
***Facultad***
***de***
***Ciencias***

# Caracterización e imitación de patrones de conducción en circuitos simulados
## (Characterisation and imitation of driving patterns in simulated circuits)

**Trabajo de Fin de Grado**
**para acceder al**

**GRADO EN INGENIERÍA INFORMÁTICA**

**Autor: Guzmán Rodríguez**

**Director: Camilo Palazuelos**

**Co-Director: Jose Luis Montaña**

**Septiembre - 2022**

# Contents

**1 Abstract** **4**

**2 Introduction** **6**
2.1 Motivation . . . 6
2.2 Objectives . . . 7

**3 Tools** **8**
3.1 Pycharm . . . 8
3.2 Sublime Text 3 . . . 8
3.3 Adobe Photoshop . . . 8

**4 Languages and modules** **10**
4.1 Python . . . 10
4.2 JSON . . . 10
4.3 Python pygame . . . 10
4.4 Python pyAgrum . . . 11

**5 Driving Simulator** **12**
5.1 Requirements . . . 12
5.1.1 Functional . . . 12
5.1.2 Non-functional . . . 13
5.2 Use cases . . . 13
5.3 Classes . . . 14
5.3.1 AbstractCar . . . 16
5.3.2 Track . . . 17
5.4 Game loop . . . 18
5.5 User Interface . . . 18
5.6 Track design . . . 20
5.7 Logging . . . 21
5.8 Keeping records . . . 22
5.9 Tests . . . 22
5.9.1 Unit testing . . . 23
5.9.2 Integration testing . . . 23
5.9.3 System testing . . . 23
5.9.4 Acceptation testing . . . 24

**6 Classification** **25**
6.1 Representation . . . 25
6.2 Learning . . . 30
6.3 Inference . . . 31

**7 Experimentation** **33**
7.1 Success rate and compute cost . . . 37
7.2 ROC curve and AUC . . . 38

**8 Cloning** **43**
8.1 Representation . . . . . 43
8.2 Learning . . . . . 43
8.3 Generating samples . . . . . 43
8.4 Results . . . . . 44

**9 Conclusions** **46**

**10 Bibliography** **47**

# 1 Abstract


### English version

Imitation learning is based on learning from the actions of an observed third party. One of the most common tasks being developed today based on observational learning is autonomous driving vehicles. These vehicles attempt to mimic a succession of actions by learning from one or more drivers. Although progress in this field is significant, most algorithms do not distinguish between the skill level of the drivers they imitate. This limits both the learning process and the results obtained [1].

The main objective of this dissertation is to propose a classification model that can distinguish between different drivers solely on the basis of their driving style. And, subsequently, to be able to replicate these learned patterns. All this developed within a driving simulator that allows to save the driving data of the users who test it.

1. Design and implement a driving simulator with different circuits of varying difficulty that allows to save all the data concerning the driving of the users who play with this simulator.
2. Develop a classification model through imitation learning that is able to identify different users who have previously driven based on simulator records.
3. Modify the above model so that it is able to mimic the behaviour of users, at least on circuits where they have previously driven.

For the development of the model, we will follow the methodology used in [5], based on dynamic Bayesian networks with latent variables.

## Spanish version

El aprendizaje por imitación se basa en aprender a partir de las acciones de un tercero que es observado. Una de las tareas más comunes que hoy en día se están desarrollando en base a el aprendizaje por observación es la conducción autónoma de vehículos. Estos vehículos intentan imitar una sucesión de acciones aprendiendo de uno o varios conductores. Aunque el avance en este campo es significativo la mayor parte de los algoritmos no distingue entre el nivel de destreza de los conductores que imita. Esto limita tanto el proceso de aprendizaje como los resultados obtenidos [1].

El objetivo principal de este TFG es proponer un modelo de clasificación que permita distinguir entre diferentes conductores únicamente a través de su estilo de conducción. Y, posteriormente, que sea capaz de replicar dichos patrones aprendidos. Todo esto desarrollado dentro de un simulador de conducción que permita guardar los datos de conducción de los usuarios que lo prueben.

1. Diseñar e implementar un simulador de conducción con distintos circuitos de dificultad variable que permita guardar todos los datos referentes a la conducción de los usuarios que jueguen con dicho simulador.
2. Desarrollar un modelo de clasificación a través del aprendizaje por imitación que a partir de los registros del simulador sea capaz de identificar a los distintos usuarios que hayan conducido previamente.
3. Modificar el modelo anterior de manera que sea capaz de imitar el comportamiento de los usuarios, al menos en circuitos en los que hayan conducido previamente.

Para el desarrollo del modelo, seguiremos la metodología utilizada en [5], basada en redes Bayesianas dinámicas con variables latentes.

# 2 Introduction

As human beings, we are accustomed from childhood to learn by watching others. Nowadays, many artificial intelligence systems [2] are trained through the use of Learning from Observation (LfO) [7], i.e., their way of acting is based on replicating the behaviour of the intelligent agents they have learned from. Generally, these algorithms replicate the users without distinguishing between them, which means that they learn from both, good and bad agents indiscriminately. This is exactly the same as it happens to us when we are children, it is important for us to differentiate between good and bad referrals.

This shortcoming, which currently Learning from Observation (LfO) systems suffer from, is what motivates our main hypothesis:

- Intelligent agents can be recognised from the traces by using very simple models.

Through the creation of a driving simulator we will be able to get the traces from different agents in order to train a simple classification system by using Dynamic Bayesian Networks (DBNs). This system will recognize the agent owner of a certain trace with a simple and computationally inexpensive execution.

The steering wheel movements are the only user action that will form the traces. By using this, we make our model [4] compatible with almost any other driving system, car, or whatever. In this way, when other systems want to use this classification system they will not have to care about speed or acceleration, just saving the steering wheel movements the users will be classified by.

Apart from the classification, the other goal of this work is to generate samples of the drivers to see if it is possible to replicate the driving style of the pilots, may even be on circuits where they have not yet driven. The main idea behind this is to create a sector-based circuit design system. In this way, we can easily classify the type of the road that the user is driving through in order to help when cloning in a new track where the user has not driven yet.

## 2.1 Motivation

Introducing a simple classification system that could be applied to all LfO systems in order to identify each agent through the traces, so that only the chosen ones will be used to create the computer intelligent agent when replicating the task.

This will be the way to identify and exclude the bad users or the users with a bad performance when creating computer intelligent agents. Likewise, this could also be applied as a raw identification system if someone has the need to find or go after some user.

The idea of making a driving simulator is due to the fact that nowadays self-driving cars are a reality that will gradually be introduced into the vehicle market. With that in mind, applying this classification to the cars and recognizing the driver just by its moves may be very useful.

## 2.2 Objectives

This dissertation has three main objectives. The first one is to design and build a driving simulator that saves the traces from the users drivings. The idea is to save as much information as possible to see how much of it is meaningful when developing the models.

The second objective, and the most important, is to create a classification model that identifies a user from a set of those by only using the driving traces taken from the simulator.

Finally, the third objective is to build an imitation model based on cloning the driving style of one of the users. For this model we will take into account the environment where the user drives for trying to replicate that driving style in tracks where the user has not driven yet.

# 3 Tools

First of all, we give a brief description of all the software applications used in this project. The tools used are a python IDE (Pycharm), a text editor for code testing (Sublime Text 3), and a photo editor to make all the graphic resources of the project (Adobe Photoshop).

## 3.1 Pycharm

An IDE (Integrated Development Environment) is a heavy program which allows you to store complete projects providing the user with specific tools on the programming language used. In this case, as the project is developed in python, Pycharm stands as one of the most important python IDEs, developed by JetBrains in 2017. JetBrains was created in 2000 and since then they have created a large number of software tools for coding, as the one used in this project, providing them with all the tools and resources.

## 3.2 Sublime Text 3

On the other hand, a text editor is also used for coding, but there are many differences between a text editor and an IDE. A text editor is a lightweight software whose principal purpose is to edit single file codes. Sublime Text 3 is the one chosen for this project, released in January 2008. It is a cross-platform code editor whose main features are the ability to enhance its functionality using Package Control and creating custom settings.

The use of a text editor over the IDE is mainly for code testing. When you want to add a new functionality to the project it is firstly developed in a separate file and after it has been tested you can integrate this code into your project keeping it clean in case of bugs appear during development.

## 3.3 Adobe Photoshop

Talking about photo editing, Adobe Photoshop is the greatest software on the market. Owned by Adobe, it was released on February 1990, developed by brothers Thomas and John Knoll as a single-layer environment where a whole range of effects, texts, markings and treatments could be applied. Nowadays, it has become a complex software used not only by designers, but also by photographers.

As this project is partly devoted to the development of a simulator, software is needed for the development of the graphical resources as this is meant to be an entirely own project, without any elements obtained from the internet. All of these will be designed using Adobe Photoshop software.

# 4 Languages and modules

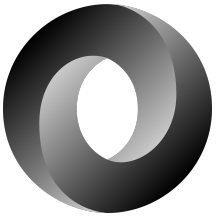
This project has been coded mostly in python, but another language and several python modules have been used for solving mathematical problems (python math), interacting with system files (python os) or controlling timings (python time). Here we take a deeper look into the main modules and languages used in this project. Apart from python, the JSON language for storing information, the main module used to develop the simulator (python pygame) and the one used to build the Bayesian network (python pyAgrum).

## 4.1 Python

Python is an open-source object-oriented high-level programming language. Released in 1992, it is an ideal language to start given its ease of writing syntax. With a huge community behind, in order to support newbies coding, it is the most popular programming language in 2021 according to [6].
The main uses of this programming language are AI and machine learning, data analytics, data visualization, programming applications, web development, game development and language development among others.

It has the capacity to create simple games easily, which will help in the development of the simulator. So, to sum up, python has all the features that this project requires.

## 4.2 JSON



JSON (Jasa Script Object Notation) is a standard text-based format for representing structured data in javascript object syntax and it is mainly used to send data in web applications. Although the purpose of this language is to store data in JavaScript applications, it is also usually used in python applications as it has its own python module to its proper use called "json".

The way to store information into json files using python is through python dictionaries. Using the previously mentioned module, python is able to store a dictionary directly into a json file changing the notation automatically, which makes json files a way to store information even for python applications.

