, the controller can gather knowledge about the unknown racing track to be better prepared before racing against the clock and eventually other opponents in the subsequent stages. If the controller is able to use this time to create some internal model of the track, it can later use this model to make decisions revolving around general driving and planning ahead.

When nearing a turn, a completely reactive controller (i.e. a controller without any internal model of its environment) will, due to only having access to limited, local sensory information, have no basis of knowing the length or strength of the turn, nor if it is directly followed by another turn. This makes it difficult to know what speed the controller should aim for, as well as how it should place itself in the track, to be able to traverse the turn in an efficient manner. If, on the other hand, a model of the track containing this information is available, it can easily be exploited in making such decisions. This serves as the driving force for wanting to classify turns and storing them in a reusable track model.

3.4.1 What is being classified

For each track, we want to classify the following:

TYPE The terrain type of the track.

LENGTH The length of the track in meters.

TURNS A list of of the track's turns.

As the goal of the controller is to navigate a track as fast as possible, we wish to find and label those segments of the track where it can not drive at maximum speed, which basically means finding and classifying turns. An exception to this is when a track has uneven terrain, which can make it difficult to drive at high speeds on relatively straight segments. Such special cases are not found directly through classification, but is entirely handled by adaptivity (see Section 3.6.1).

For each turn, the following is classified:

START Where on the track the start of the turn is located, in meters from the start line.

DIRECTION Whether it is a left or a right turn.

LENGTH The length of the turn in meters.

TYPE The type of turn, indicating how fast it can be driven.

Based on the TURN TYPE and TRACK TYPE, parameters regarding how fast a turn can be driven and where the car should be positioned on the track prior to entering the turn are set based on evolved strategies. This is explained in Section 3.5.3.

3.4.2 Classifying a track

All classification is done during and directly after the first lap of the warm-up stage, when the warm-up module (see Section 3.2.2) is safely and reactively navigating around the track.

Finding track length

While the length of the track is not something provided by the sensors (see Table 2.1), the *distFromStart* sensor (giving distance to the start line in backwards direction) is available. At the start of a race, the driver always starts a short distance behind the start line, so the *distFromStart* sensor gives a value close to the track length. To get the exact length, this initial value is first temporally stored. The sensor is then monitored as the race starts, and as soon as it returns 0 (i.e. the driver has passed the starting line), the exact TRACK LENGTH is calculated by summing together the previously stored value with the latest *distRaced* sensor (which returns the distance covered by the car so far in the race).

TURN TYPE	Requirement
Not classified as a turn	$strength \leq 15°$
HIGH	$15° < strength \leq 20°$
MEDIUM	$20° < strength \leq 30°$
SLOW	$30° < strength \leq 45°$
HAIRPIN	$strength > 45°$

Table 3.1: Distinct turn types. All turns are classified into one of these categories.

Sampling and classifying turns

To classify turns we use method extending a technique introduced by Quadflieg et. al. [2] (described in detail in Section 2.7.2), which tries to determine an angle based measure for the curvature of a turn. The curvature measure p is calculated for each game tick throughout the first lap of the warm-up stage, and a sample for each meter is stored in a list the size of the track length. Directly after the first lap, this list is used as the basis for classifying turns into distinct types. List segments with a consecutive p-values above $10°$ or below $-10°$ are used to create turn objects with DIRECTION left and right, respectively. Turn START and LENGTH are derived from the list index of the first and last p-value in such a segment. A temporary parameter *strength* is, for each turn, calculated as the absolute value of the average p-value for that turn. This is used to classify the turn into one of the distinct types denoted in table 3.1.

Determining track type

Classifying the track type is done directly after the first lap by doing a simple braking test. The distance covered when braking maximally from 100 km/h to 0 km/h is measured, and if the distance is over 20 meters, the track is classified as DIRT. Otherwise it is classified as ROAD. The track type has direct influence on factors such as braking distance and the skidding threshold in turns, which is why the strategies applied to turns after classification are tuned against one type of surface (see Section 3.5.3).

Noise handling

During the competition races, the track edge sensors used to classify turns will be affected by a normal distributed noise with a standard deviation equal to 10% of the sensor range. As the classification method relies on fairly accurate sensor

information, the noise will, if not dealt with, make it practially useless (illustrated by the performance of the controller MrRacer in [33]). To handle the noise, we introduce *sensor smoothing, turn length filtering* and *turn combination.*

Instead of relying only on the latest sensor information when calculating the curvature measure p, a linear smoothing filter is applied to the sensor readings. The data from the latest 100 track edge sensors are stored, and when calculating p, the average of each sensor over these 100 game ticks are given as input. The idea behind this is to avoid one very noisy sensor sample to have a very big impact on the measure, and, as the noise is 0-mean normally distributed, hopefully "even out" the noise to give a good approximation to the actual track edge distances. As a game tick is 20 ms, it takes 2 s to "refresh" sensor samples, i.e. replace all the old values with new values. This results in a small offset with regard to where on the track a turn is perceived as being located, but this offset is taken into account when CRABCAR the car's location on the track against the track model.

Although *sensor smoothing* tries to approximate correct sensor values, there is still a possiblity for faulty turn classification to occur. This can mean classifying a segment faulty as a turn, or failing to correctly classify a turn, for example by classifying it as two turns with a small straight segment between.

Turn combination is applied after classifying all the turns in a track, to see if two consecutive turns should in fact have been classified as one. This is done by iterating through the list of classified turns, and if two turns of the same direction are divided by only a very short distance, they are replaced by a single new turn. The new turn is eligible for further recombination.

If faulty sensor values lead to classifying a turn which does not exist, there is intuitively a good chance that it is classified with a very short LENGTH. If, for example, a sensor sample indicates that there is a right turn ahead when there is not, the subsequent samples will probably not agree. So, a *turn length filter* is applied to the list of classified turns, which simply removes the turns with a LENGTH less than a certain threshold. By running simple classification experiments without noise on different tracks, it was found that real turns seldom or never are classified to be shorter than 20 m, making it the threshold for what is considered a turn by the track model building process.

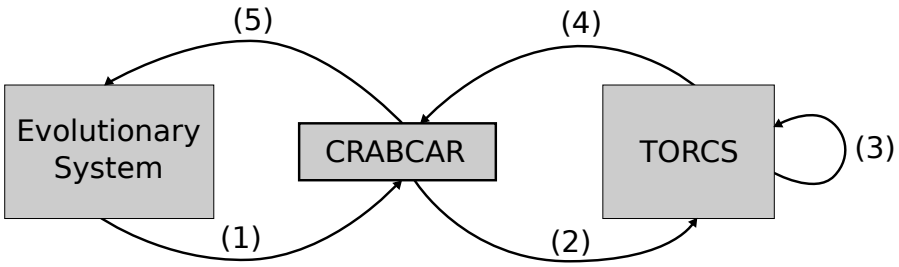


Figure 3.4: Evolutionary system using TORCS for fitness evaluation. (1) The parameters representing an individual is applied where necessary in the CRABCAR system, (2) CRABCAR connects to TORCS, (3) TORCS runs a simulation for a given amount of time, (4) TORCS returns data from the simulation run to CRABCAR, (5) CRABCAR returns the fitness value (calculated from the simulation data) of the individual back to the evolutionary system.

3.5 Evolution of strategies

3.5.1 TORCS and evolution

By eliminating the visual user interface of TORCS, the simulation speed can be up to 20 times faster than real-time. It is this speed-up that makes the use of any evolutionary computation system a feasible approach to optimization problems that require the simulation running. Such a system was designed to optimize the gearing scheme and strategies used by CRABCAR. An overview of the system setup is illustrated in Fig. 3.4.

3.5.2 Evolution of gear changes

The basic mechanic of changing gears is important to maximize the speed of a car. Specifically, the timing of shifting gears up can have big consequences for the outcome of a race. Acceleration can be limited by gear shifts that happen both too early and too late. Both the power (horsepower) and torque of an engine varies with its revolutions per minute (RPM). Normally it produces less power in the lower RPM spectrum, achieving maximum power at several thousand RPM, before dropping a little in power towards the very highest end of the spectrum. With any gear shift (up), the engine's RPM will drop since the new gear ratio requires fewer RPM to keep the driving wheels turning at the same speed. If a gear shift happens too early, this RPM drop will cause the engine to run in its lower and possibly weaker end of the RPM spectrum. This means that in addition

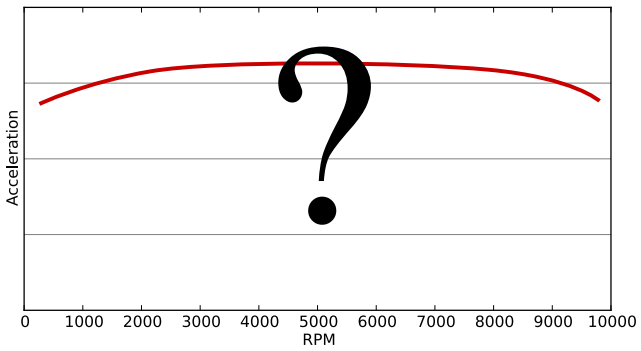


Figure 3.5: Acceleration as a function of RPM is unknown.

to not exploiting the previous gear fully in the stronger part of the engine's RPM spectrum, it will now have to struggle for longer in the new gear.

A gear shift can also happen too late. An engine is limited in the number of RPM it can do. If it at any point remains in one gear for too long, i.e. maximizing the engine's RPM and then not shifting up, the speed is limited by the ratio of that gear and the engine's maximum number of RPM. Acceleration will effectively stop.

All the bots in the competition use the same type of car, one which comes with TORCS. Its engine is modelled realistically to some degree (see Section 2.4), but the exact specifications of it are not available. The RPM-acceleration function, illustrated in Fig. 3.5, is unknown for all gears, so the problem of optimizing gear shifts is solved by treating the car and its engine as a black box system. A set of input parameters dictate when the engine should shift gears, which results in some length of track being covered in a given amount of time. The output result from TORCS is this distance.

The input parameters for this black box system can then be optimized to produce the longest possible distance covered in the given time, effectively optimizing the car's acceleration through its gear shifting scheme. Although a number of search algorithms can solve this problem, an evolutionary algorithm (EA) was used for two reasons. Firstly, the problem is structured in a way where an EA can easily be applied. Individuals can be defined as some permutation of the gear parameters, and the fitness function of an individual has already presented itself as the distance covered in a given time. Secondly, a EA system built to solve more complex problems had already been implemented. This easier problem served as a test for the EA system built around TORCS.

