Andreas Wegelius 15168
SAE Institute

TORCS AI DRIVER

Written report on the implementation of an artificial intelligence driven controller to the TORCS environment.

TABLE OF CONTENTS

INTRODUCTION

TORCS (*The Open Racing Car Simulator*) is an open source 3D racing simulator that allows players to take part in racing competitions (Wymann, 2003). In addition to delivering an environment for player controlled racing, the software allows the programming of AI that would control the car and attempt to finish the race as fast as possible without human interference. This report will cover the introduction of an AI *(Artificial Intelligence)* driven controller that will successfully complete races in a competitive manner.

METHODOLOGY

Driving a vehicle properly requires the use of various senses and is a combination of action and reaction. The everyday driver knows how the vehicle they control will react depending on his actions and is constantly surveying the surrounding environment for obstacles and potential dangers and reacting accordingly.

The 'TORCS' application's architecture indirectly provides the user with a variety of sensors, these sensors represent inputs that can be used and interpreted as information that the AI would respond to. In order to develop an intelligent agent (or bot) the user would constantly receive data from the sensors and program the bot to react accordingly via effectors (Loiacono, Cardamone, Lanzi, 2013).

In order for the AI to adequately control the vehicle and navigate the track it is introduced to, multiple techniques are applied which as a whole contribute to a smooth, realistic driven combining **track following**, **object avoidance** and **PID** *(**P**roportional, **I**ntegral, **D**erivative) algorithms.*

- Track following consists of applying a formula that will compute the distance between the car and the centre of the track, and reacting accordingly via **steering**.
- Object avoidance consists of surveying the surrounding environment, analyzing potential dangers and reacting accordingly via **acceleration, steering and braking.**
- PID *(**P**roportional, **I**ntegral, **D**erivative)* algorithms are applied to optimally reach pre-determined quotas (e.g. speed) by manipulating

acceleration and braking.

DESIGN

1. General

For the AI to display a believable and solid driving behaviour, all three “components” (**track following**, **object avoidance** and **PID)** will be computed in a specific order and then the final concluded variables will be returned to the game engine (see Figure 1).