## 4.3 Python pygame

Pygame is a set of python modules created to be the main module in python to build video games. It is inspired by SDL (Simple DirectMedia Layer) which allows you to create fully

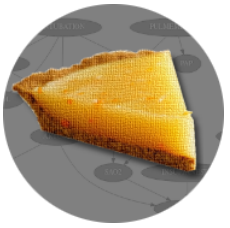
featured games and multimedia programs in the python language.

It was created in October 2000 by Lenard Lindstrom, René Dudfield, Pete Shinners, Nicholas Dudfield, Thomas Kluyver, and others. Nowadays it has been improved by the large community that this module has, being improved yearly and owning their personal game jam, discord servers for users helping and much more.

The main advantage of using pygame to build games are the features that the module has, which sets a perfect environment to develop the game loop and input interruptions easily. This makes this module perfect not only to create video games, but also to create any python applications that require a user interface and graphic designs.

### 4.4 Python pyAgrum

PyAgrum is a library firstly coded in the language C++ (agrum) that has been adapted to be a Python module as well. It is mainly based on creating Bayesian Networks and ther types of probabilistic graphical models. One thing that makes this module better than the others is that it provides graphical resources to visualise different aspects of the model you are developing.



# 5 Driving Simulator

The first objective of the dissertation is to design and implement a driving simulator with circuits of different difficulty levels to record the driving experience of different users. This simulator was created from scratch, trying to develop a good driving experience for the users making it entertaining, a desktop game which builds an artificial intelligence in the back end.

## 5.1 Requirements

Firstly, we introduce the functional and non-functional requirements of this applications. Although this thesis does not specify the features of the simulator, the functional requirements are taken by the information needs in order to build the Bayesian network.

### 5.1.1 Functional

These are the requirements that the system and the final user need in the final product.

| Reference | Requirement |
|---|---|
| RF01 | The name of the user must be registered before starting to drive |
| RF02 | There must be different tracks of varying difficulty |
| RF03 | All the circuits must have a single path to drive through them |
| RF04 | All the tracks must be designed by a set of defined sectors |
| RF05 | While driving, all the information of the drive must be logged in different files |
| RF06 | There must be several cars available |
| RF07 | The velocity of the car must be given by a gear system that the user is able to control |
| RF08 | Before starting to drive the user must see a previsualization of the car and the track selected |
| RF09 | While driving the user must have the information about how the driving is going (velocity, gear, revolutions and time lap) |
| RF10 | There must be an option to drive in order to be classified between the drivers that have already driven |
| RF11 | There must be an option to select a driver and see a simulation of how could be one of his drivings |
| RF12 | The interface must have a screen to explain the user how to control the car, explaining the function of each key |
| RF13 | All the logs must be saved in a directory system, where every user has their own directory that contains a folder for each track. That folder will contain the proper logs |
| RF14 | There must be a record of the lap times for each user and each track |

### 5.1.2 Non-functional

These other type of requirements are constraints of the system itself, regarding to portability, security, scalability and performance among others. They are not features that the system has to develop.

- Re-scalability adapting to full size window

The idea of the game is that it will perform in a full size window, so the performance should not change depending on the size of the user screen implementing a scalable game.

- Compatibility

This simulator must be able to be run in any windows system by exporting the project to a Windows installer script.

- Performance ensurance

As this is a driving game that depends on real time, lap times, the performance must be equally good for all the users so that their performance will not be affected. Using python pygame ensures this performance as it is capable of using multi-core in its execution.

- Usability

Although the goal of this project depends on the expertise of the users, they must control the basic features of the game so that their experience and performance will be good enough.

The term of usability is defined by ISO 9241-11: "Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. Effectiveness is fundamental as it is about achieving the intended goal(s). Efficiency is about the resources (such as time or effort) needed by users to achieve their goals so it can be important. In addition, it is important that users are satisfied with their experience, particularly where users have discretion over whether to use a product and can readily choose some alternative means of achieving their goals".

The aim is that the user will satisfy all the five principles of the usability: easy to learn, efficiency, remembering, error rate and satisfaction.

## 5.2 Use cases

As the functionality of this application is a simple single-user game, in the use cases we will not find the actions of an administrator. Instead, the only agent that will take actions is the user that plays the game. In this way, we can see in figure 1 that we only have the casual actions while playing the game performed by the user.

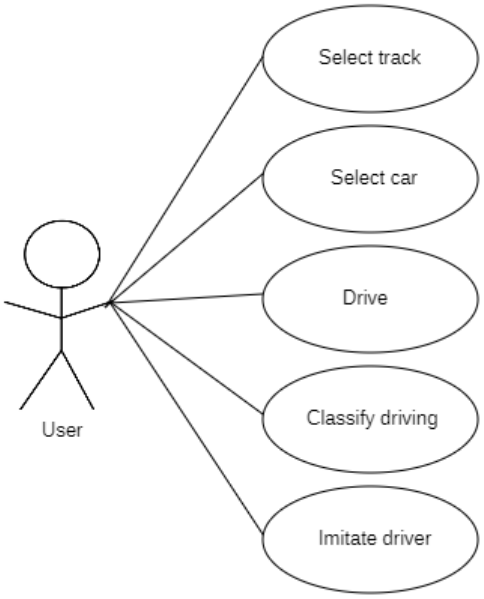



Figure 1: Use cases

### 5.3 Classes

This game is structured in three main classes. The main class which contains the game loop, the Track, and the Car. There are also some classes that support the development of the pygame interface, such as Button, Text and InputBox.

For the design of the Car class we have first created an abstract class called AbstractCar. This class contains all the different features needed in order to move the car and keeping all the different variables. From this class we have two different types of car. The PlayerCar, which will contain different functions to get the logs and some other features that are applied to the car that the user plays with. And the other one is the AICar, which will be used to simulate the driving of the cloned driver, and have some functions that make easier reading the traces created.

The figure 2 shows how the classes are structured in the class diagram.

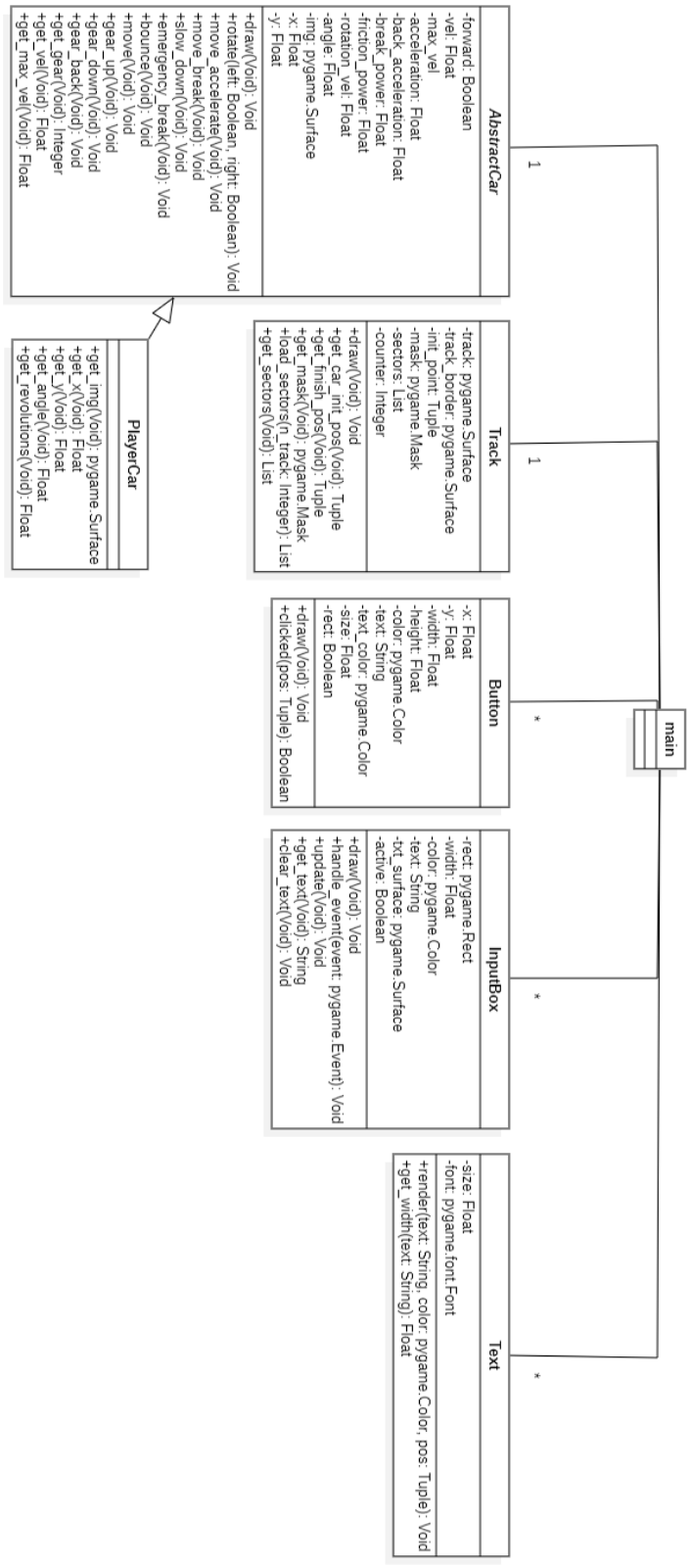



Figure 2: Class diagram

### 5.3.1 AbstractCar

The first thing that you need to implement in a driving simulator is the car movement, in this case all the car basic features and functions are defined in an abstract class called AbstractCar.

The arguments of the AbstractCar are the car image, the initial position of the car, and the angle of rotation. All this information is saved in variables and the driving parameters such as velocity, acceleration and others are set in other variables.

Car movement will be defined in 5 different functions depending on the type of movement: accelerate, break, slow down, emergency break and bounce.

#### Accelerate

Represents the action when the user presses the accelerator, this movement depends on which gear the car is in. The maximum speed the car will reach through acceleration is directly proportional to the gear, and the acceleration of the car is inversely proportional to the gear. Unless the car is in reverse gear, in which case the acceleration and maximum speed moving backwards are static as there is only one reverse gear.