3.5.3 Developing turn strategies prior to the competition

The structure of the competition is such that competitors only have ~30 minutes on the unknown track in which they do not have to care about the speed or efficiency of their driving. When the first stage is over, the real competition is on and mistakes are punished. This first stage is intended to let the cars analyze the track and optimize their behavior for it, before they attempt to qualify for the real race.

Considering the relatively short period of time the controller has available in this first stage of the competition, it is not unreasonable to assume that having a good and general starting point in behavior can benefit the outcome of the competition. This baseline behavior can then be improved on throughout the first stage of the competition, making the best use of the limited time available. CRABCAR's turn classification and track modelling capabilities enables it to benefit from knowledge acquired before the competition. This knowledge, manifested as strategies on how to handle specific types of turns, is used from the second lap and onwards in the first stage of the competition. As soon as the first lap is done and the track model is built, CRABCAR can drive much faster around the track because of these strategies. However, they only serve as the starting point for CRABCAR's behavior post classification. Adaptivity, which affects behavior from the third lap and onwards, further iterates on the strategy of every individual turn in the track to optimize the car's performance more.

The inspiration for including not only speed, but also positioning in the strategies stems from the work descibed in Section 2.7.3. The idea is to optimize the the racing line locally within turns as well as the speed at which the car drives through the turns.

The idea is that when the car is first presented with an unknown track in the warm-up stage of the competition, it uses one lap to classify the track's turns and build a track model (see Section 3.4). When a turn has been classified to be of a certain type (e.g. MEDIUM) a strategy stored previous to the competition for turns of that particular type are loaded from a database. The car is able to drive quickly and safely by using strategy which is based on extensive experience acquired before the competition. The strategies do, however, only serve as a starting point for further performance optimization. After driving the second lap of the track using the strategies from the database, the car analyzes its behavior in each individual turn and adapts the strategy associated with that particular turn accordingly (see Section 3.6).

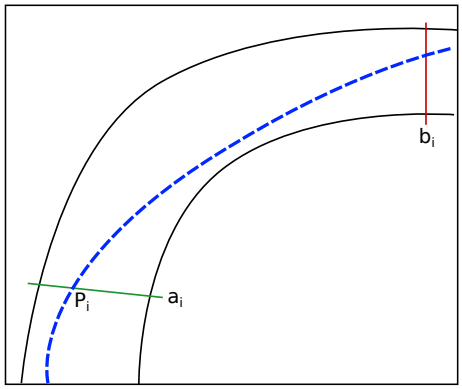


Figure 3.6: Example of a strategy in practice. In a newly built track model with many turns, turn i is defined by the starting point a_i and the end point b_i, and classified as MEDIUM. The strategy for a MEDIUM turn is applied, dictating the position P_i and a speed S_i that the car should have when reaching the starting point a_i. P_i is defined as a distance from the middle of the track, and the direction of the turn defines which side of the middle. When applied, the strategy enables the car to traverse the turn quickly and with no risk of driving off the track.

3.5.4 Optimizing strategies

We define a turn strategy as a tuple of real numbers, $\langle p, v \rangle$, denoting the entry position and entry speed, respectively. They are limited as follows:

- $0.0 \leq p \leq 1.0$
- $0.0 \leq v \leq 300.0$

p defines the position on the track the car should be in when entering a turn using the strategy, with the center of the track is $p = 0.0$ and the edge of the track is $p = 1.0$. The sign of p is determined by the direction of the turn the strategy is applied to, making it negative in RIGHT turns. v defines the speed in km/h at which the car should enter the turn when using the strategy.

The turn types that CRABCAR's classification use do not take into account turn length, track width or track slope. Because of this, the strategies stored prior to the competition that are used in the race have to be general for the type of turn they are optimized for. One MEDIUM turn can be a little sharper than another MEDIUM turn, and the strategy has to work for both of them. The result is that the limiting factor of the strategies are the longest, sharpest (i.e. slowest) turns of every turn type. Both these factors limit the speed by reducing the radius of the racing line possible through the turn.

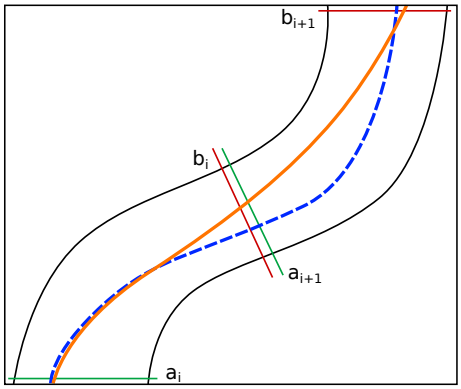


Figure 3.7: The potential advantage of double turn strategies. Two turns i and $i+1$, defined by the points $\langle a_i, b_i \rangle$ and $\langle a_{i+1}, b_{i+1} \rangle$, respectively, are very close to each other. Using a single turn strategy on both (illustrated by the blue dotted line) potentially results in the car aligning for the second turn when there is no reason to. The double turn strategy (illustrated by the orange line) bypasses this problem and imitates a straighter and likely closer to optimal racing line, which in turn enables the car to drive faster.

In an attempt to further extend the use of strategies, we propose the use of double turn strategies. These are strategies that are optimized for two turns with little or no distance inbetween them. In a tight chicane, it is sometimes possible to cut both corners and drive in an almost straight line through (see Fig. 3.7). With the use of double turn strategies, the hope is to optimize the racing line especially for cases like chicanes. By providing a better racing line than two single turn strategies would, the speed can be increased and time won. Double turn strategies are defined as a tuple of a tuple of parameters, $\langle \langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle \rangle$, corresponding to the parameters in two basic single turn strategies.

Basic single turn strategies involve all combinations of turn types and track surface types, for a total of eight strategies. The two parameters have to be optimized for them all individually, since the speed at which the car can manage e.g. a MEDIUM turn is higher on a track with the surface type ROAD than it is for a track with the surface type DIRT. Double strategies are evolved for all combinations of turns where the second turn is as slow as or slower than the first, and where the direction of the turns are different (e.g. a LEFT-RIGHT combination or vice versa). This makes for a total of 20 double turn strategies. The optimization of the basic and double turn strategies is described in Section 3.5.4

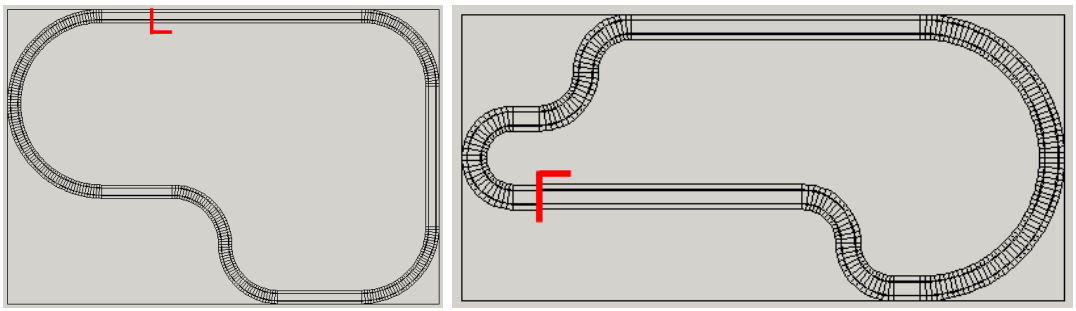
3.5.5 Using Evolutionary Programming to optimize the strategies

TORCS' physics engine allows it to simulate the complex dynamics of a car on a track, taking into account the weight of the car, the friction between tires and the track, and all the forces involved in the car taking high speed turns. Detailed knowledge of this physics engine could be used in a bottom-up approach to optimize the turn strategies used by CRABCAR. Instead, similar to the the approach used to optimize gear changes, a top-down approach using an evolutionary computation system was chosen. The approach is simple in that it allows us to treat the car as a black box, and thereby distancing us from the complex systems underlying the physical behavior of the car. By measuring performance using the distance the car travels in a given interval of time using a strategy, it is easy to compare different strategies of the same type to each other.

A set of tracks were custom built to facilitate the optimization procss. To make sure that every turn and track type combination was covered, one track for each comination was made by hand. This made it possible to optimize the strategies using only one track per strategy. The tracks are designed so that they contain turns of different length and curvature, all the while making sure that most of the turns classify as the turn type the track was made to optimize for. Two examples of tracks are shown in Fig. 3.8. The layouts of the tracks used to optimize DIRT strategies have the same layout as the corresponding ROAD tracks, producing comparable results in the final strategies evolved. As an example, the track used to optimize the parameter set for the turn type MEDIUM on a DIRT surface consists of a series of turns, each of which classifies as MEDIUM, in different combinations and with different lengths, all on a surface with traction qualifying as DIRT.

3.6 Adaptivity

While CRABCAR acquires a lot of knowledge during the first lap of the warm-up stage through track modelling (see Section 3.4) and prior to the competition through evolution of strategies (see Section 3.5), there is still optimization that can be done with regard to the handling of specific turns and other problematic parts of a track. This is where adaptivity comes into play. The idea is that when turns are divided into distinct classes during classification, they are only given (evolved) strategies based on good *general* handling of that type of turn. But two turns classified as the same type can be somewhat different (see Section 3.5.4), and



(a) (b)

Figure 3.8: Two of the track layouts used for fitness evaluation. (a) used to optimize single turn strategies for MEDIUM type turns, (b) used to optimize double turn strategies for the combination SLOW-SLOW. The direction of driving is denoted using the red line.

what we want from the adaptivity is to improve these strategies into good *specific* handling for each particular turn.

Adaptation can also offer some robustness to CRABCAR. Due to noisy sensors, there is a possibility that the classification will not classify all turns correctly, or even classify them at all (although this is attempted dealt with through noise handling, see Section 3.4.2). Also, a relatively straight part of the track, which should not be classified as a turn, can be difficult to maneuver in maximum speed due to rough, uneven terrain (which is the case on some tracks). Both of these cases could for example lead to CRABCAR always driving out at the same part of a track. To avoid this from happening, we want the controller to learn from its mistakes. This is done by employing reinforcement learning, where a track segment receieves a punishment in form of a speed decrease if it crashes or drives off the track.

3.6.1 How the adaptivity works

An adaptivity manager monitors and, if necessary, makes adjustments to CRABCAR's behavior. Its functionality can roughly be divided into two cases: (1) Adaptivity to increase turn speed and (2) Adaptivity to reduce turn speed and handle dangerous areas. The idea behind (1) is that we want to evaluate how well CRABCAR traverse a turn and, if the evaluation suggests that the turn could have been driven faster, tune the turn strategy's speed parameter to give it a slight speed increase. In case (2), we search to prevent from making the same mistakes twice, by having the controller store its mistake locations, i.e. where it drove off the

track, and take precautionary action the next time it approaches such a location.