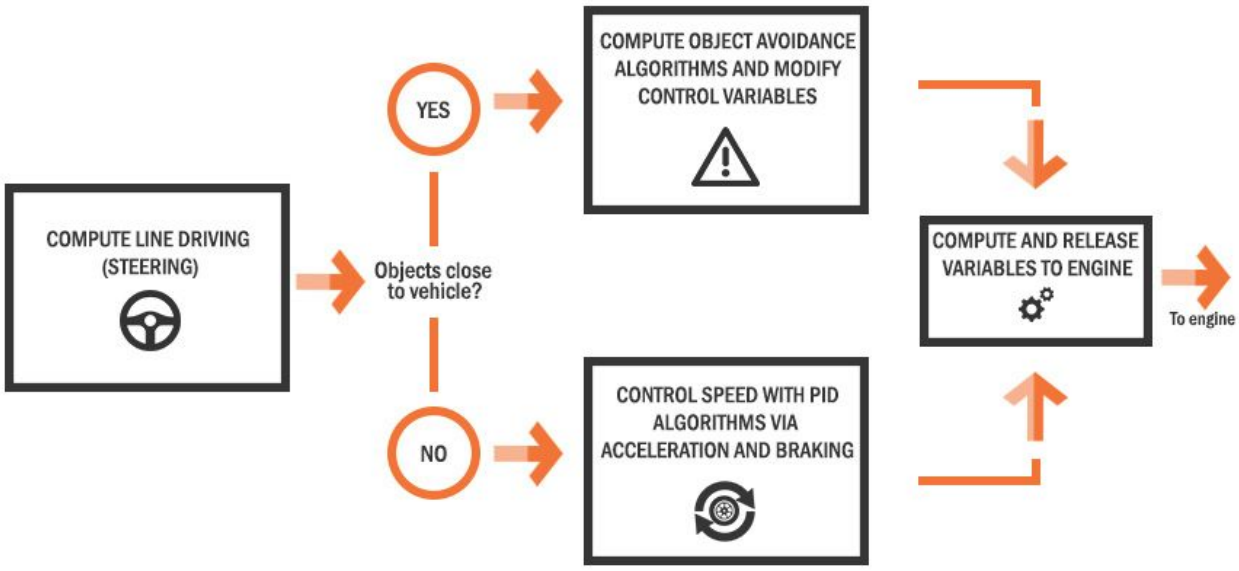


Figure 1. General approach to controlling the vehicle

The first control variable to compute is the steering variable, this will assure the vehicle will remain on the track. It is achieved by deducing the distance between the car the centre of the track, and manipulating a steering variable accordingly.

A check for nearby objects is performed and then depending on the situation, different functions are executed, the object avoidance function or the PID function.

The object avoidance will deduce where the surrounding objects are located and will react accordingly by manipulating acceleration, braking and steering.

The PID algorithm will provide a smooth acceleration and braking approach to control the vehicle’s speed relative to a predetermined speed variable.

2. Line Following

The Line Following function fetches the angle of the car relative to the centre of the

track and the position of the car relative to the centre of the track. The function that returns the angle of the car returns normalized tangent angle value between $-\pi$ and $+\pi$. The initial steering value required to realign the vehicle with the centre of the track is the difference between the car angle and the track position.

3. **Object Avoidance**

In order to alter an AI's driving behaviour and allow it to avoid collision with other objects within the racetrack, the use of sensors will be adopted. The sensors will return various different values in various forms and will subsequently be interpreted by the AI to aid in the reactive process.

Check front of vehicle - Sensors are used to check if an object in close to the front of the vehicle, if so, the AI will release all acceleration, brake slightly to avoid collision and swerve to the side.

Check back of vehicle - Due to the unknown intentions of other drivers, if the back sensors deduce a vehicle behind the car, the AI will accelerate a little bit, whilst still following its initial course.

Check sides of vehicle - If the sensors return a presence on either side of the vehicle, the AI will take countermeasures to swerve slightly away from them.

Check combination of angles - The design of the object avoidance system will based off of all of the available sensors, sensors that point in the same general direction will be grouped up to ease the reactive process (LEFT, FRONT LEFT, FRONT, FRONT RIGHT, RIGHT, BACK RIGHT, BACK, BACK LEFT).

In order to not cause abrupt changes in acceleration, steering or braking, the functions that follow the checks will return additive or subtractive values that will be applied to our pre-computed line driving behaviour (see Figure 2).

Figure 2. Process applied to re-calculating steering for an object avoidance scenario.

4. PID Controller

For the AI to compute an output that controls its acceleration, braking and steering most efficiently, a PID controller will be introduced alongside functions that compute the best possible *setpoint* variables to feed the controller. The setpoint variables are constant variables that the PID uses to determine the margin of error (speed to attain).

The controller is a loop feedback function that calculates an *error* value based on a initial *setpoint* value and a *process parameter* which is fed back into the function. The aim is to minimize the error value by manipulating a *control variable* (Araki, M. 1984).

Compute Ideal Setpoint Value - In order for the PID controller to function, it needs to receive a Setpoint Value (e.g. Max Speed). This value is computed based on the scenario (obstacles incoming, turns incoming etc..).

Compute Control Variable - The PID controller will compute an output variable that will be passed to the CarController deducing whether to accelerate, decelerate and so forth.

Release Control Variable and return error. - This final stage of the PID loop will release the Control Variable to the CarController and return the *error* back into our PID loop and proceed to re-compute the Control Variable to minimize the next *error*.