#### Break

If the user breaks, the velocity will just be reduced until reaching 0, unless the car is moving backwards, in which case the velocity will increase up to 0, and the car will stop. In this case it does not depend on the gear.

#### Slow down

In case of the car is moving and the driver does not take any action the car must slow down lightly until it stops. The car will be affected by a friction power made by the ground and wind and the velocity will be reduced or incremented depending if it is below 0.

#### Emergency break

When the car is at maximum speed while the game is being played, with keyboard inputs it is difficult to reduce drastically the velocity as you cannot measure how strong you break, it is just a boolean breaking or not. In order to fix this, another way to break is introduced, as the same break movement previously explained but with a higher breaking power.

#### Bounce

As the car is not supposed to get out of the track, this method runs when the car collides with the track border so it will be bounced back at a certain speed so that it is

impossible to exit the circuit.

All these methods just modify the velocity of the car, so, after running these functions there is another one which is in charge of moving the car.

**Move**

This function applies the velocity previously calculated modifying the car position, keeping in mind that the car does not only move straight forward but also in certain degree of rotation.

Apart from controlling driving, this class is also in charge of drawing the car at the right position so that the main code is more clear. When the car rotates the image does not change because after many rotations the morphology of the image could be damaged, so, each time the car is drawn on the screen a copy of image rotated is created.

#### 5.3.2 Track

The Track class mainly contains the current used track info (image, borders, initial point, track sectors) and is responsible for drawing the track and its assets.

The arguments that the Track receives are the track number and the initial point of that track. All the tracks are saved as images with the same format, so by only knowing the track name we can get the track image and the track border image manipulating strings.

It is necessary to get the track and the track border independently because one of the requirements is that the car must follow a single path circuit, which means that the car cannot get out of the track limits, we need to mask that border of the track to handle collisions.

Collide function receives the car, the angle of the car, and the mask of the track border. Collisions in pygame are treated through masks, so the first thing that the method do is obtaining the car mask. In order to do this the car image has previously to be rotated at the degree given to the function. Once we got both masks we can see if the images collide in any pixel that is not transparent. It is necessary to do this because when an image is loaded it always has a rectangular shape, but the real image is just the one formed by the colored pixels.

Apart from the track and track border images each track is subdivided into sectors. Each sector set of images is saved in a unique directory, so the way of getting all these sectors is the same as to get the track image, through string management.

The idea of dividing the track into sectors will serve for the Bayesian network as a way to identify the environment where the user is located during the driving. This will be discussed in more depth below.

## 5.4 Game loop

A game, fundamentally, is an infinite loop 3, that in every iteration does a set of steps over and over again. Lets explain how the loop runs, and what is done in each step.

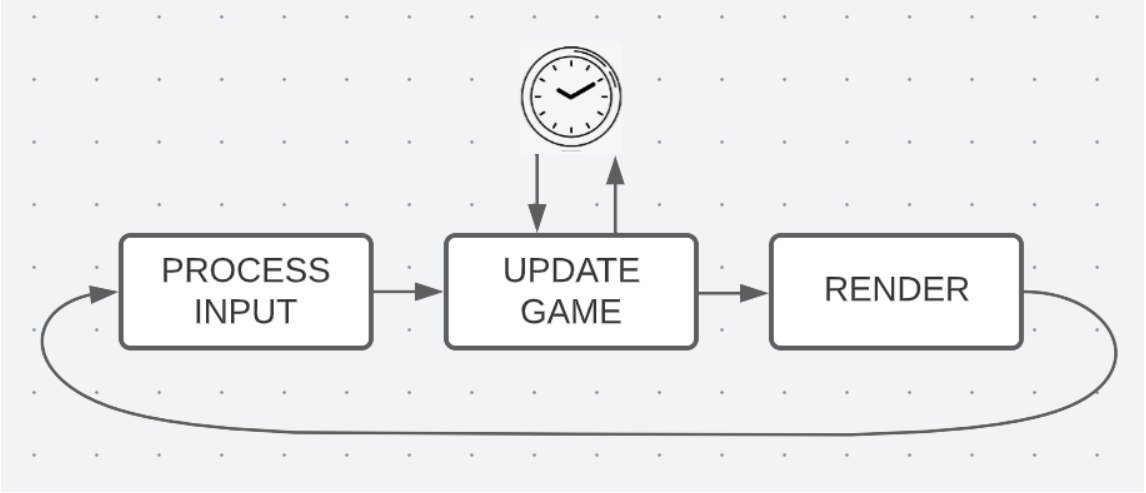



Figure 3: Game loop

One step is processing inputs, were we catch the user interaction with the game, where we see if any key has been pressed, or if the user have moved or clicked the mouse.

Any user action has effect on the game, so in the next step, the game state is updated with the actions of the user as well as the normal functioning of the game. In this stage we need to make all the checks, if there has been a collision, if the driver has ended a lap, if the user changed the color of the car, and more.

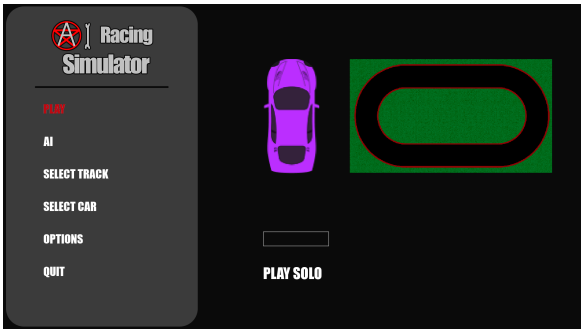
After this two steps, when all the game has changed according to a regular execution and all the inputs, the game screen is fully updated so that the user can see in which state the game is.

Of course, this loop can not run constantly so we need to set a clock that will determine the clock ticks were the game is gonna be executed.

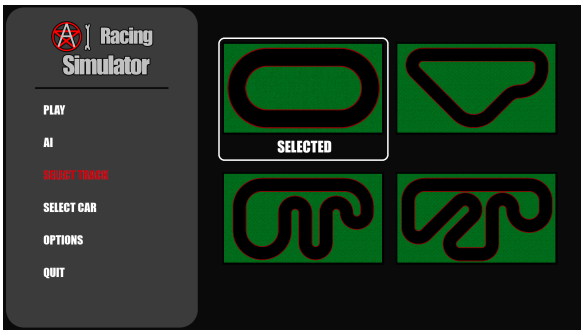
## 5.5 User Interface

When the game is opened the first thing you see is a screensaver 4a with the logo of the game that is skipped pressing the space button. The main goal of this screen is to present the user to the game and to show the logo in order to please him.

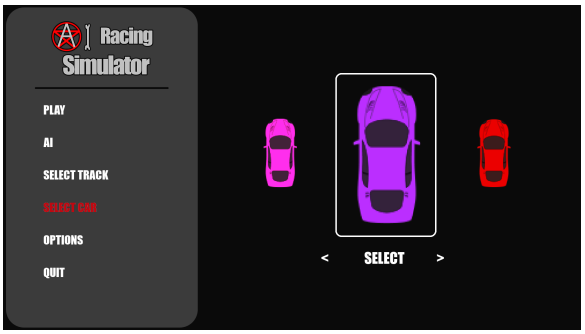

(a) Screensaver



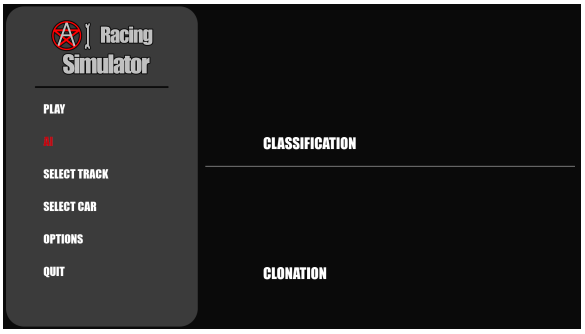

(b) Main menu



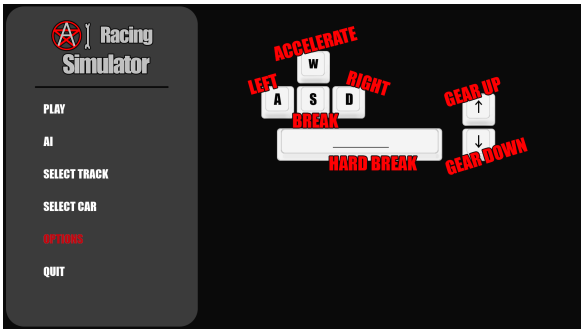

(c) Track selection




(d) Car selection




(e) AI menu




(f) Options

Figure 4: User interface of the simulator, showing the different menus

Once you have pressed the space button you will find a main menu 4b with several options to navigate between the different screens of the interface. First screen is the play menu, there is a preview of the current car and track and an input box to enter the user's name and play.

After that, you have the AI screen 4e, where you can test the functionality of classification and imitation made with the Bayesian networks.

In order to change the car or the track you have two more screens where you can choose between all the cars 4d and tracks 4c available. In the track selection there are four previews of the tracks and you can select which one you want by clicking on the image. On the other hand, the car selection is a kind of slider where the previews of the cars are rotating and you can select a car by pressing the button select.

Finally, there is an options screen 4f where the operation of the keys is indicated.

### 5.6 Track design

As we want to add an environment to the Bayesian network we need to know how is the road in each moment the user is driving. In order to do this, we have designed a circuit system based on independent sectors that are connected in order to form a complete track. Figure 5 shows the shapes of all the sectors available.

Figure 5: Track sectors

Apart from the straight line, the curves are divided into three different types, close curve, normal curve and open curve. Each curve can have a 180º, 135º, 90º, 45º graduation.

After creating the track connecting all the sectors it is necessary to extract the track borders in a different image so it will be easier to determine whether the car has hit the limits or not.

In order to create a more beautiful and more informative interface, in the bottom right corner we have designed a driver's panel so that the driver can see some information about the driving; velocity, gear, revolutions, lap time and last lap time.