3.6.2 Adaptivity to increase turn speed

Using the track model obtained through classification as a basis, CRABCAR utilizes adaptivity to try to optimize the parameters of each specific turn. To do this, the car's *track position* (i.e. positioning on the track) is sampled for each game tick throughtout each turn, and directly after the turn these samples are evaluated in search of possibilities for a speed increase.

To evaluate a turn traversal, the system iterates through the list of track positions to find when the car drifts furthest to the opposite side of the turn direction. In a right turn, this means finding the track position of the car when it is furthest to the left during the turn. If the track position found is at a distance more than $\frac{1}{4}$ of the track size from the track edge, the evaluation concludes that the turn could potentially have been driven faster, and the maximum speed for the turn is given a slight increase. If this is not the case, i.e. the car drifts near the edge of the track, the maximum speed for the turn stays the same. The reason for evaluating how well a turn is traversed by seeing how far to the side of the track the car drifts, is that if the car has too high speed in a turn, the result will most likely be that it drives off the track on the side opposite to the turn direction. If the evaluation is not a success, if it for example leads to a speed increase which makes the car crash, the method explained in the following section will reduce the speed and lock it so will not get increased again.

3.6.3 Adaptivity to reduce turn speed and handle dangerous areas

While CRABCAR is driving on a track, the adaption manager is at all times (except during the classification part of the warm-up stage) keeping a history of the state of the controller for the last 1000 game ticks. The state consist of different sensor values, which are *distance from start line*, *speed*, *lateral speed*, *angle* to *track axis*, and *track position* (i.e. alignment on the track). If a crash occurs, this history information can then be exploited to make necessary adjustments for preventing it from happening again in the future. The size of the history gives a hindsight of 20 seconds, which intuitively should always be enough to find when (and thus where) problems started to occur.

When we write "crash", it is not only meant in its literal sense, but more as a collective term for certain events that we wish to avoid. These events are as follows:

- The lateral speed of the car is very high, which means the car has likely partially or completely lost its traction: $\|speedY\| > 30$ (km/h).
- The car is not pointing in the direction of the track axis: $\|angle\| > \frac{\pi}{2}$.
- The car is (partially) located outside the track: $\|trackPos\| > 0.9$.

A regular crash in its literal sense (i.e. banging into a wall or similiar) is always covered by the mentioned events, as this would mean that at least parts of the car would be located outside the track.

If a crash occurs, i.e. one of the three requirements for a crash is fullfilled, the adaption manager uses the history to try to determine the cause for the crash and prevent it from happening again. It first marks the location where the crash occured as a crash point. Then it starts iterating backwards through the history of states (starting with the most recent state and moving back in time) to find a safe point - a point recognized by the car having very low lateral speed, and angle to track axis close to zero (i.e. the car is pointing in the same direction as the track axis). The idea behind finding such a safe point, is that the mistake done by the controller (which eventually lead to a crash) almost certainly occured between this point and the crash point. An example of a crash caused by driving too fast in a turn is depicted in Fig. 3.9.

What the adaption manager does next depends on whether or not there are any turns between these two points:

1. If there is a turn, or multiple turns, between the safe point and the crash point, the adaption manager assumes that it was the handling of these which lead to the crash, and reduce the maximum turn speed for the involved turns slightly. These turns are also labeled with a crash speed, which is set to be the maximum speed for the turn prior to the reduction. This parameter is used by the adaption manager to prevent it from later increasing the maximum turn speed back up to a unsafe threshold. The reason for potentially giving multiple turns a speed reduction, instead of only the last occuring turn, is that interconnected turns should to some degree be considered, and treated as, a single turn. This because crashing in such turn could be due to having too high speed in the preceeding turn, so only adjusting the speed of the last turn would not resolve the problem.
2. Not all crashes can be linked back to already classified turns. This can occur because not all turns always get classified perfectly, largely due to noise, and because there can be other responsible factors such as uneven or rough terrain. If there is not any turns found between the safe point and the crash

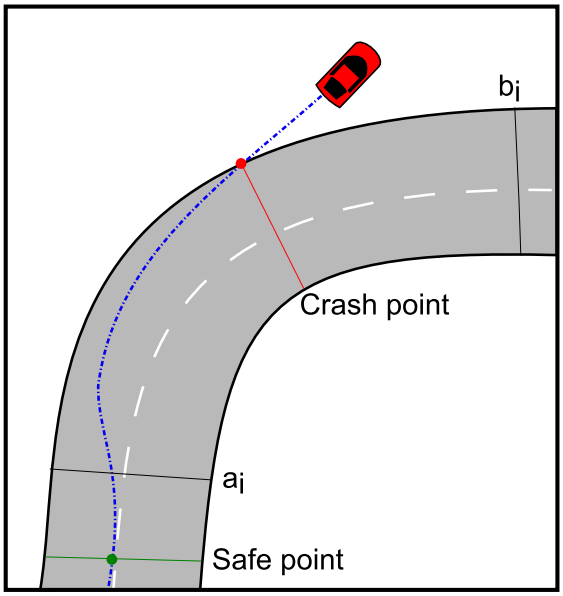


Figure 3.9: An example of driving off the track due to too high speed in a turn starting at a_i and ending at b_i. The crash point is stored, and the controller iterates back in through its history to find a safe point. Any turns between the safe point and the crash point, in this case only the turn $\langle a_i, b_i \rangle$, gets a reduction in SPEED to try to avoid such a crash from happening again.

point, the adaption manager marks this area as a dangerous area. This is done by creating something similar to a turn, but without any direction or entry position. The maximum speed for this "dummy turn" is set to 10% below the car speed that lead to the crash, and can be further optimized by adaption in subsequent laps.

3.6.4 Preparing for qualifying stage

After the warm-up stage, we are interested in using the track model information, modified through adaption, that lead to the highest performance, i.e. fastest driving. This is not necessarily always the track model used at the very end of the warm-up. As adaptivity tries to optimize the speed of how turns are traversed, it can find itself balancing on the edge of chaos. If a turn has a current SPEED nearing the maximal of how it can be driven, a speed increase can lead to a crash. If such occurs, the following decrease in how fast it should be driven might be lower than what it was before the increase. So, during the entire warm-up stage, the current track model information is sampled for each lap. The best configuration,

i.e. the one leading the the lowest lap time, is saved as a new track model, and is to be used in the following stages of the competition.

Chapter 4

Results and discussion

In the previous chapter we developed the CRABCAR system. In this chapter we put our implementation to the test, and explore how the different parts of the system perform, as well as the system as a whole. Four different experiment scenarios are investigated.

In the first scenario, we take a look at how the sensor noise affects our system in terms of performance. Scenario two focuses on how adapting prelearned strategies online can improve the overall performance of the controller. In the third scenario the use of evolution in offline learning is investigated, before we see how CRABCAR performs compared to other systems in scenario four.

4.1 The consequense of noisy sensors

The range sensors, which return the distance to the edge of the track at different angles, are affected by a normal distributed noise with a standard deviation equal to 10% of the sensor range. The classification method used to obtain a track model is dependent on fairly accurate sensor readings, which means that the noise has to be filtered in some way for the method to have any practical value. CRABCAR utilizes a linear smoothing filter technique, which averages each sensor value over the past 100 game ticks (2 seconds), before calculating the curvature of the track. After classifying the turns in a track, turn combination and length filtering is employed to try to reduce the occurence of incorrectly classified turns.

It is difficult to give good quantitative measure of how well the classification works in itself, as small variations might not have any effect on the performance of the driver utilizing the track model. Say for example that a turn gets classified as a MEDIUM turn when it should in fact have been a SLOW turn. This could potentially

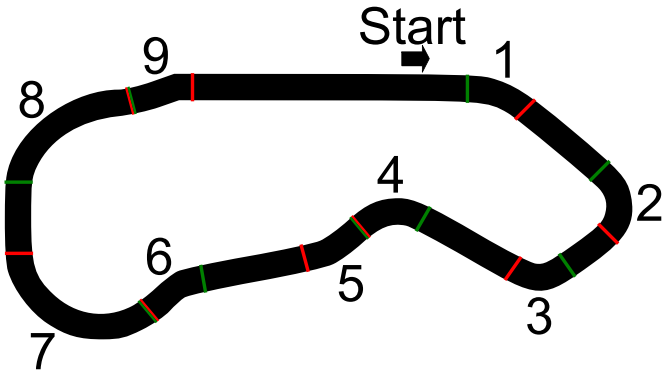


Figure 4.1: Illustration of the track Dirt5. Turns are added based on visual impression. The green and red lines denotes the start and end of a turn, respectively.

lead to the driver driving off the road in this turn, but this is then handled by adaptivity to keep it from happening again. So what we want from the track model is a good *basis* for further learning, and a "good" classification is therefore a model that eventually leads to the controller being able to drive fast and reliable.

4.1.1 Example of classifying with and without noise

An example of how classification can vary with and without noise enabled is shown in Table 4.1, and is based on the track Dirt5 (see Fig. 4.1). It is worth noting that many of the turns classified when noise is enabled has a START value 20-30 m higher than those classified without noise. This is related to the slight offset which is the result of smoothing each sensor value over a period of time, and CRABCAR compensates for this by taking precautions before turns earlier than it otherwise would. As can be seen in Table 4.1, turn 2 and 3 are classified somewhat differently in this example. In the case without noise, it has been classified as a single long turn, while it in the other case has been classified as two smaller turns. Such cases can occur, as well as other variations of misclassification, due to the noise handling not completely eliminating the effect of noise. This might have an impact on the peformance of the driver, which is tested by letting the controller drive the track with a classified track model as a basis.

Without noise					With noise				
START	LENGTH	DIR.	TYPE	TURN	START	LENGTH	DIR.	TYPE	TURN
38	101	R	MED.	1	62	90	R	MED.	1
159	166	R	SLOW	2,3	171	91	R	HAIR.	2
					283	58	R	SLOW	3
357	92	L	SLOW	4	365	91	L	SLOW	4
454	47	R	MED.	5	474	48	R	SLOW	5
530	69	L	MED.	6	561	41	L	SLOW	6
603	282	R	SLOW	7,8	626	283	R	SLOW	7,8
916	35	R	HIGH	9	935	33	R	MED.	9

Table 4.1: Classification of the track Dirt5 with and without noisy sensors. START and LENGTH values are in meters. The DIR. columt refers to direction, which can be either right or left, and the TURN column refers to which turn depicted in Fig. 4.1 the classified turn corresponds to.