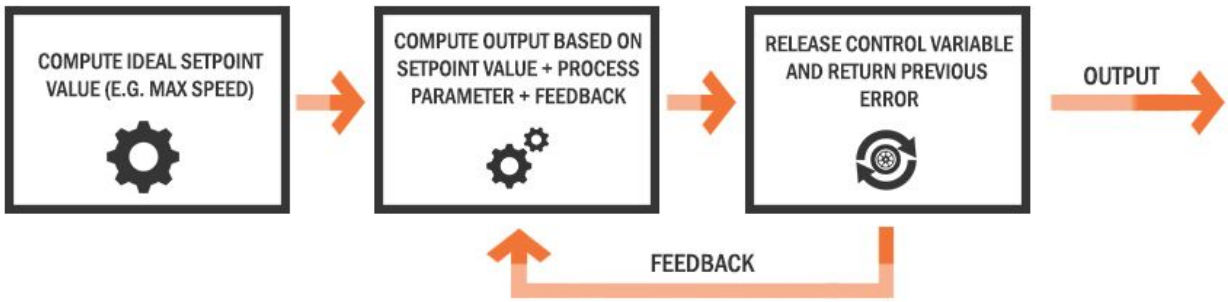


Figure 1. Process applied to calculating an output based on various variables.

IMPLEMENTATION

1. Line Driving

i . Calculate steering

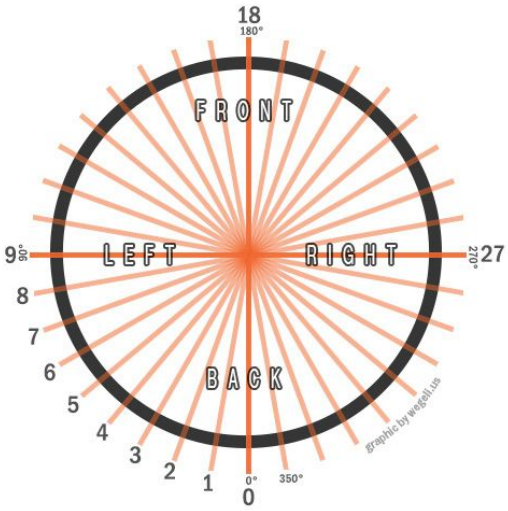
To be able to continuously steer the vehicle towards the centre of the track a function that fetches the position of the car and the angle of the car, both relative to the centre of the track is executed and the track position is subsequently subtracted from the car's angle, resulting in the steering variable that is passed to the engine (Loiacono, 2007).

2. Object Avoidance

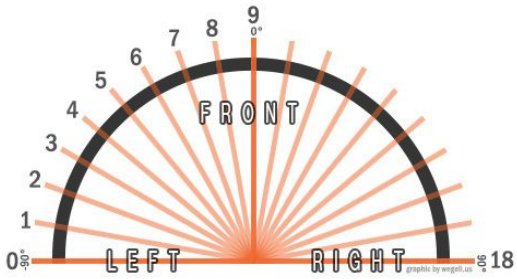
i. Sensors

The sensors that will be used are the ones associated with the "GetOpponents(x)" function (where x represents which sensor). This function returns the distance in metres between the car and the closest opponent's car (if less than 200m) (Loiacono, D, Cardamone, L & Lanzi, P. (2013).

However, due to lack of documentation on which values represent which sensors, the testing and identification of sensors were performed by driving a player-driven vehicle next to our AI to observe which position triggers which directional sensor. The following graphic represents the observations made:



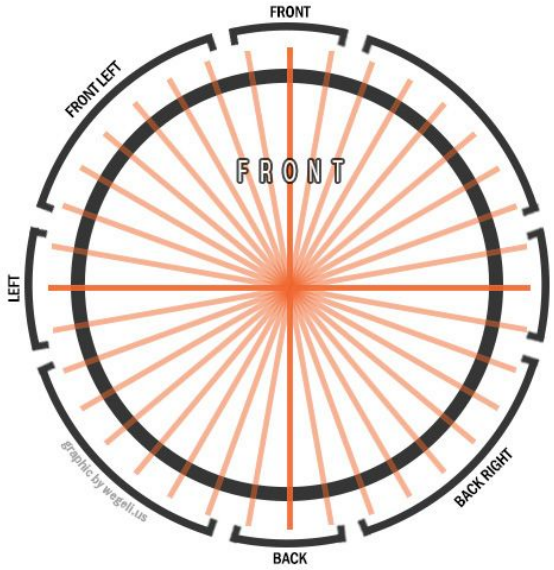
360° “GetOpponents(x)” sensors.