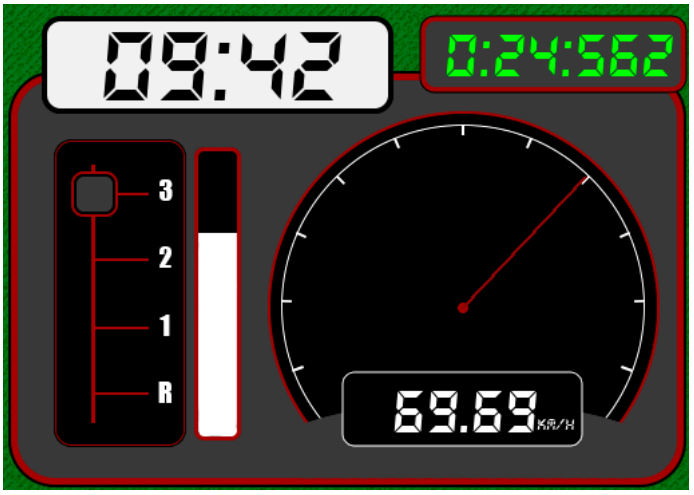



Figure 6: Driver panel

This graphical panel has one digital and one analogical speed meter, a gear lever and the tachometer and two timers for current lap time and last lap time. It is shown in figure 6.

## 5.7 Logging

When the user ends a lap all the information from that lap is saved in a text file. This file shows a tuple of information for each trace taken every hundred milliseconds.

The tuple has this format:
(time, key_a, key_d, key_w, key_s, key_space, gear, sector, curve, direction)

- time: Integer = current lap time
- key_a: Boolean = whether the key "a" is pressed or not
- key_d: Boolean = whether the key "d" is pressed or not
- key_w: Boolean = whether the key "w" is pressed or not
- key_s: Boolean = whether the key "s" is pressed or not
- key_space: Boolean = whether the key "space" is pressed or not
- gear: Integer = current gear
- sector: Integer = type of the sector
  - 0 = straight line

  - 1 = close curve
  - 2 = normal curve
  - 3 = open curve
- curve: Integer = degree of the curve
  - 0 = straight line
  - 1 = 45º
  - 2 = 90º
  - 3 = 135º
  - 4 = 180º
- direction: Integer = direction of the curve
  - 0 = straight line
  - 1 = left curve
  - 2 = right curve

### 5.8 Keeping records

As a way to motivate users to play this game in order to get the logs and develop the network, the game records all the lap times from every user. These times are kept in a JSON file, one file for each track.

Each JSON file contains a python dictionary with this information:

- name: String = name of the driver
- best_time: Integer = best lap time (ms)
- all_times: List = all lap times of this user

This information is not only to track the users, but also allows them to see if they have improved his best time or made the best time of all during the drive. As can be seen, in the driver's panel image the top right time represents the last lap time. The time is shown in a color representing the last lap time over the others. Following a color coding, the same as in the formula 1, if the time is red, you have not beaten your previous record, if the color is green, you have done your personal record, and if the text is purple means that you have done a better lap time than all the other users until now.

### 5.9 Tests

First of all, we must differentiate between:

- Defect: Feature of a system able to cause malfunction. Something which is wrong in the code. Affects the quality of the project.
- Failure: During execution, something that has gone wrong. Affects the reliability of the project.

- Error: Caused by the user.

The testing is done in order to find defects, being almost impossible to detect all of them because of the amount of tests and resources that should be taken in order to do that. When a test detects a failure, it is successful.

The software testing phase consists of dynamically testing the behaviour of a program for a finite set of test cases, comparing it with the expected behaviour in order to evaluate its quality and improve it, identifying possible defects and problems. The main objective is to detect defects, because it is impossible to determine the total absence of faults. The testing phases are divided in four different sets that will be seen here after.

### 5.9.1 Unit testing

This testing is applied to isolated code pieces and is used to verify the perfect functioning of the code. They are usually done by the developer, and drivers and stubs are often used for this purpose. In this case none of them were available so they were done in raw.

Through the development of the code this testing was done all the time in every step forward. Due to the lack of other resources to do the unit tests, they were done by using the function "print()". With this function the developer can evaluate every value taken through the terminal output.

### 5.9.2 Integration testing

In this case, this testing is applied to a whole group of code. The idea is to verify the interaction between different components of the system. The usual mistakes found by these tests are a bad use of the interface or lack of synchronisation between variables.

Once the code was enough developed and it had a functional interface this testing could be done. The testing mainly consisted on navigating through the application menus looking for bugs and the most important thing was to check that the vehicle was properly driven (collisions with the track border, velocity, gear change, etc).

The most important flaw found in this phase was that the collision of the car with the track border was not pixel perfect. This was because when measuring the overlap between the car and the track border layer, the car one was not rotated to the actual angle of the car. As a result, the collision was evaluated with the raw image resource of the car.

### 5.9.3 System testing

This time all the system is involved in the testing at the same time. All the behaviour of the system is tested taking more account of non-functional requirements. In this case the developer is not usually in charge of this work.

For this testing all the implementation of the application was made available to a small

group of users so they can play with the simulator and test whatever they want. During their trials they were able to find some mistakes which were communicated to the developer. They were mainly little failures.

#### 5.9.4 Acceptation testing

Once everything was tested and we have a final version of our project it could be released to the client. In this phase the main goal is not to find mistakes but to seek the customer's acceptance.

In order to release the application it was exported to an executable file so that all the users can execute the application in his windows devices. This file was sent to all the users keeping in touch with them to hear about their acceptance and possible changes and updates suggested by them.

# 6 Classification

Once the simulator is done, the last part of the dissertation is to develop some imitation learning models which will identify and replicate the movements of the users. This development is based on dynamic Bayesian networks with latent variables which will be explained here below [8].

## 6.1 Representation

A Bayesian network is a probabilistic graphical model (PGM) [3]. The main idea behind these networks is to represent joint probability distributions, in a graphical way by using directed graphs.

In probability, the chain rule is used to compute the joint probability in a set of events occurring as:

$$p(x_1, x_2, ..., x_n) = p(x_1)p(x_2|x_1)p(x_n|x_{n-1}, ..., x_1)$$

The chain rule is applied in Bayesian networks, according to the relations between nodes defined in the graph. This makes each node only dependent on its parent nodes. This means that if we have, for example, six variables, and we know the parent nodes of the variable $x_6$ are $\{x_1, x_4, x_5\}$ the probability distribution for that variable will look like this:

$$p(x_6|x_1, x_2, x_3, x_4, x_5) = p(x_6|x_5, x_4, x_1)$$

For each variable $x_i$ there is always a set of parent nodes variables $x_{Ai}$.

In short, we will have several factors $p(x_i|x_{Ai})$ that will generally be represented as probability tables. This happens only when the variables are discrete, which means that they represent categories, i.e., variables contain an integer within a numerable set. But this is what usually happens. The table rows represent all the combinations of the ancestor variables values $x_{Ai}$, and the columns are the values of the actual variable $x_i$. Inside each cell of the table we will have the probability for every case. Let us see this table with a classical example.

Imagine that we have this set of variables:

- $x_r$ that represents if it is raining or not
- $x_s$ which represents whether the sprinklers are on or off
- $x_w$ that symbolize if the floor is wet or not

The variables $x_r$ and $x_s$ are the ancestors of the variable $x_w$. So the probability of $x_s$ is given by the values of the other two variables, that will be the entries of the table. The final table will look like this (the odds are made up).

| $x_r$ | $x_s$ | $x_w$ | |
|---|---|---|---|
| - | - | 0 | 1 |
| 0 | 0 | 99/100 | 1/100 |
| | 1 | 1/10 | 9/10 |
| 1 | 0 | 1/10 | 9/10 |
| | 1 | 1/100 | 99/100 |

Table 1: Probability table: $p(x_w|x_r, x_s)$

Now that we know how this type of nets work, let us see how their graphical representation looks like. Bayesian networks are represented by a directed acyclic graph (DAG). A DAG is formed by nodes connected by directed edges and this type of graphs do not contain cycles, i.e. while traversing a DAG, it is not possible to visit a node that has already been visited.

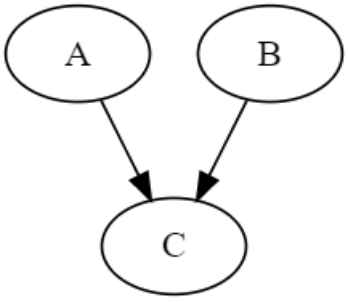



Figure 7: Bayesian network example

Figure 7 shows an example of a Bayesian network, the nodes represent the variables we are going to handle. These variables are named with a capital letter and the lower letters are reserved for the values these variables take in certain case.

Furthermore, the edges represent the relationships between the different variables, meaning that the variable $C$ depends on the variables $A$ and $B$. However, this nodes, $A$ and $B$, do not depend on the the variable $C$ because the relations are one-sided, therefore, as we already know, the network is directed.

Each node will have their own probability table as we have seen before, representing for each variable their probability distribution. The joint probability distribution over these three variables according with their relations will be the shown here below:

$$p(A, B, C) = p(A)p(B)p(C|A, B)$$

The probability tables may not be known at first. If they are unknown they can be generated after training the net with a data set.

Formal definition of a Bayesian network:

$G = (V, E)$, where

- $G$: directed graph
- $V$: set of nodes, for each node $i \in V$ the network has a variable $x_i$
- $E$: set of directed edges between nodes

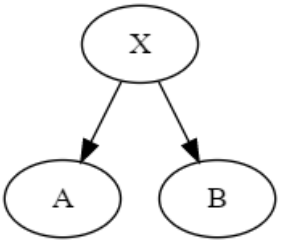
For each variable there is a conditional probability distribution (CPD) which determines the probability of $x_i$ depending on their ancestors.

As we have seen so far, the Bayesian networks represent the probability distribution for a set of variables, forming smaller conditional probability distribution by applying this only to the variables each one depends on. Because of this, it is important to distinguish the independencies between nodes.

To idenfity the set of all independencies we will use the notation $I(p)$ for a joint distribution $p$. For instance, if we have the probability distribution determined by the DAG of the figure 7: $p(A, B, C) = p(A)p(B)p(C|A, B)$, we can say that $A \perp B \in I(p)$. Which means that the nodes $A$ and $B$ are independent.