4.1.2 Test setup

To test how noisy sensors affects the performance of our controller, and thus the performance of the classification, we compare how fast CRABCAR is able to drive using track models generated with and without noise activated. The warm-up stage of the SCRC comptition is simulated, which means the the controller has 100.000 game ticks (approximately 30 minutes) to classify and learn a track. Such runs are performed 20 times per track on 10 different tracks for each of the two cases, and the average of the best lap time from each run is used as a measure of performance.

4.1.3 Results and discussion

Table 4.2 shows the results of testing on different tracks. On eight out of the ten tracks, the average of the best lap time over 20 runs is slightly better in the case where sensors are not noisy, while two of the tracks (Forza and Ruudskogen) show a very slight advantage when noise is enabled. On four of the tracks, the runs with noise enabled have a lap time increase of over 1%, with 3.2% on Dirt5 being the highest. In average, noisy sensors gives approximately 1% increase in best lap time on our test tracks.

Interestingly, the results from these experiments show no direct correlation between length of the track and the difference between the two cases. The two longest tracks tested, Olethros and Alpine1, show a 0.6% and 1.9% increase in time when noise

Track	Length	Without noise	With noise	Difference	
				seconds	percent
Wheel1	4257.6 m	89.0 s	89.2 s	+0.2 s	+0.2%
Forza	5784.1 m	101.5 s	101.1 s	$-0.4\,s$	−0.4%
Olethros	6282.8 m	125,.1 s	125.9 s	$+0.8\,s$	+0.6%
E-track2	3147.5 m	94.9 s	96.5 s	$+1.6\,s$	+1.7%
Alpine1	6355.7 m	153.5 s	156.4 s	$+2.9\,s$	+1.9%
Alpine2	3773.6 m	111.7 s	112.2 s	$+0.5\,s$	+0.4%
Dirt4	3260.4 m	103.1 s	103.5 s	$+0.4\,s$	+0.4%
Dirt5	1072.9 m	41.2 s	42.5 s	$+1.3\,s$	+3.2%
Ruudskogen	3274.2 m	70.5 s	70.4 s	$-0.1\,s$	−0.1%
Aalborg	2587.5 m	105.3 s	106.4 s	$+1.1\,s$	+1.0%

Table 4.2: Average of best lap time over 20 runs with and without noisy sensors.

is applied, as well as the third longest track Forza which comes out better in the case of noise. As longer tracks correspond to more turns to be classified (this is at least the case with the tracks tested), one might intuitively think that noisiness would pose a bigger threat to such tracks. This does, according to these test results, not seem to be the case. A possible explanation for this could be that the longer tracks seem to have more turns in both directions, in form of chicanes and interconnected turns. A shorter track, such as Dirt5, has often most turns in one direction (imagine a small track circuit with only right turns), which gives fewer "sharp" edges for the classification to detect. When classifying the length of a right turn, for example, it is easier to find the exact end of a turn when sensor values suddenly get input of a left turn up ahead. When a turn is followed by a straight segment, or a turn in the same direction, the transition of sensor inputs is more gradual.

In the case of Forza and Ruudskogen, where the results improve when noise is enabled, the improvement is so small that it is most likely incidental. It is an indication of the classification, on this particular track, being equally as good as it is without noise. There is no reason to believe that classification with noisy sensors should ever be able to outperform classification without noise, as the noise filtering is only trying to approximate what the sensor value would be without noise. It is, however, a good sign with regard to the noise filtering that we in these cases seem to be able to classify as good with noise as without.

The results observed are impressive, and very encouraging. While the original classifier described by Quadflieg et al. [2] breaks down completely when noise is presented, introducing relatively simple noise filtering makes it able to produce

good track models which can be used as a basis for further learning.

4.2 Optimization of gear shifting scheme and turn strategies

Evolutionary programming is used to optimize both the gear shifting scheme and the turn strategies. Although mutation is usually the driving force in search by evolutionary programming, single-point crossover is also used as an additional evolutionary operator to provide diversity in reproduction. The selection strategy used is tournament selection.

4.2.1 Evolution of gear shifting scheme

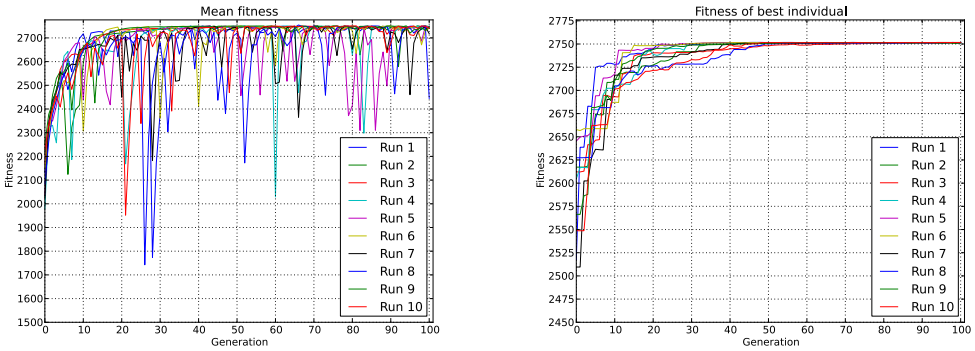
The gear shifting scheme has to be optimized to maximize the car's acceleration capabilities (see section 3.5.2). In practice, this means optimizing the parameters that dictate at which RPM the engine has to be at before the gear change can happen. There is one parameter R per gear change, making each individual a vector of six parameters $R_1, \ldots, R_6$ which are all integer values. Mutation of an individual involves adding a random value M to one or more of the parameters in an individual, where $M \sim N(0, 30^2)$, with probability $p_{mr} = 0.03$. Single-point crossover can switch 1-5 parameters between two parent individuals with probability $p_c = 0.6$, producing two new individuals. In addition, two parameters may be swapped within an individual with a probability $p_{ms} = 0.05$. The parameters are constrained such that

- $R_i \in [0, 10000]$ for all i

since the car engine's maximum RPM values are a little lower than 10000, depending on the gear it is in.

The initial population is randomly generated. Fitness evaluation is done by letting a car drive with its accelerator maxed on a straight track. The fitness value is the distance covered in 2000 game ticks, around 40 seconds of simulated time, with the parameters of the individual applied in the car's gear shifting routine.

Tournament selection is the selection strategy, and there are 2 elites in each population. The population of 50 is evolved over 100 generations.



(a) Average fitness (b) Best fitness

Figure 4.2: Gear shifting scheme evolution

4.2.2 Results and discussion

Figures 4.2a and 4.2b show the average and best fitness, respectively, of the ten evolutionary runs conducted to optimize the gear shifting scheme. The fitness of the best individuals converge in all ten runs after 50-75 generations, and they all converge to a value around 2751. Inspecting the best individuals from all the runs reveal that they have also converged on a parameter basis. differing only by a maximum of 30 RPM. This, in addition to the rapid convergence in fitness in all runs, lead us to believe that this is in fact the optimal gear shifting scheme to achieve as high acceleration as possible.

The plot showing the average fitness of the ten runs is also interesting. It reveals common occurences of sudden drops in the average fitness in all the ten runs. This can be explained by the parameters' upper RPM constraint being slightly higher than what is possible for the engine to produce. If, for example, the parameter concerning shifting gears from second to third gear is too high (i.e. outside the engine's natural RPM limits), the result is that the car is never able to get in higher than second gear. This severely limits the speed of the car, resulting in less distance covered during the fitness evaluation. The worst case scenario is not being able to shift into first gear, leaving the car stationary at the starting line of the race track and resulting in a distance of 0 m covered in the fitness evaluation. These cases explain the sudden drops in the average fitness of populations.

The results also reveal something about the nature of the RPM-power function (see Section 3.5.2) of the engine in the car. There seems to be a very small, if at all existant, distance between the optimal solution and the total failure that lies beyond it in terms of RPM. When approaching the optimum in the fitness

landscape from the lower RPM side, it is a steady climb. On the other side, however, there seems to be a sheer cliff. This tells us that it is, in fact, best to use the whole RPM spectrum of every gear, that the acceleration does not benefit from the engine shifting gears before it is at its RPM limit.

The experiments conducted were time-consuming with every evolutionary run taking around six hours to complete. They did, however, provide good results to be used in the competition.

4.2.3 Evolution of turn strategies

In the case of single turn strategies, individuals are represented by a tuple of real values $\langle p, v \rangle$, the former representing the entry position of the strategy and the latter representing the entry speed of the strategy. A population of 25 individuals evolves over 200 generations. The initial population is random. The fitness of an individual is the distance the car races over the duration of 3000 game ticks (around 1 minute of simulated time) when the values of the individual are applied as the strategy in relevant turns.

Specific tracks are used in the fitness evaluation. Each is designed with the purpose of providing a set of turns which represent an assortment, both in length and radius, of the type of turn the strategy is being optimized for. The same layout is used to optimize both DIRT and ROAD strategies, with only the surface being different. Because the track length varies with the different layouts, the fitness of different turn type strategies are not comparable.

Individuals are mutated by adding random values from gaussian distributions to either or both of the floating point values. In the case of the entry position, the random value is $T \sim N(0, 0.03^2)$. For the entry speed the random value is $U \sim N(0, 10.0^2)$.

Both entry position p and entry speed v are constrained when randomly generated or mutated so that

- $0.0 \leq p < 0.85$
- $0.0 < v \leq 285.0$

$p \geq 0.85$ is defined as the car being off-track. The difference in traction on and off-track is great, and having even one wheel off the track affects steering and/or speed control. While it may not punish the lap time in some cases, the risk is too big to let the car off-track willingly. If a best strategy were to evolve where $p \geq 0.85$, it

would likely be very specific to the track used in the strategy optimization process, and dangerous elsewhere.

The car can, on very, very long straight sections of track, achieve greater speeds than 285 km/h. This is not the case on the tracks that are used to optimize turn strategies. v is therefore limited to a value closer to the actual maximum speed attainable on the optimization tracks. If two strategies evolved where the first had $v = 289.3$ and the second had $v = 293.2$, they would both have the same fitness seeing as the car would never be able to accelerate to the speed v dictated by the strategies. If allowed, the strategies would be unreliable if the car ever encountered a track with a long straight before a turn where those strategies would be used. Entering the turn at a much higher speed than ever attempted before, it would risk crashing, losing precious time and potentially causing enough damage to be disqualified from the race.

The process of evolving double turn strategies is much the same as with single turn strategies. Parameters p_1, v_1 and p_2, v_2 are evolved, representing the entry positions and speeds of the two turn types encompassed by the strategy. Different tracks are used in the fitness evaluation process, designed to contain turn combinations corresponding to the strategy.