The track-edge detection sensors used for the initial line-driving computation are differently laid out, therefore potentially creating confusion.

ii. Sensors grouping

The reaction associated with the presence of an opponent within similar sensor zones would essentially be the same, therefore it makes sense to group various sensors into their own respective general directions.



The various groups that would return true or false values based on the detection of objects within a predetermined distance from sensors. The grouping allows a cleaner structure to the programming of the object avoidance function, as well as allowing the combination of conditions for the AI to perform more complex behaviour in the future (overtaking, pressure, aggression etc.)

The boolean true and false values of each 8 sensor groups are determined as follows :

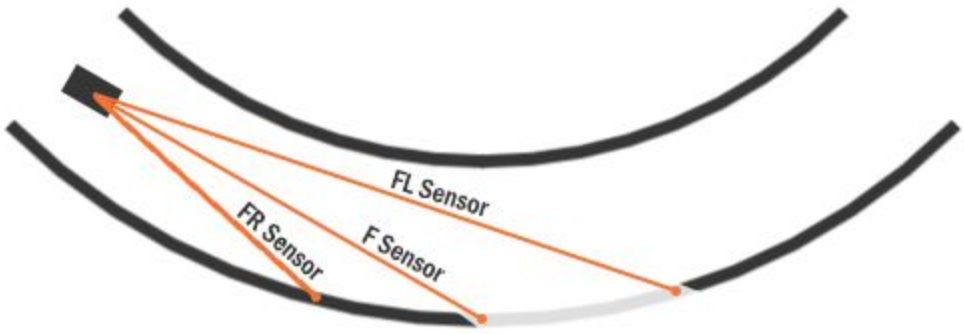
Each sensor belonging to a specific group will be checked, and if any of those sensors returns a valid response to the presence of an opponent within the predetermined distance and the AI car, it will turn a boolean (true/false) variable to true. In this case,

we are checking the three sensors associated with the front of the car (17, 18 & 19, see graphic p.5). If any of the three sensors detect an object less than the predetermined range of 3 metres (modifiable variable) it will tell the program to shift its 'front' boolean to true.

iii. Reactions

If the opponent detection functions return positive values from sensors, the AI will react accordingly; this is achieved by receiving the previously computed line-driving steering and driving behaviour first, then the AI will conclude whether alterations must be made. The alterations to driving behaviour are also considering whether it is safe to steer and to what degree by calling a 'turnDirection' check.

The objectAvoidance() function checks the value of the 'Right' boolean, if it is positive (true), it will react accordingly: braking a little bit and steering towards the opposite direction slightly, in order to create some distance between itself and the opponent. The check of the integer variable 'turnDirection' is performed, this integer represents whether a turn is coming up (-1 = left turn, 0 = no turn, or 1 = right turn), the AI is on a turn or if there is no turn in sight. The AI steers less if close or within a turn, reducing the risk of losing control of the vehicle (See Figure 3).



(Figure 3) If the front left sensor > front sensor -> left turn ahead.
If the front right sensor > front sensor -> right turn ahead.

A 'turnDirection' integer is manipulated depending on whether a turn is coming up within 100 meters, and whether it leans left or right. The simple formula that computes turns and turn directions uses the three forward facing 'distance to edge' sensors and determines the direction based on the distances returned.

The function will first compute the basic line driving variables required to keep the car on the track, as well as the speed it should adopt, the braking intensity, the gear it

should be on and the clutch.

It will then compute the turn: checking whether there is a turn and if so, in which direction.