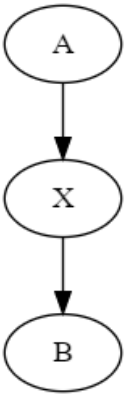
In order to determine all dependencies let us see the three main types of structures in a three-node DAG. With this we will be able to solve all kinds of dependencies in larger graphs.

- Common parent



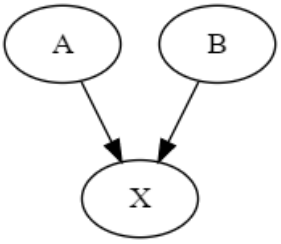

This is the same case as in the DAG we saw in figure 7, but there is another case that we must consider, if the node $X$ is not observed then $A \not\perp B$. This is because being not being observed $X$ will contain the only information that is applied to the nodes $A$ and $B$.

- Cascade




In this case, $A \perp B|Z$ as $Z$ is observed, otherwise $A \not\perp B$. Following the same reasoning as in the previous case, the information of $Z$ will influence the information of $B$, but $A$ is never involved in this operation.

- Common child.




Unlike the previous cases, here $A \not\perp B|Z$ with $Z$ observed. But, when $Z$ is not observed, $A \perp B$.

The main objective of our Bayesian network is to infer the thinking of the driver at every moment of the driving. This means that our system must evolve over time as a dynamic Bayesian network (DBN). A DBN represents the distributions over continuous time, which means that the variables previously mentioned will be determined by a temporal instance t.

First thing we have to do to build a DBN is to discretize time, to achieve that we divide the problem into little time slices $\Delta$. Now, the variable $A$ is represented by $A^t$ meaning the value of $A$ at time $t\Delta$, so we have multiple copies, one for each time. Finally, we have to build a representation that is able to represent the probability distribution for all the variables at every time instance.

Another thing we must take into account are latent variables. This type of variables are used to represent abstract concepts that cannot be measured. For example, talking about an student, we can represent his grades or IQ score as a number, but his intelligence as an abstract idea cannot be represented with a number because there is no

possible way to measure all his capabilities.

This variables are also represented in the DAG but as hidden, i.e. it has their own node and all the necessary relations but they will not have a value during training, the value will be generated through the other variables.

We came to the conclusion that we should use this method through the paper [Ontañon et al.]. Thus we can see in Figure 2 how they keep the abstract idea of the agent's thinking in every moment as a latent variable.

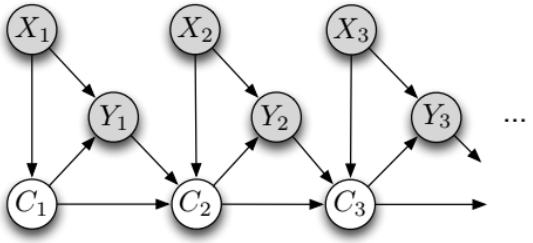



Figure 8: Model from [5]

This is the main concept that we will take into account when developing our network. As you can see, we have the nodes $C_i$ representing the thinking of the agent every time slice reaching $C_{i+1}$ as the moment when the actions ended. These nodes are connected to the next one directly, and they are affected by the environment $X_i$ and the actions of the agent $Y_i$. These two type of nodes are visible while the $C_i$ nodes are not.

With the idea of classifying intelligent agents, we will have a variable $C$ which represents the agent. This node connects to the $C_i$ nodes, which follow the scheme defined above. But, in order to classify, our idea is to identify the driving patterns using only the user's actions on the steering wheel and accelerator. So let's dispense with the node representing the environment.

Figure 9 represents how our model will look like.

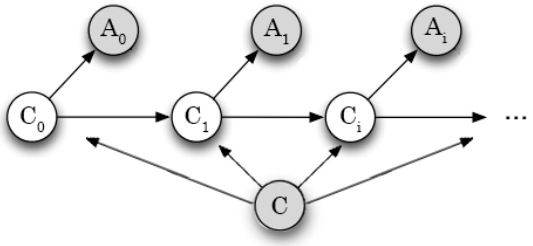



Figure 9: Model shown as a DAG

This Bayesian network could be applied to as much users as you want, but, the more users, the more number of possible values for $C$, and this increases the complexity and the compute time. Furthermore, using this net as an identifier of all the users at the same time could make the precision lower. For example, if we have a sample of 100 users the network have to split the probabilities between all of them, so there might be many of them with very similar probability.

In order to fix this, we have decided to create a different network for every user, the same as in the original model, but the variable $C$ has a cardinality of $2$ instead of $4$. Each net is "owned" by a user, and will determine if the user is the owner of the net or not.

## 6.2 Learning

Once we have designed and understood how our network works, it is time to train it. Normally, a model is trained by giving it information of every variable. But in this case this is not possible as we have latent variables, which are unknown. Because of this, the usual learning methods cannot be used. Instead of those, we will see the main training algorithm to work with latent variables, the Expectation-Maximization algorithm (EM).

Before starting to describe the algorithm let's first look at the formal definition for a latent variable model (LVM). The probability distribution for this type of models is written like this:

$p(x, z; \theta)$, where

- $x$: observed variables
- $z$: latent variables
- $\theta$: CPDs

Now, let's take a look at how the EM algorithm works. Basically, the algorithm performs a two-stage loop, first estimating $\theta_t$, and then computing $p(z|x)$ to infer the values for $z$, until convergence is reached.

The way to compute $p(z|x)$ to get $z$ is by computing the expected log-likelihood

$$E_{z \sim p(z|x)} \log(p(x, z; \theta))$$

This expectation gives part of the name of the algorithm, and can only be executed if $z$ is not too high-dimensional.

Once the expectation is done, it is time to maximize the results of the log-likelihood. This is done again and again until the algorithm converges by a given tolerance.

Now that we know how the algorithm mainly works, let's describe its execution step by step. We consider D as the dataset.

First, the algorithm chooses randomly a $\theta_0$ from which the expectation will start. After that, the Expectation-step is made, where for all the possible values for $x$ it is computed $p(z|x;\theta_t)$. Then, among all these calculated values, the Maximization-step is done, from which the best value for $\theta_{t+1}$ is chosen. The formula is shown here below:

$$\theta_{t+1} = arg\max_{\theta} \sum_{x \in D} E_{z \sim p(z|x)} log(p(x,z;\theta))$$

This learning algorithm has been applied to all the nets in our system using 75% of the data. For each net we have to modify the value of the original data for the variable $C$. Originally the variable $C$ keeps the information of which user is the owner of that trace having a cardinality equal to the number of users. As we have a multi-net system we need to adapt the data for each net, so that the variable $C$ will have a value of $1$ if the owner of the trace is the same as the owner of the network, and $0$ otherwise. Thus, each trace will be different depending on the Bayesian network it is being applied to.

The computational difficulty comes mainly from the learning phase as it is the stage where the Bayesian network goes from a simple DAG to a functional system able to operate with the data given. So, it is important to save the resultant CDPs for each variable as they are responsible for the model's ability to predict. Thanks to this, it is possible to load them, reducing the computational complexity almost to $0$.

### 6.3 Inference

After the process of training, it is time to apply the rest of the data to the trained model in order to infer and examine the results. The inference is the process of predicting some variable values given an evidence to a trained system.

In our case we are going to use 25% of the data we have not used in the training phase. The variable we want to predict is $C$, which identifies the driver, so that variable will be erased from the dataset and the rest of the data will be the evidence that the model will get.

The algorithm we have used to make the inference is the Variable Elimination (VE). As we already know, the Bayesian network represents a probability distribution that can be seen as a product of factors, one for each variable in the net. This algorithm is a two-step loop, first it does the product operation, and then the marginalization. In the first step all the factors are multiplied, and in the second step one variable and its factor are eliminated and replaced by $\tau$.

Our model has observed and unobserved variables, as we know, the unobserved ones cannot be provided in the evidence as they are calculated during the training and have a set value. On the other hand, the observed variables are the user inputs and the driver identifier $C$. As the idea is to predict the user, the only variables that will form the

evidence are the user inputs of every time slice.

This evidence will be given to every network of our system, as we said before, one for each user. The application of the VE algorithm will give us for each net the probability of being or not the user owner of that net. So, the user predicted by this system will be the owner of the net who has the highest probability of being the owner among all the user nets.

For example, imagine that we have a system for three users ($u0$, $u1$ and $u2$), formed by three nets ($bn0$, $bn1$ and $bn2$). After applying the inference we will have this type results for each trace:

Results for $bn0$:

- $P(C = 0|trace0) = 0.8$
- $P(C = 1|trace0) = 0.2$

Results for $bn1$:

- $P(C = 0|trace0) = 0.23$
- $P(C = 1|trace0) = 0.77$

Results for $bn2$:

- $P(C = 0|trace0) = 0.95$
- $P(C = 1|trace0) = 0.05$

Analyzing these results we will get that the user owner of the trace $0$ is $u1$, as his net has the highest probability of $C = 1$, which means that the owner of the net is the owner of the trace.

# 7 Experimentation

In this section we will see how the theory explained in previous sections is put into practice. As you know, we have two different approaches. Firstly we wanted to develop a classification system to identify intelligent agents.

First of all, let's take a look at the users who have participated in the tests, i.e. from whom we have obtained the traces. This information may be useful for analysing the results later on.

For this test section, we have had four different users. From each user and track we have 20 traces. This users were chosen because they have very different driving style, so they should be clearly differentiated.

- User 0

This user is not a person as such, but a user in which a chaotic driving mode has been used, in order to have a reference of something far away from the rest.

- User 1

In this case, this is a person. His driving style is based on long strokes with sustained movements over time. The driver is mainly characterised by constant steering wheel movements when cornering.

- User 2

In contrast to the previous user, this driver is characterised by several steering wheel strokes for cornering. Also tends to make the lines very close to the inside edge.

- User 3

This last user has a complete different driving style. This is an inexperienced driver who is simply trying to stay on the track. The movements may vary depending on the situation.

As a reminder, this simulator has been developed to work only by keyboard inputs, so all the user actions are binary, 1 or 0, when pressing or not each key. Apart from this, in every time slice that the inputs have been recorded there is also information about the road the car is riding in at that moment. This is important as for the cloning phase it will not matter the whole track, it will be cloned also in different tracks.