4.2.4 Strategy evolution discussion

Optimizing the strategies for single turns was unproblematic. For all turn types, the fitness converged after 30-100 generations. The relatively small search space, being only two-deimensional, produced similar optimal parameters in all cases. They were averaged to make the parameters of the final strategies used (see Table 4.3), ensuring that a highly specialized strategy would not bring unnecessary risks with it into the competition. Fig. 4.3 shows the three evolutionary runs used to find the optimal HIGH single turn strategy for ROAD type tracks. The average and best fitness of each run being so close to each other indicate that the populations became homogenous after a 30-80 generations. The driving force of the evolution was likely mutation. The plots of the average fitnesses have the same sudden drops in them as the ones in the gearing scheme optimization (see Figure 4.2a). The reason is the same: when all the individuals are producing strategies that push the car to its limits, one small mutation can cause the car to crash and the individual ending up a total failure.

Looking at best fitness plots, there are small variations in the fitness even after convergence. However, these are variations only in fitness, and not in strategy parameters. The elitism ensures that the best individual always survives to the next generation, normally resulting in a plateau in the fitness plot if that individual

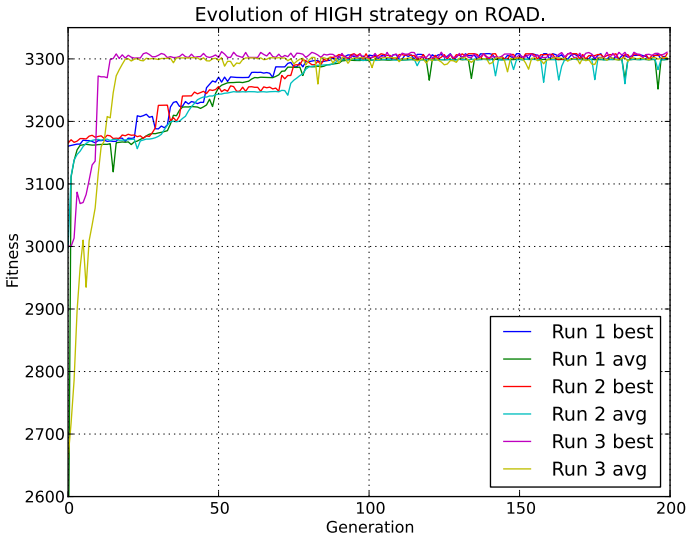


Figure 4.3: Evolution of the single turn strategy concerning the turn type HIGH on ROAD track type.

	p	v (km/h)
Best run 1	0.21	275
Best run 2	0.23	275
Best run 3	0.29	276
Final strategy	0.24	275

Table 4.3: Calculating a final single turn strategy.

were to be the best of the following generations. The cause of the discrepancies must be the fitness evaluation, which is done within the TORCS environment. If an action is performed a few milliseconds earlier or later than it was during a previous fitness evaluation, the result might differ. This is especially true for the optimization of driving quickly through turns. The optimal driving is pushing the car to its limits in terms of traction and the use of track width, meaning that small differences in behavior can have big consequences in the form of the car driving off-track or crashing. Either TORCS itself is non-deterministic or the communication that occurs between TORCS and CRABCAR is. Either way, the problem is not addressable without modifying the environment that is set up for the SCRC, which under the actual competition is out of the participants' control. This means that the use of extremely optimized strategies in the competition comes with great risks.

During the evolution of double turn strategies, the average fitness was consistently lower than the best fitness, indicating a heterogenous population (see Fig. 4.4). The much larger search space with the heterogenous population meant crossover could play a bigger part in maintaining diversity. To produce the final double turn strategy, parameters were taken from the best individual of the three produced from the three evolutionary runs. To avoid using extremely specialized double turn strategies, the speed parameters were tuned down by 1km/h.

The difference in the corresponding ROAD and DIRT strategies is apparent. As expected, the speed at which the car is able to drive through a turn, particularly of the classes HAIRPIN and SLOW, is lower on DIRT than on ROAD. This is clearly visible when inspecting the double turn strategies as well. The entry position p_1 used in the first turn of the double turn strategies is overall closer to the edge of the track for DIRT than for ROAD. The DIRT tracks seem to force the car to drive slower and take less sharp turns than on ROAD tracks. The fact that there is such a large difference in the evolved strategies for two track types indicates that there is indeed a need for different behavior depending on the properties of the track surface, and that the optimizations of twice the amount of strategies was worthwhile.

The running time for the experiments were relatively comparable to those conducted when optimizing the gearing scheme. Using double the number of generations, the runtime of each evolutionary run became around six hours long. This limited the number of evolutionary runs possible to complete within reasonable time. The complete collection of strategies evolved is listed in Appendix B.

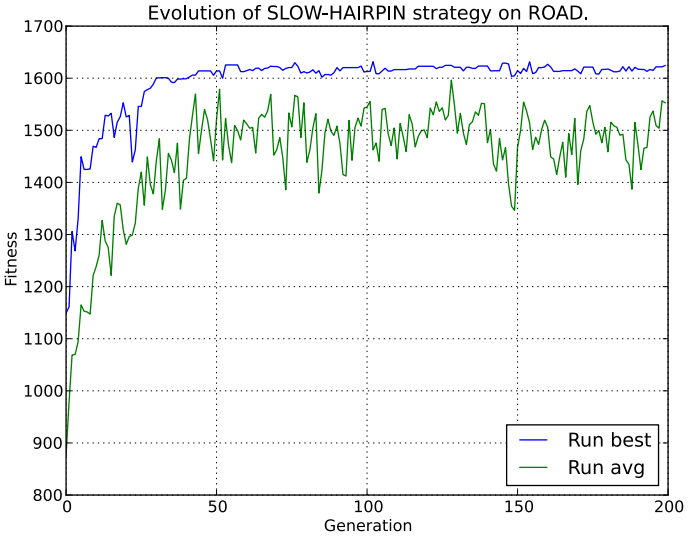


Figure 4.4: One evolutionary run of the double turn strategy for SLOW-HAIRPIN on track type ROAD.

Track	Length	Single turn strategies	Double turn strategies	Difference
Wheel1	4257.6 m	97.9 s	89.2 s	−8.9%
Forza	5784.1 m	102.0 s	101.1 s	−0.9%
Olethros	6282.8 m	132.2 s	125.9 s	−4, 8%
E-track2	3147.5 m	99.9 s	96.5 s	−3, 4%
Alpine1	6355.7 m	164.0 s	156.4 s	−4.6%
Alpine2	3773.6 m	115.9 s	112.2 s	−3.2%
Dirt4	3260.4 m	116.4 s	103.5 s	−11.1%
Dirt5	1072.9 m	44.4 s	42.5 s	−4.3%
Ruudskogen	3274.2 m	72.8 s	70.4 s	−3.3%
Aalborg	2587.5 m	108.0 s	106.4 s	−1.3%

Table 4.4: Average of best lap time over 20 runs using single turn strategies and double turn strategies. Adaptivity was enabled in both cases.

Turn type combination	SLOW-SLOW	HIGH-HAIRPIN
Double turn strategy	$\langle 0.06, 150\rangle, \langle 0.0, 107\rangle$	$\langle 0.25, 260\rangle, \langle 0.18, 66\rangle$
Single turn strategy equivelant	$\langle 0.8, 124\rangle, \langle 0.8, 124\rangle$	$\langle 0.24, 275\rangle, \langle 0.6, 65\rangle$

Table 4.5: Two examples illustrating the difference between single and double turn strategies.

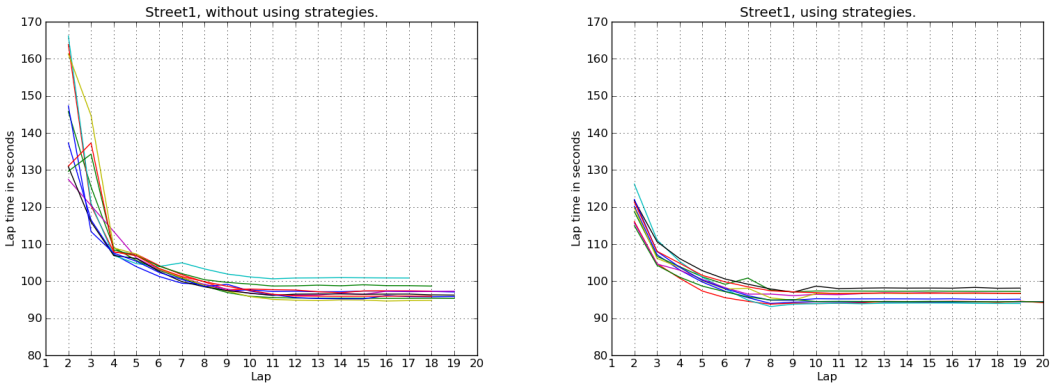
4.2.5 The effect of double turn strategies

Table 4.4 shows the results of CRABCAR's performance on 10 tracks using single turn strategies only and both single and double turn strategies. The performance is better in the noisy competition environment when using the double turn strategies in 10 out of 10 tracks. The improvement is significant, averaging at 4.6%. The performance increase is the greatest on the short and curvy tracks Wheel1 and Dirt4, whose track layouts stand out from the rest of the tracks in the experiment. They both feature many combinations of short and uneven turns where the double turn strategies are applicable. This is a testament to potential of the double turn strategies and the difference the speed and racing line optimizations can make in cases with consecutive turns.

Looking at the evolved double turn strategies, the parameters in the second turn strategies have a tendency of directing the car more towards the center of the track than their single turn strategy equivelants. This difference is particularly visible where the second turn strategy concerns one of the slower turn types. The single turn strategies for the higher speed turn types have this property already. This, combined with the fact that the higher speed turns are normally relatively long, making the speed and position at the entry of the turn less important, results in less of a difference in the double turn strategies concerned with the higher speed turns. Both scenarios are illustrated in Table 4.5. The results are consistent with the belief that it is possible to cut corners and use a straighter racing line through short, consecutive turns (as illustrated in Fig. 3.7). The speed parameters are also different in the double turn strategies compared to the equivelant single turn strategies, displaying a more cautios approach to turns when there are slower turns following.