If an opponent is within the “Right” sensor’s perimeter, the function will check the variable within the ‘turnDirection’ and determine whether the general avoidance variables or the ‘safer’, less abrupt variables should be considered.

The variables chosen are then added and/or subtracted to our basic line driving variables.

These final variables are then released to the engine via a “CarControl” structure.

3. PID Controller

i. Compute setpoints

In order to compute adequate *setpoints* to be sent to our PID function, the constant setpoint variables will be determined based on specific conditions.

For example, the Max Speed variable setpoint is determined based on obstacles ahead or incoming turns (see figure 4).

Figure 4. Simple representation behind the logic of choosing a MaxSpeed setpoint.

In order to compute the severity of each turn, the front car sensors are used. Those sensors return their distance specific distance between the car and the wall ahead.

By subtracting the value of the front sensor (F Sensor) from the front left sensor (FL Sensor), a value representing the difference remains, and this difference value can be

used to determine the severity of a turn where the smaller value would represent a tighter turn (see Figure 5).

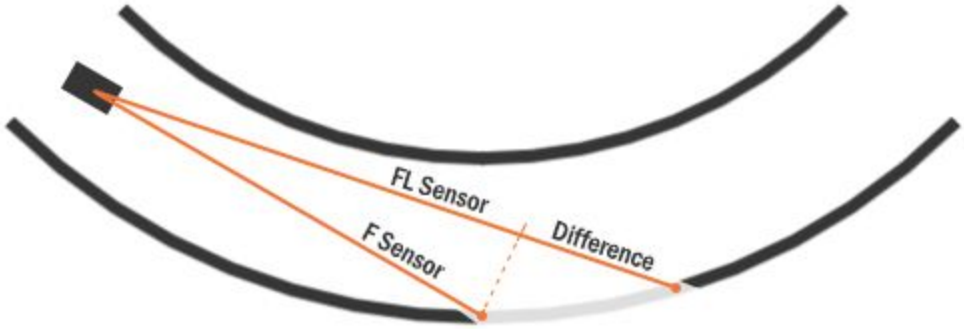


Figure 5. Calculating the severity of a turn (Difference = FL Sensor - F Sensor).

The turn severity is linked to a specific speed, and that speed variable will be passed into the PID controller.

ii. PID control

The PID controller relies on three elements to create an appropriate result: the *proportional* element, the *integral* element and the *derivative* element (Araki, M. 1984).

- *The proportional element account for the current (present) values of the error.*
- *The Integral element accounts for the past values of the error.*
- *The Derivative accounts for the possible prediction of future values of the error.*

The general equation representing the mathematics behind a PID controller is as follows:

$$u_c = K\left[1 + \frac{1}{T_r}\int e\,dt + T_d\frac{de}{dt}\right]$$

PID Equation (Murray, I. 2013)

However, in the case of applying a PID controller to control speed, a discretized version of the equation is adapted, this is required for the computer to comprehend it (http://www.ivoryresearch.com/wp-content/uploads/2013/05/Ian-Murray-sample-paper 1.pdf) :

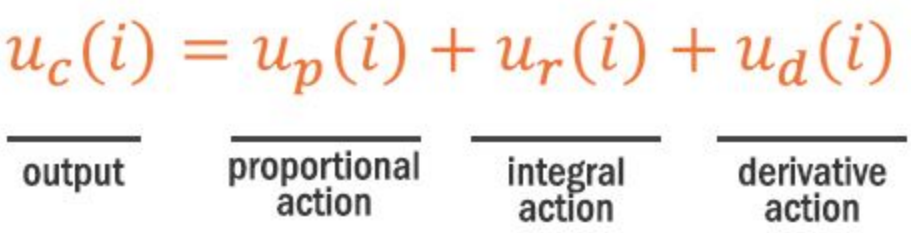


Figure 6. Discretized PID Equation (Murray, I. 2013)

iii. Actions

The final speed output of the PID controller consists of the additive sum of each specific action (See figure 6).
Each action relies on an error value, the error value of the speed PID controller is calculated as following (Murray, I. 2013) **:**

error = SPSpeedX - SpeedX
where SPSpeed is the predetermined setpoint value
SpeedX is the current speed value received via the speed sensor

The proportional action is the sum of the error multiplied by the proportional gain.