Now that we know everything about the traces used for the learnings we can start analyzing the results obtained.

First thing we are going to review is the development of the Bayesian Network from the user 0. This user should be very easily recognized as his behavior is chaotic and completely unequal to the others.

| User | Trace | $P(C = 0\|trace)$ | $P(C = 1\|trace)$ |
|---|---|---|---|
| 0 | 0 | $3.13e-18$ | 1.0 |
| | 1 | $1.62e-09$ | 0.99 |
| | 2 | $5.62e-18$ | 1.0 |
| | 3 | $8.68e-18$ | 1.0 |
| | 4 | $1.23e-19$ | 1.0 |
| 1 | 5 | 1.0 | $1.19e-29$ |
| | 6 | 1.0 | $5.40e-27$ |
| | 7 | 1.0 | $7.96e-22$ |
| | 8 | 1.0 | $6.98e-24$ |
| | 9 | 1.0 | $5.36e-25$ |
| 2 | 10 | 1.0 | $1.05e-26$ |
| | 11 | 1.0 | $5.13e-25$ |
| | 12 | 0.99 | $2.61e-15$ |
| | 13 | 1.0 | $1.28e-25$ |
| | 14 | 1.0 | $7.99e-24$ |
| 3 | 15 | 1.0 | $1.35e-23$ |
| | 16 | 1.0 | $1.80e-25$ |
| | 17 | 1.0 | $4.55e-24$ |
| | 18 | 1.0 | $9.28e-28$ |
| | 19 | 1.0 | $1.37e-19$ |

Table 2: Results obtained from the $BN0$ to all the traces used in the inference phase. For each trace it is shown the user owner of the trace and the probabilities got from the network.

First of all, let's explain what each row and column in the table means. In the first row we can see the identifiers for the traces used in the inference phase. Each group of five belongs a different user. The next two rows represent probabilities. $P(C = 0|trace)$ refers to the probability for the user of not being the owner of the net. $P(C = 1|trace)$ means the opposite, the probability of being the owner of that net.

Taking this into account we can see in table 2 that, indeed, for user 0 all networks identify him correctly. The chances of not having detected the user correctly are practically 0, so, as expected, this user is easily recognisable. Now we are going to see the results for the next user net, where there should be more divergence.

| User | Trace | $P(C=0\|trace)$ | $P(C=1\|trace)$ |
|---|---|---|---|
| 0 | 0 | 0.99 | $3.22e-05$ |
| | 1 | 0.99 | 0.007 |
| | 2 | 0.99 | $2.00e-08$ |
| | 3 | 0.99 | $1.34e-09$ |
| | 4 | 0.99 | $5.45e-06$ |
| 1 | 5 | 0.03 | 0.97 |
| | 6 | 0.37 | 0.63 |
| | 7 | 0.08 | 0.92 |
| | 8 | 0.34 | 0.66 |
| | 9 | 0.16 | 0.84 |
| 2 | 10 | 0.99 | 0.0007 |
| | 11 | 0.99 | 0.0007 |
| | 12 | 0.99 | 0.003 |
| | 13 | 0.98 | 0.02 |
| | 14 | 0.99 | 0.001 |
| 3 | 15 | 0.99 | 0.01 |
| | 16 | 0.97 | 0.03 |
| | 17 | 0.99 | 0.008 |
| | 18 | 0.91 | 0.09 |
| | 19 | 0.99 | $8.90e-07$ |

Table 3: Results obtained from the $BN1$ to all the traces used in the inference phase. For each trace it is shown the user owner of the trace and the probabilities got from the network.

This time we can appreciate 3 that the results are by far different from the first ones. The results from the user $0$ are still in the same shape, but, when we try to distinguish real users differences are smaller. In this case, when we try to identify the user $1$, who is the owner of the net, probabilities are not so close to $1.0$. Although all the probabilities for $P(C=1|trace)$ are higher than $0.5$, which means that it has guessed correctly, this network presents some not such good results that may be overcome by other nets.

On the other hand, when this net works with traces which are not from the user $1$, so they should be guessed as $C=0$, the success is much higher. In this case the network results have guessed the traces with almost $1.0$ of accuracy.

Now, let's jump to the next results, the $BN2$ from user $2$.

| User | Trace | $P(C=0\|trace)$ | $P(C=1\|trace)$ |
|---|---|---|---|
| 0 | 0 | 0.99 | $8.79e-08$ |
| | 1 | 0.99 | $1.35e-06$ |
| | 2 | 0.99 | $1.06e-11$ |
| | 3 | 0.99 | $1.70e-09$ |
| | 4 | 0.99 | $9.28e-09$ |
| 1 | 5 | 0.99 | 0.002 |
| | 6 | 0.99 | 0.0009 |
| | 7 | 0.99 | $3.67e-06$ |
| | 8 | 0.99 | 0.0002 |
| | 9 | 0.99 | 0.0001 |
| 2 | 10 | 0.0009 | 0.99 |
| | 11 | 0.004 | 0.99 |
| | **12** | 0.99 | $9.67e-05$ |
| | 13 | 0.001 | 0.99 |
| | 14 | 0.0008 | 0.99 |
| 3 | 15 | 0.99 | 0.005 |
| | 16 | 0.99 | 0.0007 |
| | **17** | 0.27 | 0.73 |
| | 18 | 0.99 | 0.004 |
| | 19 | 0.99 | $2.33e-10$ |

Table 4: Results obtained from the $BN2$ to all the traces used in the inference phase. For each trace it is shown the user owner of the trace and the probabilities got from the network.

In contrast to previous networks, this time 4 there have been two failures in the odds. If we examine the results to the traces from user $2$ we can see how the trace $12$ is a complete failure because there is almost $1.0$ of probability of not being the right user. So we can say that this is a mistake from the model. Contrary to the rest of the traces that have been correctly identified almost 100 percent.

As in the previous case, when this net analyzes traces from user $0$, it is a complete success. Moreover, traces from users $1$ and $3$ show the results to the comparisons to the rest of users, which are generally well guessed except for one case. The trace $17$ may be considered as another mistake from the network, as the probabilities for $C = 0$, which should be higher in case of the net has succeeded, are $0.27$. Being less than $0.5$ should be determined as a mistake, but, the final guess is set by the highest $P(C = 1|trace)$ among all the nets, so we still don't know if it is a clear mistake from the model because $0.27$ can still be higher than the guesses from the other networks.

Finally, we are going to see the results for the last net, $BN3$, owned by the user $3$ 5.

| User | Trace | $P(C=0\|trace)$ | $P(C=1\|trace)$ |
|---|---|---|---|
| 0 | 0 | 0.99 | $4.13e-05$ |
| | 1 | 0.99 | $2.19e-06$ |
| | 2 | 0.99 | $1.12e-05$ |
| | 3 | 0.99 | 0.0003 |
| | 4 | 0.99 | $2.99e-08$ |
| 1 | 5 | 0.99 | 0.001 |
| | 6 | 0.99 | 0.0004 |
| | 7 | 0.97 | 0.03 |
| | 8 | 0.99 | 0.002 |
| | 9 | 0.99 | 0.002 |
| 2 | 10 | 0.99 | $5.43e-05$ |
| | 11 | 0.99 | 0.001 |
| | 12 | 0.96 | 0.04 |
| | 13 | 0.99 | 0.0006 |
| | 14 | 0.99 | $8.53e-06$ |
| 3 | **15** | 0.67 | 0.33 |
| | **16** | 0.54 | 0.46 |
| | **17** | 0.97 | 0.03 |
| | **18** | 0.82 | 0.18 |
| | 19 | 0.33 | 0.67 |

Table 5: Results obtained from the $BN3$ to all the traces used in the inference phase. For each trace it is shown the user owner of the trace and the probabilities got from the network.

As in previous cases the results for the traces owned by user $0$, are exactly the same. Almost 100 percent of accuracy so we can conclude that the classification for user $0$ has been perfect.

Furthermore, when classifying the other users that are not user $3$, the results are again quite good, this time there is no possible mistakes, all the traces has been guessed with close to $1.0$ of probability for $C = 0$.

Lastly, this net has not done a good classification to the traces from his own user. Any trace has been classified with a high accuracy as many others before, and, there is only one trace that has been guessed counting all probabilities greater than $0.5$ as a success. One of the traces, the number $17$, is the highest failure with only $0.03$ probabilities for $C = 1$.

In the introduction of the users we said that user $3$ do not have a determined driving style as his driving level is much lower than the others and many laps could be very different from the others. So, the results obtained may has been determined because of that.

## 7.1 Success rate and compute cost

Now that we have seen the development of each Bayesian network, let's take a look at the success rate. In order to do this, we have to take for each trace the probabilities for $C = 1$ from all the nets and compare them to see which user is predicted.

| Trace \ BN | $BN0$ | $BN1$ | $BN2$ | $BN3$ |
|---|---|---|---|---|
| 0 | 1.0 | $3.22e-05$ | $8.79e-08$ | $4.13e-05$ |
| 1 | 0.99 | 0.007 | $1.35e-06$ | $2.19e-06$ |
| 2 | 1.0 | $2.00e-08$ | $1.06e-11$ | $1.12e-05$ |
| 3 | 1.0 | $1.34e-09$ | $1.70e-09$ | 0.0003 |
| 4 | 1.0 | $5.45e-06$ | $9.28e-09$ | $2.99e-08$ |
| 5 | $1.19e-29$ | 0.97 | 0.002 | 0.001 |
| 6 | $5.4e-27$ | 0.63 | 0.0009 | 0.0004 |
| 7 | $7.96e-22$ | 0.92 | $3.67e-06$ | 0.03 |
| 8 | $6.98e-24$ | 0.66 | 0.0002 | 0.002 |
| 9 | $5.36e-25$ | 0.84 | 0.0001 | 0.002 |
| 10 | $1.05e-26$ | 0.0007 | 0.99 | $5.43e-05$ |
| 11 | $5.13e-25$ | 0.002 | 0.99 | 0.001 |
| 12 | $2.61e-15$ | 0.003 | $9.67e-05$ | 0.04 |
| 13 | $1.28e-25$ | 0.02 | 0.99 | 0.0006 |
| 14 | $7.99e-24$ | 0.001 | 0.99 | $8.53e-06$ |
| 15 | $1.35e-23$ | 0.01 | 0.005 | 0.33 |
| 16 | $1.80e-25$ | 0.03 | 0.0007 | 0.46 |
| 17 | $4.55e-24$ | 0.008 | 0.73 | 0.03 |
| 18 | $9.28e-28$ | 0.09 | 0.004 | 0.18 |
| 19 | $1.37e-19$ | $8.9e-07$ | $2.33e-10$ | 0.67 |

Table 6: Success table: green color represents right guesses, and red color represents wrong guesses

As can be seen, although the results for each net shown quite a few mistakes, using a system that has multiple networks and selecting the best results reduces the failures. From the $20$ different traces that were used to classify, only $2$ of them were bad classified, so we have $90\%$ of success rate. Keeping in mind that our experimentation has a rather low number of traces we can assume that our results are pretty good. Perhaps with a large-scale distribution of the simulator, more conclusive results could be achieved.