4.3 The effect of adaptivity

The track model created through classification uses evolved strategies when determining how fast a turn can be traversed, depending on the classified turn type and track type. These strategies are bound to be somewhat general, as turns with



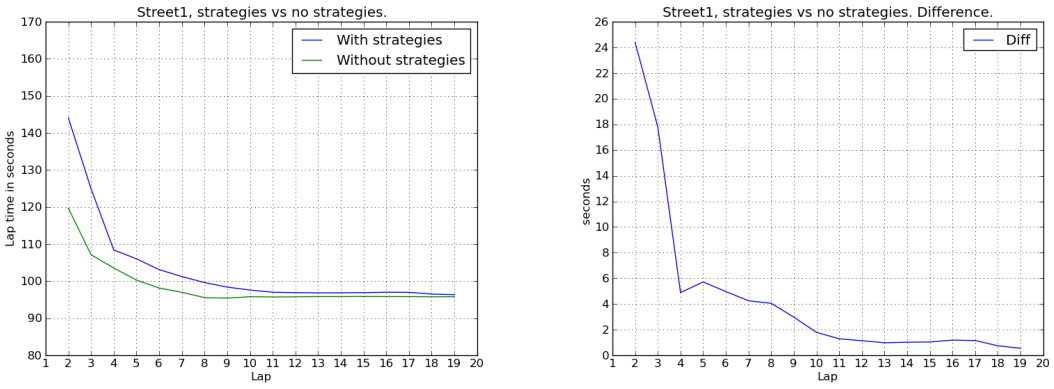
(a) (b)

Figure 4.5: Example of the influence of strategies on adaptivity, on the track Street1. Both plots start at lap 2 when the adaptivity begins to affect performance. (a) Without using strategies. Here, all turns are initially initiated to have a preferred speed of 100 km/h and preferred position of 0.0 (middle of road) when entering turn. (b) Using strategies.

slight differences are classified as the same type. Through adaptivity, we wish to improve these strategies for each specific turn, as well as learn from mistakes due to misclassification or uneven terrain.

4.3.1 Example of applying strategies

The evolved strategies applied to the track model after classification serve to give the controller some prelearned knowledge of how the different turns should be traversed. The idea is that instead of trying to optimize each preferred turn parameter from scratch, we apply a start value evolved offline to give a good indication of how each turn type should be handled. An example of how this can affect the adaption is shown in Fig. 4.5, where the controller is able to drive fast right off the bat after classification with the use of strategies, while needing more time to adapt to the track without them. The difference between the two approaches wears off over the following laps, as the controller adapts to the track. This is seen in Fig. 4.6, where the difference with and without strategies decreases after some period of time. On long tracks, where the controllers do not have the possibility to drive as many laps (corresponding to adaptation iterations), having strategies can increase the chance of learning the track better during the limited warm-up stage.



(a) (b) .

Figure 4.6: Comparison of runs done with and without the use of strategies. (a) Average lap time of 10 runs with and without the use of strategies. (b) The difference of the average lap times.

4.3.2 Testing the effect of adaptivity

To test how adaptivity can increase the performance of the controller after classification, several tracks bundled with TORCS are classified by letting CRABCAR drive one classification lap on each to create a track model with evolved strategies applied to each turn. Sensor noise is not enabled during classifiction to get consistency between the track models. Using such a track model as a basis, several runs are done on each track with adaption enabled and without adaption. Simulating the warm-up stage in the real compeition, CRABCAR gets 100.000 game ticks (approximately 30 minutes of driving time) on each track. When adaption is disabled, the driver does not adjust its behaviour in any way during the race, which means that it has to rely completely on the knowledge gained through classification and the use of static strategies. For each track, we let CRABCAR drive 20 runs with and without adaption.

4.3.3 Results and discussion

The results of the runs are shown in Table 4.6. On every track, the controller is able to make some improvement in performance by adjusting its behaviour through adaption. The performance increase from adaptivity varies alot on the different tracks tested. While only improving the lap time on Wheel1 with an average of 0.6 s, corresponding to a decrease of 0.7%, the lap time on E-track2 is reduced by

as much as 27.3 s, giving a reduction of 22.1% from the 123.6 s obtained without adaption. On average, the average best lap time is reduced by approximately 10.7% over the ten tracks, which shows that there is still optimization that can be done after applying prelearned strategies to classified turns. This does not, however, imply that using evolved strategies as a starting point serves no purpose. As a time limit is imposed on the warm-up stage in the competition, it is intuitively better only having to tweak slightly on turn strategoes through adaptivity, rather than having to develop them from scratch. This because adaption takes time, and an entire lap is required to see if a change in a strategy actually lead the car to driving a turn faster, or rather caused it to crash.

There does not seem to be any direct correlation between the length of a track and how much adaption is able to improve the strategies judging from the results. The longest track in the test, Olethros, receives an average lap time decrease of only 2.9% through adaption. The biggest improvement is observed on tracks of medium and short lengths, namely E-track2, Alpine2 and Dirt5, while the medium sized track Wheel1 shows only a slight performance increase. On a short track, adaptivity can adjust the behaviour of the car over more laps than on a long track. On a long track, on the other hand, there are usually more turns that can be tuned, which might lead to a larger number of small improvements. This could mean that there is a correleation between the impact of adaptivity and track length for both long and short tracks, but since it benefits both, it evens out and makes it hard to draw any conclusions.

The most important thing do draw from these results is that applying strategies developed through offline learning should not be used as an replacement for online adaption, but rather as a starting point for further adjustment. As adaption is able to reduce the lap time on every track tested, sometimes to a major extent, it shows that it is able to play a central role in learning to drive unknown tracks properly.

4.4 CRABCAR vs. other controllers

In this section we take a look at how CRABCAR performs compared to controllers that have proved to perform well in previous years' competitions. The controller Jorge, developed by Jorge Munõz [32], won the championship leg at WCCI-2010. COBOSTAR, developed by Butz and Lönneker [27], is the continued work of the controller that won CIG-2009 championship leg, and came in second place after Jorge at WCCI-2010. The methods used by these controllers are briefly described in Section 2.7.1. We also compare it to MrRacer, a controller developed by Quadflieg et al. for the SCRC 2010, which finished third at the leg hosted

Track	Length	Without adaption	With adaption	Difference	
				seconds	percent
Wheel1	4257.6 m	90.5 s	89.9 s	$-0.6\,s$	-0.7%
Forza	5784.1 m	114.0 s	101.9 s	$-12.1\,s$	-10.6%
Olethros	6282.8 m	131.6 s	127.8 s	$-3.8\,s$	-2.9%
E-track2	3147.5 m	123.6 s	96.3 s	$-27.3\,s$	-22.1%
Alpine1	6355.7 m	175.0 s	158.3 s	$-16.7\,s$	-9.5%
Alpine2	3773.6 m	132.4 s	112.3 s	$-20.1\,s$	-15.2%
Dirt4	3260.4 m	134.1 s	117.1 s	$-17.0\,s$	-12.7%
Dirt5	1072.9 m	46.2 s	38.7 s	$-7.5\,s$	-16.2%
Ruudskogen	3274.2 m	80.1 s	70.4 s	$-9.7\,s$	-12.1%
Aalborg	2587.5 m	108.0 s	102.6 s	$-5.4\,s$	-5.0%

Table 4.6: Average of best lap time over 20 runs with and without adaption.

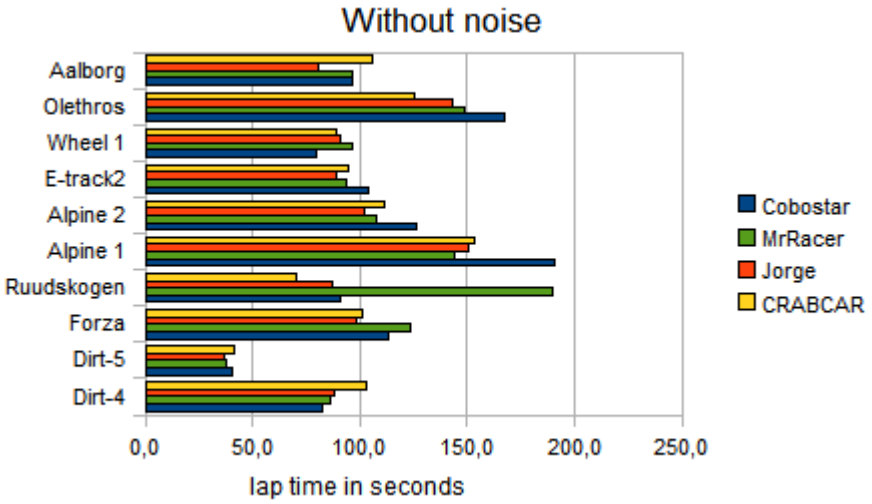
at GECCO. It uses a classification technique similar to ours, as our method is inspired by Quadflieg et al. [2]. Since their classification did not take noise into account, a workaround where they classified the entire track as a straight was used for the SCRC 2010 [33].

The setup of the experiment is that each controller get 100.000 game ticks to learn and drive a track, which is the same as the length of the warm-up stage at the competions. The controllers get 20 such warm-up runs on each track, and the best lap time from each run is averaged to get an estimation of how fast the controller is able to navigate the tracks. To get an impression of how the noisy sensors affect the controller, all runs are done twice. Once with noise disabled and once with noise enabled.

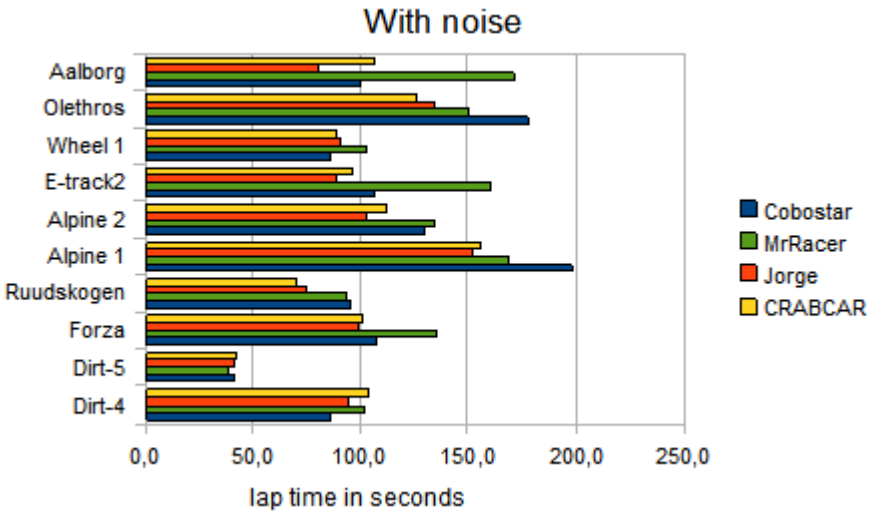
Fig. 4.7a and Fig. 4.7b shows the results from the runs where noise is disabled and from when it is enabled, respectively.

Without noise enabled, CRABCAR achieves the lowest lap time on two out of ten tracks, namely Olethros and Ruudskogen. It does, however, also produce the worst lap time on two of the tracks (Aalborg and Dirt-4). MrRacer and Cobostar are fastest on one and two tracks, respectively, while Jorge is unmatched by producing the fastest time on five of the tracks tested.