Kp* error

The integral action is the sum of the error multiplied by the time differential plus the integral multiplied by the integral gain.

Ki*integral
where integral = integral + error*dt;

The derivative action is the sum of the error minus the previous error (which was fedback through the pipeline, see figure 5.) divided by the time differential.

Kd*derivative
where derivative = (error - previous_error) / dt;

iv. Output

After calculating all of the various actions, the output is released and sent to the simulation, however, the final computed error is fed back into the loop in order to contribute to the computing of another output (see figure 7).

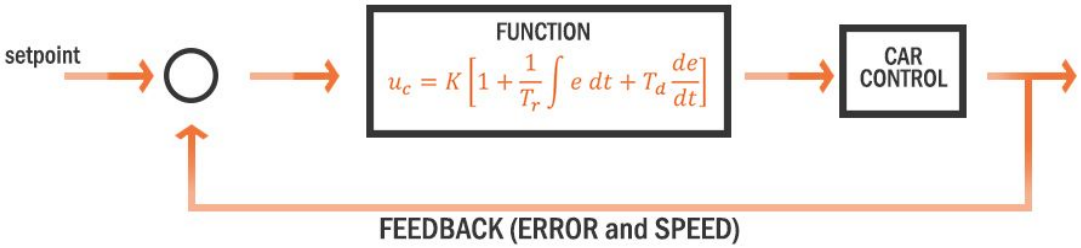


Figure 7. Block Diagram for the control system (Ian Murray more)

CONCLUSION

The combination of the three main components of the AI driver have produced a reliable, competitive driver. The AI drives effectively drives the car around the track and actively avoids other players, occasionally prevents them from overtaking, slows down at tight turns and accelerates at straights. By tweaking the maximum and optimal speed the vehicle would go, the AI would be able to compete more aggressively by taking more risks or drive more safely by taking less.

However, the manipulation of initial variables such as sensors ranges for object avoidance, sensor ranges for turn detection and speed manipulation as well the additive/subtractive steering and acceleration variables within the object avoidance function could contribute to a more efficient driver. The framework is created and only extensive testing and tweaking would optimize the AI ‘s behaviour.

REFERENCES

Wymann, B. (2003) TORCS Robot Tutorial. Available at http://www.berniw.org/tutorials/robot/tutorial.html (Accessed : 01/02.2016)

Edwards, C. (2003) TORCS The Open Race Car Simulator. Available at http://xed.ch/help/torcs.html (Accessed : 01/02.2016)

Competition Software Manual. Available at http://arxiv.org/pdf/1304.1672.pdf (Accessed : 01/02.2016)

Loiacono, D. (2007) SimpleDriver.h Code. Available at SAE Intranet (Accessed : 01/02.2016)

Rudzits, R & Pugeault, N (2014) Efficient Learning of Preattentive Steering in a Driving School Framework*. http://personal.ee.surrey.ac.uk/Personal/N.Pugeault/publications/RudzitsPugeault2014.pdf (Accessed : 20/02.2016)

Loiacono, D, Cardamone, L & Lanzi, P. (2013) Simulated Car Racing Championship Competition Software Manual. Available at http://arxiv.org/pdf/1304.1672.pdf (Accessed : 01/02.2016)

Murray, I. (2013) How to Build a Simulation of a Control System for an Unmanned Automatically Guided Vehicle in C#. Available at http://www.ivoryresearch.com/wp-content/uploads/2013/05/Ian-Murray-sample-paper1.pdf (Accessed : 2.03.2016)

Araki, M. (1984) Control System, Robotics and Automation, PID Control, Vol II Available at http://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf (Accessed : 2.03.2016)