It should also be borne in mind that a large-scale distribution will increase the training time and the computational cost. In our experimentation we have $4$ different users with $15$ training traces for each user. This has led to a 48 hour training which would grow exponentially with the emergence of new users.

The good thing about this system is that although the training phase may be a bit costly, once the model is trained with a decent amount of data it will develop a very good classification within a few seconds.

## 7.2 ROC curve and AUC

In order to know which network has done a better job, now we are going to get the Receiver Operating Characteristic (ROC) curves for every BN and compare their AUCs. First let's look at a brief explanation of what they are and how they work. As $BN0$ is used as reference and is much different from the others we are going to use it as an example and the other three networks are the ones to be compared.

This methods are applied to calculate the performance of a classification model. Firstly,

every ROC curve is drawn in a chart, and then it is calculated the Area Under the Curve (AUC) for each curve, and the greatest area will define which classification model is better.

To create the ROC curve it is necessary to build the confusion matrix, which is a way to summarize the results of the classification. This matrix is formed by two columns and two rows, which represent the values $1$ and $0$ for the real data and the predictions respectively. Inside the matrix we will find different results for the guessed data as shown in table 7.

| Predicted \ Actual | 1 | 0 |
|---|---|---|
| 1 | True Positives (TP) | False Positives (FP) |
| 0 | False Negatives (FN) | True Negatives (TN) |

Table 7: Confusion matrix

Each cell held the number of predictions that satisfy the features given by each row and column. $TP$ and $TN$ will get the count of the cases in which the model classified correctly, $TP$ for the instances where the model predicted correctly $1$ and $TN$ for when the model predicted correctly $0$. On the contrary, the cells marked in red will held the wrong guesses. $FP$ for the cases when the model predicted $1$ but in reality the value should be $0$ and $FN$ the other way round.
Let's see the confusion matrix for $BN0$ as an example.

| Predicted \ Actual | 1 | 0 |
|---|---|---|
| 1 | 5 | 0 |
| 0 | 0 | 15 |

Table 8: Confusion matrix for $BN0$

As this model classifies perfectly we can appreciate that in the secondary diagonal the values are $0$ because there is no mistakes when evaluating. The $TP$ are $5$ as the $5$ traces from the user $0$ were guessed as $1$ and the $TN$ are the remaining $15$ from users $1$, $2$ and $3$.

In this case, reading the results as $1$ or $0$ is easy as the values got from this net are either almost one or almost zero. But in general cases there must be set a threshold to divide the results in $1$ or $0$. Normally it is set in $0.5$ but when drawing the ROC curve this value changes.

Once the confusion matrix is done we have to define some terms that are used to create the ROC curve. The first of these terms is the sensitivity, which refers to the proportion of cases where the system got the TP right, the True Positive Rate (TPR). The other term is the specificity, which is the proportion of TN, True Negative Rate (TNR).

The ROC curve is represented in a graph where the x axis refers to False Positive Rate (FPR) and the y axis refers to True Positive Rate (TPR).

The TPR or sensitivity is calculated by the next formula:

$$\frac{TP}{TP + FN}$$

On the other hand, the FPR is the contrary of the TNR so it is defined as $1-$Specificity.

$$FPR = 1 - TNR = 1 - \frac{TN}{TN + FP} = \frac{FP}{TN + FP}$$

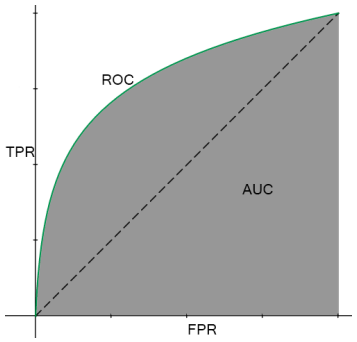



Figure 10: Example of ROC curve and its terms

To draw the ROC curve you have to change the threshold from $0$ to $1$ and for each step it is calculated the confusion matrix. With the values taken from that matrix you will get a value for FPR and a value for TPR. This two values represent a point in the curve. After changing the threshold to all possible values you will have a set of points that form the ROC curve. Let's see a brief example with $BN0$.

The first value for the threshold is $0$, so all the traces are guessed as $1$ because all the probabilities for $C = 1$ are above the threshold. T the confusion matrix that we get for this step is shown in table 9.

| Predicted \ Actual | 1 | 0 |
|---|---|---|
| 1 | 5 | 15 |
| 0 | 0 | 0 |

Table 9: Confusion matrix for $BN0$ with threshold set as 0

With this table completed, we are able to calculate the two values that form the first point in the ROC curve.

$$TPR = \frac{5}{5 + 0} = 1$$

$$FPR = \frac{15}{15 + 0} = 1$$

This two values form the point of the ROC curve, (FPR, TPR) $= (1, 1)$. This point is displayed in the top right corner of the chart as the range is $[0, 1]$ in both axis.

The next value of the threshold is given by the lower value for $C = 1$, letting just $1$ value behind the threshold and the other $19$ after it. The results got from this new parameter are the following.

| Predicted \ Actual | 1 | 0 |
|---|---|---|
| 1 | 5 | 14 |
| 0 | 0 | 1 |

Table 10: Confusion matrix for $BN0$ with threshold set as the lower value for $C = 1$

$$TPR = \frac{5}{5+0} = 1$$
$$FPR = \frac{14}{14+1} = 0.93$$

So the next point in the curve is $(0.93, 1)$. This step is done over and over again until every value is behind the threshold. All the points got are the ones that form the ROC curve 11.

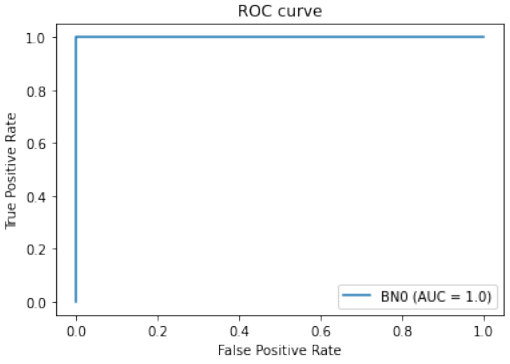



Figure 11: ROC curve for $BN0$

As we could imagine, this net predicts perfectly the user with probabilities very close to $1.0$ and to $0$ so the ROC curve is drawn so that the area underneath it is maximum. As the accuracy of the classification decreases, the curve will move closer to the centre. In case the curve is inverse, i.e. its AUC is less than 0.5, we would say that the classification is being done in reverse.

Now let's apply this method to the other three Bayesian networks to see which one classifies the best.

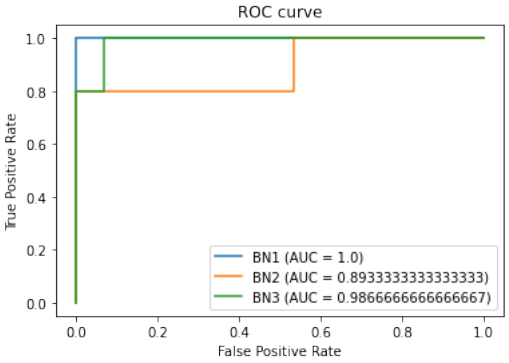



Figure 12: ROC curves comparison for $BN1$, $BN2$, $BN3$

The performance of the Bayesian networks from user $1$, $2$ and $3$ draw three different ROC curves shown in figure 12. To compare each net it is just necessary to compare the AUC from each curve. This will lead to this ranking:

1. $BN1$ - AUC $= 1$
2. $BN3$ - AUC $= 0.987$
3. $BN2$ - AUC $= 0.893$

All this areas are much greater than $0.5$ so all the nets complete a very good classification. This ranking could be applied to decide between one net or another in case of similar probabilities.

# 8 Cloning

The next and last step of this paper was to develop an imitation model which will try to clone the driving style of a user. This stage follows the same system, it is done by using dynamic Bayesian networks (DBN) with latent variables. The model used for cloning is not the same as for the classification, so the principal differences and characteristics are going to be explained in this section.

## 8.1 Representation

In this case, we are going to use the same methodology as in the previous model. The one from the paper [Ontañon] as seen in figure 8. In the classification we did not use the environment variables as we only wanted to track the steering wheel movements. But now, controlling the environment is one of the keys of our imitation model.

Therefore, in the resulting network, we will have the variables $X$ referring to the current environment where the car is placed in each moment. Basically, this node will represent in which kind of sector from the set seen in 5 the car is situated. So, the network will follow exactly the same format as in 8, also removing the node $C$ as now we do not want to guess the user, now we just want to replicate its actions.

## 8.2 Learning

Taking into account that this model also works with latent variables, the training algorithm should be the same as the one used in the previous model, the EM algorithm. Which we already know how it works.

The training process for the cloning model will be a bit different from the other model. Instead of training with 75% of the data from all the users, now the model will be trained with all the amount of traces only from the user we want to replicate. As for this process 20 traces may not be enough we have raised number of traces up to 40 just for the user that will be replicated.

As this net has more nodes than the previous one, the computational cost will be bigger. But this time the system only needs a single network, so we still have low computational cost systems.