We are mostly interested in how our controller performs when sensors are influenced by noise, as this will be the case under the competitions. Also here, CRABCAR is able to drive fastest on the tracks Olethros and Ruudskogen, and Jorge again shows it has an edge by outperforming the other controllers in five of the tracks. Quadflieg et al.'s MrRacer demonstrates that it does not handle noise very well, by being the slowest controller in five of the tracks, compared to three



(a)



(b)

Figure 4.7: Comparing the performance of CRABCAR to other controllers. Average of best lap time over 20 runs are plotted for each track. Fig. (a) shows the results when sensor noise is disabled and Fig. (b) when noise is enabled.

when noise was disabled. It still manages to produce the fastest time on Dirt-5, however.

The fact that all of the controllers manage to produce the fastest driving on one or multiple tracks, while also being slowest on other tracks (with the exception of Jorge), gives an impression of how complex the simulated racing environment is. Different controllers seem to excel in different environments, showing that aiming to be best at everything may perhaps be an unachievable goal. Among the controllers tested, Jorge comes out best overall. CRABCAR is, however, in many cases not far behind, and even beats Jorge on three of the tracks tested. This suggests that even though there is a way to go compared to the best controller(s), CRABCAR's performance is indeed very competitive in the SCRC setting.

Chapter 5

Conclusion and future work

In this thesis we have given a brief introduction to the history of autonomous driving and mentioned how simulated car competitions can serve as an encouragement for developing techniques applicable for real world scenarios. The Simulated Car Racing Championship (SCRC) is a competion held annually, and our system is the implementation of a car controller which is to eventually participate in the championship hosted at the Genetic and Evolutionary Computation Conference the summer of 2011.

We conclude that it is indeed possible to build and use a non-perfect track model acquired in a noisy environment. By extending a technique described by Quadflieg et al. [2] to filter noise it was possible to construct an abstract track model which proved good enough to serve as a basis for further applications of learning techniques. When testing to what degree the noise affected the total performance of CRABCAR, an average increase of only 1% to the lap time was observed. This is a very encouraging result, when it is taken into consideration that the original technique broke down completely when presented with noisy sensory data. A good part of the performance is thanks to the online behavior adaption system that optimized the behaviour of the car to fit specific racing tracks. The noise filtering introduced, together with the adaption system, made it possible to take advantage of a non-perfect track model.

We also sought out to investigate how offline learning could aid performance when there is very limited time for online learning during the competition. In our work we use evolutionary computation offline to generate strategies related to general handling of different turn types. After classifying a track and its turns online, such strategies are applied to give the controller some general knowledge of how the different types of turns should be handled. Using the strategies as a starting point, the controller utilizes online adaption in an attempt to optimize strategies

on an individual turn basis.

Using knowledge acquired through offline learning can indeed produce a better result when the track model is non-perfect and the time available for online learning is limited. This is especially true for longer tracks. The extended strategies proposed in this thesis show promising results. Optimizing the racing line used in turns together with speed proved successful, especially when taking into account the special cases of consecutive turns by using double turn strategies.

Offline learning by means of evolutionary computation has shown itself as quite a time-consuming process. Working with a simulator, which experimental runs are practically difficult to parallelize, limits the potential of employing such techniques in the search for good behavior.

Comparing CRABCAR's performance in the SCRC context with other notable entries from the last two years showed that the system as a whole performed as good as, and in a some cases better than, the other controllers. This is promising with regards to the championship where CRABCAR will compete. Even though pure performance was not the goal of developing the system, it is inspiring to see that it is performing on the same level as the better controllers from the previous championships.

A common theme throughout the work done in this thesis has been the fragility that has been evident in highly optimized car racing behavior. Driving on the very limits of what the car is capable of in all situations makes for a very fine line between the ultimate success and total failure. A small difference in the dynamics between the car and the track, the position of the car or the speed of the car will likely lead to the car losing control and crashing.

It is clear that having some form of online reinforcement learning plays an important role when working with a discretized track model. The problem lies with the evaluation used to guide the search for better performance of the car online. A time based approach can focus on the only thing that matters in a competition setting, which is the time it takes the car to circuit the track. The total lap time is not precise enough though, as many changes may occur during a lap around the track. Determining which changes affected the lap time in a positive way should be the goal of the evaluation. It needs to work on a local scale, down to evaluating performance in single turns. It also needs to take into account combinations of turns, as a change in the strategy for one turn can negatively affect the performance in subsequent turns. By developing a better heuristic than trial and error (i.e. not requiring a crash before deciding to reduce driving speed), the search could be sped up. With the limited time available, the speed of the search is paramount.

The process of building the track model has significant room for improvement.

The noise filtering in particular can be handled more elegantly than what is done in this thesis. Kalman filters could for example be employed to model the noise and more accurately calculate sensor signals closer to the truth. Based on more accurate sensor signals, a more granular turn type system could refine the car's behavior and ease the need for online adaptivity to optimize strategies to specific racing tracks. A more accurate track model could also be used to model an optimal racing line in its entirety, using the approach proposed by Cardamone et al. [35] which was based on perfect track layout knowledge. Knowing both the optimal racing line and the layout of the track ahead of the car could make it possible to incorporate advanced opponent handling, minimizing the departure from the optimal racing when passing other cars and exploiting the shape of turns efficiently.

The encouraging results from using the more advanced strategies when navigating turns inspire further work in the area. Having turn strategies that take turn length and track width into account when calculating speed and position is a possibility. Given that both turn length and track width are both in theory unlimited, they could either be intergrated into a track model building process where they are defined by discrete classes, or they could serve as input for matehmatical functions that compute the speed and position in a more granular way. Adding length and width classes to the already existing curvature and surface classes would increase the total number of combinations dramatically, potentially making the process of optimizing parameters for every one of them an unfeasible task. Either way, it would be interesting to incorporate the factors of speed and racing line even better into the strategies.

It has become clear to us throughout working with a simulated car racing environment that it truly poses some complex problems, which again are riddled with smaller problems. The results from comparing CRABCAR to other controller systems reveals that it is possible to outperform other controllers on some tracks, while underperforming on others. It might be theoretically possible create a controller that is consistently best in every environment, but a more realistic goal is probably to aim for a good all-rounder which hopefully is able to excel when averaged across a number of tracks.

The methods and results presented in this report has mainly focused on behaviour and performance when driving alone on a track. The opponent module in its current state is trivial, and is only concerned with trying to avoid other cars in close vicinity. Expanding the functionality of the module is a natural continuation of the work done in this thesis. Using the track model, it is possible to exploit the layout of the track ahead in situations where opponent cars are involved. Overtaking on the inside of turns, blocking opponents that are approaching from behind in sharp turns and safely navigating the densely populated field of cars existing right after a race has started are a few behaviors that could improve CRABCAR.

Bibliography

[1] Wall Street Journal. http://online.wsj.com/article/SB119948828539568677.html?mod=hpp_us_whats_news. Date accessed: 13.06.2011.

[2] Jan Quadflieg, Mike Preuss, Oliver Kramer, and Günter Rudolph. Learning the Track and Planning Ahead in a Car Racing Controller, 2010.

[3] E.D. Dickmanns. *Dynamic vision for perception and control of motion*. Springer Verlag, 2007.

[4] Popular Science. Bonnier Coorporation, oct. 1985.

[5] M. Williams. PROMETHEUS-The European research programme for optimising the Road Transport System in Europe. In *Driver Information, IEE Colloquium on*, page 1. IET, 2002.

[6] M. Maurer, R. Behringer, S. Furst, F. Thomanek, and ED Dickmanns. A compact vision system for road vehicle guidance. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 3, pages 313–317. IEEE, 2002.

[7] No hands across america. http://www.cs.cmu.edu/afs/cs/usr/tjochem/www/nhaa/nhaa_home_page.html. Date accessed: 13.06.2011.

[8] VisLab ARGO. http://www.argo.ce.unipr.it/ARGO/english/index.html. Date accessed: 13.06.2011.

[9] DARPA 04. http://archive.darpa.mil/grandchallenge04/. Date accessed: 13.06.2011.

[10] DARPA 05. http://archive.darpa.mil/grandchallenge05/. Date accessed: 13.06.2011.

[11] DARPA 07. http://archive.darpa.mil/grandchallenge/index.asp. Date accessed: 13.06.2011.

[12] The Google Blog. http://googleblog.blogspot.com/2010/10/what-were-driving-at.html. Date accessed: 13.06.2011.

[13] VIAC. http://vislab.it/Projects/view/32/VisLab's_adventure_on_the_Silk_road. Date accessed: 13.06.2011.

[14] J Togelius and S Lucas. The 2007 IEEE CEC simulated car racing competition. *Genetic Programming ...*, 2008.

[15] D. Loiacono, J. Togelius, P.L. Lanzi, L. Kinnaird-Heether, S.M. Lucas, M. Simmerson, D. Perez, R.G. Reynolds, and Y. Saez. The wcci 2008 simulated car racing competition. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 119–126. IEEE, 2009.

[16] D Loiacono and PL Lanzi. The 2009 Simulated Car Racing Championship. *... Intelligence and AI ...*, 2010.

[17] TORCS web page. http://torcs.sourceforge.net/. Date accessed: 13.06.2011.

[18] TORCS project history. http://torcs.cvs.sourceforge.net/torcs/torcs/torcs/doc/history/history.txt. Date: Date accessed: 13.06.2011.

[19] GNU General Public Licence. http://www.gnu.org/licenses/gpl.html. Date accessed: 13.06.2011.

[20] Free Art Licence 1.3. http://artlibre.org/licence/lal/en. Date accessed: 13.06.2011.

[21] The TORCS Racing Board. http://www.berniw.org/trb/. Data accessed: 13.06.2011.

[22] LJ Fogel. Artificial intelligence through simulated evolution. 1966.

[23] D. Floreano and C. Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*. 2008.

[24] RC Arkin. Behavior-based robotics. 1998.

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

[26] L. Cardamone, D. Loiacono, and P.L. Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186. ACM, 2009.

[27] M.V. Butz and T.D. Lönneker. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. *... and Games, 2009. CIG 2009. IEEE ...*, 2009.

[28] D Perez and G Recio. Evolving a fuzzy controller for a car racing competition. *... Intelligence and Games, ...*, 2009.

[29] J. Togelius, R. De Nardi, and S.M. Lucas. Making racing fun through player modeling and track evolution. *Optimizing Player Satisfaction in Computer and Physical Games*, page 61, 2006.

[30] N. Van Hoorn, J. Togelius, D. Wierstra, and J. Schmidhuber. Robust player imitation using multiobjective evolution. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 652–659. IEEE, 2009.

[31] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Learning Drivers for TORCS through Imitation Using Supervised Methods, 2009.

[32] J Munoz. A human-like TORCS controller for the Simulated Car Racing Championship. 2010.

[33] SCRC @ CIG-2010. `http://www.slideshare.net/dloiacono/2010-simulated-car-racing-championship-cig2010?from=ss_embed`. Date accessed: 13.06.2011.

[34] F Braghin and F Cheli. Race driver model. *Computers & Structures*, 2008.

[35] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the Optimal Racing Line Using Genetic Algorithms, 2010.

[36] Watchmaker framework. `http://watchmaker.uncommons.org/`. Date accessed: 13.06.2011.

Appendix A

Competition rules and setup

Following is a direct copy of the rules and regulations of what will be the 2011 Simulated Car Racing Championship [1]. In addition, tables A.1, A.2,A.3, describe the interface that the controllers have to work with, consisting of the effectors available for manipulation and the sensory data received from the simulation.[2].

A.1 Rules and Regulations

The championship consists of several races on different tracks divided into legs. Teams will be allowed to submit a different driver to each leg.

Each Grand Prix consists of three stages:

- the warm-up
- the qualifying
- the race

During warm-up, each driver races alone. Drivers can collect useful information about the tracks and can tune their behaviors for the next stages. Accordingly, the performance of drivers in this stage is not taken into account for their scores.

During the qualifying stage each driver races alone on each track of the leg. The eight controllers that bridge the longest distances qualify for the actual Grand Prix races.

[1] SCRC rules and regulations: http://cig.ws.dei.polimi.it/?page_id=175

[2] Competition software manual:
https://sourceforge.net/projects/cig/files/Championship%202010%20Manual/1.0/manual.pdf/download

During the final races, these best eight drivers race together. The races consist of eight runs on each of the three tracks. At the end of each race, the drivers are scored using the F1 system: 10 points to the first controller that completes the race, 8 points to the second one, 6 to the third one, 5 to the fourth, 4 to the fifth one, 3 to the sixth, 2 to the seventh, and 1 to the eighth. The driver performing the fastest lap in the race will get two additional points. The driver completing the race with the smallest amount of damage will also get two extra points. The starting grid of the first race will be based on the performance obtained in the qualifying stage. Each subsequent race, the starting grid will be shifted by one so that each driver starts from every position of the starting grid exactly once.

A.2 Effectors and sensors

Name	Range	Description
accel	$[0, 1]$	Virtual gas pedal (0 means no gas, 1 means full gas).
brake	$[0, 1]$	Virtual brake pedal (0 means no brake, 1 means full brake).
clutch	$[0, 1]$	Virtual clutch pedal (0 means no clutch, 1 means full clutch).
gear	$\{-1, 0, 1, ..., 7\}$	Gear value.
steering	$[-1, 1]$	Steering value: -1 and +1 means full right and full left turn, respectively, corresponding to and angle of $\pi/4$ rad.
focus	$[-90, 90]$	Focus direction (see *focus* sensors in Table A.2) in degrees.
meta	$\{0, 1\}$	This is the meta command. 0 does nothing, 1 asks the competition to restart the race.

Table A.1: Effectors available to the SCRC participant controllers.

Name	Range (unit)	Description
angle	$[-\pi, +\pi]$	Angle between the car direction and the direction of the track axis.
curLapTime	$[0, +\infty)$ (s)	Time elapsed during current lap.
damage	$[0, +\infty)$ (point)	Current damage of the car (the higher the value is, the higher the damage).
distFromStart	$[0, +\infty)$ (m)	Distance of the car from the start line along the track line.
distRaced	$[0, +\infty)$ (m)	Distance covered by the car from the beginning of the race.
focus	$[0, 200]$ (m)	Vector of 5 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 meters. When *noisy* option is enabled sensors are affected by i.i.d. normal noise with a standard deviation equal to 1% of the sensors' range. The sensors sample, with a resolution of 1°, a 5° space along a specific direction provided by the client (the direction is defined with the *focus* command and must be in the range $[-\pi/2, +\pi/2]$ w.r.t. the car axis). Focus sensors are not always available: They can be used only once per second of simulated time. When the car is outside of the track (i.e., *pos* is less than -1 or greater than 1), the focus direction is outside the allowed range ($[-\pi/2, +\pi/2]$) or the sensor already has been used once in the last second, the returned values are not reliable (typically -1 is returned).
fuel	$[0, +\infty)$ (l)	Current fuel level.
gear	$\{-1, 0, 1, ..., 7\}$	Current gear: -1 is reverse, 0 is netural, and $1-7$ is gear $1-7$.
lastLapTime	$[0, +\infty)$ (s)	Time spent completing the previous lap.
opponents	$[0, 200]$ (m)	Vector of 36 opponent sensors: Each sensor covers a span of $\pi/18$ (10°) within a range of 200 meters and returns the distance of the closest opponent in the covered area. When *noisy* option is enabled, sensors are affected by i.i.d. normal noise with a standard deviation equal to 2% of the sensor's range. The 36 sensors cover all the space around the car, spanning clockwise from $+\pi$ up to $-\pi$ with respect to the car axis.

Table A.2: Sensory information available to the controllers (part I).

Name	Range (unit)	Description
racePos	$\{1, 2, ..., N\}$	Position in the race with respect to other cars.
rpm	$[2000, 10000]$ (rpm)	Number of revolutions per minute of the car engine.
speedX	$(-\infty, +\infty)$ (km/h)	Speed of the car along the longitudinal axis of the car.
speedY	$(-\infty, +\infty)$ (km/h)	Speed of the car along the transverse axis of the car.
speedZ	$(-\infty, +\infty)$ (km/h)	Speed of the car along the Z axis of the car.
track	$[0, 200]$ (m)	Vector of 19 range finder sensors: Each sensor returns the distance between the track edge and the car within a range of 200 meters. When *noisy* option is enabled, sensors are affected by i.i.d. normal noise with a standard deviation equal to 10% of the sensors' range. By default, the sensors sample the space in front of the car every 10°, spanning clockwise from $+\pi/2$ up to $-\pi/2$ with respect to the car axis. The configuration of these sensors can be set manually by the controller before each race. When the car is outside of the track (i.e., *pos* is less than-1 or greater than 1), the returned values are not reliable.
trackPos	$(-\infty, +\infty)$	Distance between the car and the track axis. The value is normalized w.r.t. the track width: It is 0 when the car is on the track axis, -1 when the var is on the right edge of the track and $+1$ when it is on the left edge of the track. Values greater than 1 or smaller than-1 mean that the var is outside of the track.
wheelSpinVel	$[0, +\infty]$ (rad/s)	Vector of 4 sensors representing the rotation speed of the wheels.
z	$(-\infty, +\infty)$ (m)	Distance of the car mass center from the surface of the track along the Z axis.

Table A.3: Sensory information available to the controllers (part II).

Appendix B

Strategies evolved

B.1 Single turn strategies

This table contains the strategies evolved for use in single turns.

Turn type	Surface type	Strategy $\langle p, v \rangle$
HAIRPIN	ROAD	$\langle 0.6, 65 \rangle$
SLOW	ROAD	$\langle 0.8, 124 \rangle$
MEDIUM	ROAD	$\langle 0.4, 233 \rangle$
HIGH	ROAD	$\langle 0.24, 275 \rangle$
HAIRPIN	DIRT	$\langle 0.24, 52 \rangle$
SLOW	DIRT	$\langle 0.6, 83 \rangle$
MEDIUM	DIRT	$\langle 0.61, 162 \rangle$
HIGH	DIRT	$\langle 0.84, 273 \rangle$

Table B.1: The final single turn strategies.

B.2 Double turn strategies

This table contains the strategies evolved for use in double turns.

Turn types	Surface	First strategy $\langle p_1, v_1 \rangle$	Second strategy $\langle p_2, v_2 \rangle$
HAIRPIN-HAIRPIN	ROAD	$\langle 0.51, 90 \rangle$	$\langle 0.34, 62 \rangle$
SLOW-HAIRPIN	ROAD	$\langle 0.17, 148 \rangle$	$\langle 0.26, 68 \rangle$
SLOW-SLOW	ROAD	$\langle 0.06, 150 \rangle$	$\langle 0.0, 107 \rangle$
MEDIUM-HAIRPIN	ROAD	$\langle 0.3, 170 \rangle$	$\langle 0.23, 68 \rangle$
MEDIUM-SLOW	ROAD	$\langle 0.28, 194 \rangle$	$\langle 0.03, 106 \rangle$
MEDIUM-MEDIUM	ROAD	$\langle 0.42, 238 \rangle$	$\langle 0.28, 202 \rangle$
HIGH-HAIRPIN	ROAD	$\langle 0.25, 260 \rangle$	$\langle 0.18, 66 \rangle$
HIGH-SLOW	ROAD	$\langle 0.11, 264 \rangle$	$\langle 0.27, 117 \rangle$
HIGH-MEDIUM	ROAD	$\langle 0.06, 276 \rangle$	$\langle 0.47, 208 \rangle$
HIGH-HIGH	ROAD	$\langle 0.02, 285 \rangle$	$\langle 0.04, 285 \rangle$
HAIRPIN-HAIRPIN	DIRT	$\langle 0.6, 76 \rangle$	$\langle 0.31, 47 \rangle$
SLOW-HAIRPIN	DIRT	$\langle 0.34, 89 \rangle$	$\langle 0.22, 46 \rangle$
SLOW-SLOW	DIRT	$\langle 0.12, 88 \rangle$	$\langle 0.09, 71 \rangle$
MEDIUM-HAIRPIN	DIRT	$\langle 0.33, 145 \rangle$	$\langle 0.42, 50 \rangle$
MEDIUM-SLOW	DIRT	$\langle 0.34, 158 \rangle$	$\langle 0.1, 106 \rangle$
MEDIUM-MEDIUM	DIRT	$\langle 0.12, 167 \rangle$	$\langle 0.23, 130 \rangle$
HIGH-HAIRPIN	DIRT	$\langle 0.22, 248 \rangle$	$\langle 0.34, 53 \rangle$
HIGH-SLOW	DIRT	$\langle 0.28, 252 \rangle$	$\langle 0.18, 82 \rangle$
HIGH-MEDIUM	DIRT	$\langle 0.18, 253 \rangle$	$\langle 0.23, 150 \rangle$
HIGH-HIGH	DIRT	$\langle 0.1, 268 \rangle$	$\langle 0.07, 272 \rangle$

Table B.2: The final double turn strategies.