## 8.3 Generating samples

Finally, the last phase of the cloning system is to generate traces from the trained model. This time we will not have to use any traces, the own trained model will create different samples that we will compare to the originals and see if the driving style is similar or not.

## 8.4 Results

Hereafter we are going to check the development of the cloning system defined in this section. After generating the samples, the first thing we can do is to examine the shape of the trace generated to see if there is anything unusual. The first thing that is striking is that, in many moments of time, the acceleration and braking are simultaneous. This is something strange as in a normal lap there is no need to brake and, above all, a driver never accelerates and brakes at the same time.

| Time | w | s | a | d | space | gear | Time | w | s | a | d | space | gear |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 3 | 20 | 1 | 0 | 0 | 0 | 0 | 3 |
| 1 | 1 | 0 | 0 | 0 | 0 | 3 | 21 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 | 0 | 3 | 22 | 1 | 0 | 0 | 0 | 0 | 3 |
| 3 | 1 | 0 | 0 | 0 | 0 | 3 | 23 | 0 | 1 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 1 | 0 | 0 | 3 | 24 | 1 | 0 | 0 | 0 | 0 | 3 |
| 5 | 1 | 0 | 1 | 1 | 0 | 3 | 25 | 0 | 0 | 1 | 1 | 1 | 3 |
| 6 | 1 | 0 | 0 | 1 | 1 | 3 | 26 | 1 | 0 | 0 | 0 | 0 | 3 |
| 7 | 1 | 0 | 0 | 0 | 1 | 1 | 27 | 1 | 0 | 0 | 0 | 1 | 3 |
| 8 | 0 | 1 | 0 | 1 | 0 | 2 | 28 | 1 | 0 | 0 | 0 | 0 | 2 |
| 9 | 0 | 1 | 0 | 1 | 1 | 3 | 29 | 1 | 0 | 0 | 0 | 0 | 3 |
| 10 | 1 | 0 | 0 | 0 | 0 | 1 | 30 | 1 | 0 | 0 | 0 | 0 | 3 |
| 11 | 0 | 0 | 1 | 0 | 1 | 3 | 31 | 1 | 0 | 0 | 0 | 0 | 3 |
| 12 | 1 | 0 | 1 | 0 | 0 | 3 | 32 | 1 | 0 | 0 | 0 | 0 | 3 |
| 13 | 1 | 0 | 0 | 0 | 0 | 3 | 33 | 0 | 0 | 0 | 0 | 0 | 1 |
| 14 | 1 | 0 | 0 | 0 | 0 | 3 | 34 | 0 | 1 | 0 | 0 | 1 | 3 |
| 15 | 1 | 0 | 0 | 0 | 0 | 3 | 35 | 0 | 0 | 1 | 1 | 0 | 3 |
| 16 | 1 | 0 | 0 | 0 | 0 | 3 | 36 | 1 | 0 | 0 | 0 | 0 | 3 |
| 17 | 1 | 0 | 0 | 0 | 0 | 3 | 37 | 1 | 0 | 0 | 0 | 0 | 3 |
| 18 | 1 | 0 | 0 | 0 | 0 | 3 | 38 | 1 | 0 | 1 | 1 | 0 | 3 |
| 19 | 1 | 0 | 0 | 0 | 0 | 1 | 39 | 1 | 1 | 1 | 0 | 1 | 2 |

Table 11: Part of a trace generated by the cloning model

As shown in table 11, the trace does not follow a fixed pattern of progress and has a chaotic behaviour.

Despite the error, it is something that can be rectified indicating during the reading of the trace that you cannot accelerate and brake, and making the car just accelerating. In order to take a deeper look into the development of the system, the simulator is able to simulate a driving from the traces.

This simulation was done and the results cannot be worst. If there is not treatment to fix the accelerating and braking the car is stuck just spinning, and when we treat the accelerating and braking as just accelerating the development is not much better. The car only moves randomly and then it stays stopped.

In order to seek an alternative solution we tried to generate samples with the same network as in the classification section 9. This takes out the variable environment but we do not know if there will be better results. Of course the variable $C$ is deleted from

this net as there is only one driver examined.

| Time | w | s | a | d | space | gear | Time | w | s | a | d | space | gear |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 3 | 25 | 0 | 0 | 0 | 0 | 0 | 3 |
| 1 | 1 | 0 | 0 | 0 | 0 | 3 | 26 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 3 | 27 | 1 | 0 | 0 | 1 | 0 | 2 |
| 3 | 1 | 0 | 0 | 0 | 0 | 3 | 28 | 1 | 0 | 0 | 0 | 0 | 3 |
| 4 | 1 | 0 | 0 | 0 | 0 | 3 | 29 | 0 | 0 | 0 | 0 | 0 | 3 |
| 5 | 1 | 0 | 0 | 0 | 0 | 3 | 30 | 1 | 0 | 0 | 0 | 0 | 2 |
| 6 | 1 | 0 | 0 | 0 | 0 | 3 | 31 | 1 | 0 | 1 | 0 | 0 | 3 |
| 7 | 1 | 0 | 0 | 0 | 0 | 3 | 32 | 1 | 0 | 0 | 1 | 0 | 2 |
| 8 | 1 | 0 | 0 | 0 | 1 | 3 | 33 | 1 | 0 | 0 | 0 | 0 | 3 |
| 9 | 1 | 0 | 0 | 0 | 0 | 3 | 34 | 1 | 0 | 0 | 0 | 0 | 3 |
| 10 | 1 | 0 | 0 | 0 | 0 | 3 | 35 | 1 | 0 | 0 | 1 | 0 | 3 |
| 11 | 1 | 0 | 0 | 0 | 0 | 3 | 36 | 1 | 0 | 0 | 1 | 0 | 3 |
| 12 | 1 | 0 | 0 | 0 | 0 | 3 | 37 | 1 | 0 | 0 | 1 | 0 | 3 |
| 13 | 1 | 0 | 0 | 0 | 0 | 3 | 38 | 1 | 0 | 0 | 1 | 0 | 3 |
| 14 | 1 | 0 | 0 | 0 | 0 | 3 | 39 | 1 | 0 | 0 | 1 | 0 | 3 |
| 15 | 1 | 0 | 0 | 0 | 0 | 3 | 40 | 1 | 0 | 0 | 0 | 0 | 3 |
| 16 | 1 | 0 | 0 | 0 | 0 | 3 | 41 | 1 | 0 | 0 | 1 | 0 | 3 |
| 17 | 1 | 0 | 0 | 0 | 0 | 3 | 42 | 1 | 0 | 0 | 1 | 0 | 3 |
| 18 | 1 | 1 | 0 | 0 | 0 | 1 | 43 | 1 | 0 | 0 | 1 | 0 | 3 |
| 19 | 1 | 0 | 0 | 0 | 0 | 1 | 44 | 1 | 0 | 0 | 1 | 0 | 3 |
| 20 | 1 | 0 | 0 | 0 | 0 | 2 | 45 | 1 | 0 | 0 | 1 | 0 | 3 |
| 21 | 1 | 0 | 0 | 0 | 0 | 3 | 46 | 1 | 0 | 0 | 1 | 0 | 3 |
| 22 | 1 | 0 | 0 | 0 | 0 | 2 | 47 | 1 | 0 | 0 | 0 | 0 | 3 |
| 23 | 1 | 0 | 0 | 0 | 0 | 3 | 48 | 1 | 0 | 0 | 1 | 0 | 3 |
| 24 | 1 | 0 | 0 | 0 | 0 | 1 | 49 | 1 | 0 | 0 | 1 | 0 | 3 |

Table 12: Part of a trace generated by the modified cloning model

Surprisingly, the problem of accelerating and braking at the same time has disappeared 12, and the driving of the car tends to do the correct ride but with some error margin. Performance is still far from what is required.

The only explanation we can give for this situation is that the traces used for training are sometimes from the "first lap" from the car, where the car starts from standstill and that makes the traces different between each other. This was supposed to be fixed with the use of the environment but the performance is not the spectated.

# 9 Conclusions

To conclude this dissertation we will review the initial objectives and whether or not they have been met. We will also review how these objectives have been met and the different problems that have arisen during development.

The first objective was to develop a driving simulator with circuits of varying difficulty that allows to save information about the user drivings. This simulator was developed in python, using the module pygame as the tool to build the game. The way the game stores drivers' driving data is through writing files organised in a directory system. So we can say that this objective was completely succeeded. Perhaps a future work remains to launch a portable version of the game capable of storing the driving data of all users who play the game in the cloud.

The main thrust of this work was the development of a classification system able to identify different users who have previously driven based on simulator records. Despite the fact that we have only a few users for experimentation, we got a 90 percent of success rate when trying to classify. Furthermore, one of our goals was to make it with low computational cost, and we have managed to make the model we took as reference from [5] even smaller 9. This solves our main hypothesis, which is the following:

- Intelligent agents can be recognised from the traces by using very simple models

Finally, the last part of this dissertation was to modify the model adding environment variables in order to make it able to imitate a driver. In this case the results obtained were not the desired ones. In the first approach, the traces generated from the model presented divergences to a normal driving like accelerating and braking at the same time. This make us to redesign the model trying to exclude the environment and even the results were better they did not reach the goal to meet. Maybe, in future works, the use of another type of models like decision trees or neural networks could be applied to imitate the driver actions.

# 10 Bibliography

## References

[1] Mark Beliaev, Andy Shih, Stefano Ermon, Dorsa Sadigh, and Ramtin Pedarsani. Imitation learning by estimating expertise of demonstrators. *arXiv preprint arXiv:2202.01288*, 2022.

[2] Mariusz Flasiński. *Introduction to artificial intelligence*. Springer, 2016.

[3] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[4] Hector J Levesque. Knowledge representation and reasoning. *Annual review of computer science*, 1(1):255–287, 1986.

[5] Santiago Ontañón, José L Montaña, and Avelino J Gonzalez. A dynamic-bayesian network framework for modeling and evaluating learning from observation. *Expert Systems with Applications*, 41(11):5212–5226, 2014.

[6] Statistics times. www.statisticstimes.com top computer languages, 2021.

[7] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019.

[8] Stanford University. Stanford CS228 course on probabilistic graphical models, 2022.