A Fuzzy Logic Based Autonomous Car Simulation in Unity

Justyna Walotek[1], Jagoda Oleksiak[1], Pawel Cebula[1], Adam Stanek[1] and Mateusz Szczypinski[1]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, Gliwice, Poland

Abstract

Our project focuses on using fuzzy logic to make a car drive by itself. Autonomous cars are nothing new nowadays, but the path that we took differs from what is commonly implemented in such cases. When searching for a self-driving car, most of the results rely on a neural network, that learns how to navigate through the track. The issue with this approach is that the network only knows this one particular track and any other track would require training the network all over again. Our idea was to make a car, that would be able to navigate through any given track without the need to learn how to do it. To better describe the technologies that we used first we need to talk about the history of games and AI, then move focus to the more technical aspects of this project, including how the car gathers input and how it is being analyzed, to then cover all of the tests conducted on different car settings consisting of weight, drivetrain and maximum torque, finally reaching our small contest between AI and two different players.

Keywords

unity, fuzzy, car, simulation, game

1. Introduction

To understand the connection between video games and artificial intelligence we need to know how these two came into existence. Let us focus on games first. In October 1958 physicist William Higinbotham made the first video game - Pong. It was a really basic game with the aim to bounce a ball past the opponent. Since then games began to appear rapidly, starting quite simple but getting more and more complex parallel to the advances in the technology available. Suddenly a need for something new appeared - something that would make games more challenging and fun to play. Conveniently around this time, artificial intelligence came in handy, giving game developers a lot more room for creativity. Artificial intelligence started to become an integral part of video games in the 1970s, with the first wellknown game containing AI - Space Invaders. Code of this game analyses player's movement and increases difficulty as time goes on. Another famous game with an AI onboard is Pac-Man, where different ghosts have different approaches for hunting the main character. Today AI in games is far more advanced and takes to account much more variables. For example in racing games, AI is aware of the track pattern as well as weight distribution and powerband of the car. Using this and the knowledge about defensive, offensive, or balanced driving it can be a really tough opponent. In our project we decided to focus on one task: avoiding obstacles. Even though it might sound quite simple, a lot of work had to be put into the code, yet we did not fully achieve it.

ICYRIME 2021 @ International Conference of Yearly Reports on Informatics Mathematics and Engineering, online, July 9, 2021

justwal728@student.polsl.pl (J. Walotek)

2. Related works

Among other works related to the topic of games, game theory, and fuzzy logic, we can mention 'A Game Theoretical Based System Using Holt-Winters and Genetic Algorithm With Fuzzy Logic for DoS/DDoS Mitigation on SDN Networks' [1] where the authors propose a system to faster detect possible attacks based on the denial of service (DoS) using the anomaly detection and identification provided by an HWDS system with an autonomous decision-making model based on game theory. Another worth mentioning paper is 'A fuzzy logic and game theory-based adaptive approach for securing opportunistic networks against black hole attacks' [2] where a security protocol named FuzzyPT is proposed to combat blackhole attacks in OppNets. Fuzziness aids the system in being adaptive by modeling the single nodes as neither benign nor malicious but rather judging a set of nodes based on relationships between different parameters, resulting in a decreased number of false positives and false negatives. Again in [3], the fuzzy-entropy-based game was proposed by the analysis from the perspective of uncertainty. Similarly, a soft matrix game was shown in [4], where another approach of hesitant fuzzy MCDM was used. However, fuzzy logic can be used not only in the Internet of Things - in 'An adaptive self-organizing fuzzy logic controller in a serious game for motor impairment rehabilitation' [5] the authors present ReHabGame, a game that can be easily used by patients and therapists to assess and enhance sensorimotor performance and also help to increase the activities in the daily lives of patients. The different quantities of movement provide fuzzy input from which crisp output is determined and used to generate an appropriate rehabilitation game level, making it a personalized, autonomously learned rehabilitation program. The latest years brought many

interesting approaches to game theory and its applications. For instance, virtual reality [6] was improved by using the convolutional neural networks and many different sensors for increasing the inversion in virtual reality. Machine learning solutions was also applied in many different real scenario [[7, 10]. Games are used for different purposes, not only in entertainment but also in educational areas. It was shown in [14, 16, 17, 12, 13], where fuzzy logic was analyzed in mobile games for students. Fuzzy logic as other areas of artificial intelligence can be used in many areas [18, 20, 21].

3. Fuzzy logic and the power of words

One of the biggest differences between humans and computers is the ability to use abstract concepts to describe reality. We can say for example that something is small, short, blue or that it is far away from us. While doing it, we do not assign a certain value to it, like $2cm^3$ 2 seconds, #007EFF or 100km. Instead, we perceive those values as a scale - a certain range of values can be assigned to a certain word. For example, one person can perceive something as light when it weighs up to 2 kg, while another person may think that 5kg is still light and their scale for the definition of the word 'light' is more like 1-6kg because something under 1kg is perceived as 'very light'. But how does it apply to computers? Computers do not understand abstract concepts, they use exact values to describe things. While in some cases it is the preferred approach, in others it might be a real issue. Imagine using a brake only when the distance between the driven car and the car in front of it is smaller than 1.5m and in addition always using only the full force of the brake. Standing in traffic would be unbearable, would it not? One of the solutions for this type of issue is using fuzzy logic. Fuzzy logic is a type of logic, where input containing exact values is converted into linguistic values called antecedents, which can be then processed using rules, giving us different linguistic values describing the output called consequents, which are then defuzzyfied giving us new exact values for the final output. This may sound difficult or complicated, but actually, it is not that hard to achieve. Using the example of our project, we will try to show this process as simply as possible. Our car measures the distance to the nearest obstacle in 7 directions: to the front, 15°, 30° and 75° to the left, similarly to the right. Next, those falues are being used to calculate the weighted average for 3 directions: left, front, and right. It is done because of two reasons:

1. we decided to have just two output values - horizontal and vertical - which determine if the car goes forward and if it turns in either direction,

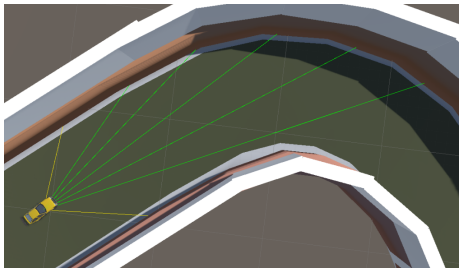

Figure 1: Linguistic values depending on the distance measured

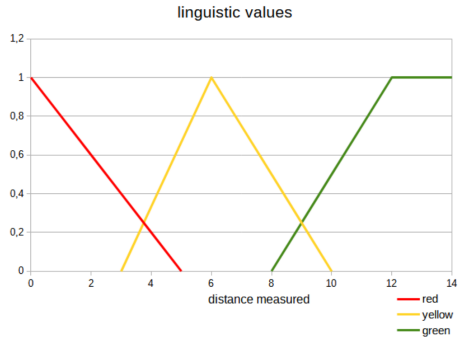

Figure 2: Measuring the distance using raycast in Unity

2. the number of input values determines the count of rules - having 7 input values and 3 possible linguistic values gives us 3 7 possible situations and all of them need a rule. Decreasing the number of input values to 3 the count of rules changes to 3^3 , making it possible to manage and less resource consuming.

Every input value needs to be fuzzified - a linguistic value is being assigned depending on the range the input value suits best.

The mathematical equation (eq. 1) used for fuzzyfying the input:

$$f(x) = \begin{cases} 0 & for\ x \in (-\infty; a\rangle \\ \frac{x-a}{b-a} & for\ x \in (a; b\rangle \\ \frac{x-c}{b-c} & for\ x \in (b; c\rangle \\ 0 & for\ x \in (c; +\infty) \end{cases} \quad (1)$$

In the eq. 1 the values a bvand c depend on the linguistic value the equation is used for. If we write those values as linguisticValue=[a,b,c] then:
red=[0, 0, 5]
yellow=[3,6,10]
green=[8, 12, 100]
This means, that every input value has 3 new values assigned, one for each linguistic value, so for example input=5 would give us 0 for red, 0.6 for yellow, and 0 for

```
//27 rules because each of the 3 rays has 3 options and 3*3*3=27

Rule a = new Rule("R", "R", "R", "front", "back");
//can't go anywhere
Rule b = new Rule("R", "Y", "R", "front", "go");
Rule c = new Rule("R", "G", "R", "front", "go");
//both sides red, but can go forward
Rule d = new Rule("Y", "R", "R", "left", "go");
Rule e = new Rule("G", "R", "R", "left", "go");
//can only go left
Rule f = new Rule("Y", "Y", "R", "slight left", "go");
Rule g = new Rule("Y", "G", "R", "slight left", "go");
Rule h = new Rule("G", "Y", "R", "left", "go");
Rule i = new Rule("G", "G", "R", "slight left", "go");
//can go left-front
Rule j = new Rule("R", "R", "Y", "right", "go");
Rule k = new Rule("R", "R", "G", "right", "go");
//can only go right
Rule l = new Rule("R", "Y", "Y", "slight right", "go");
Rule m = new Rule("R", "G", "Y", "slight right", "go");
Rule n = new Rule("R", "Y", "G", "right", "go");
Rule o = new Rule("R", "G", "G", "slight right", "go");
//can go right-front
Rule p = new Rule("Y", "R", "Y", "trouble", "go");
//can pick a side but can't go forward
Rule q = new Rule("Y", "Y", "Y", "front", "go");
Rule r = new Rule("Y", "G", "Y", "front", "go");
//go forward
Rule s = new Rule("Y", "Y", "G", "slight right", "go");
Rule t = new Rule("G", "Y", "Y", "slight left", "go");
//go to green
Rule u = new Rule("G", "G", "Y", "slight left", "go");
Rule v = new Rule("G", "G", "R", "slight left", "go");
// go left-front
Rule w = new Rule("R", "G", "G", "slight right", "go");
Rule x = new Rule("Y", "G", "G", "slight right", "go");
//go right-front
Rule y = new Rule("G", "R", "G", "trouble", "go");
Rule z = new Rule("G", "Y", "G", "trouble", "go");
//can't go forward
Rule aa = new Rule("G", "G", "G", "front", "go");
//can go anywhere so go forward
```

Figure 3: rules

green. When all of the input values have those linguistic values assigned, we can move on to the rules. When all of the input values have those linguistic values assigned, we can move on to the rule shown in Fig. 3

As we said before, we need 27 rules to cover every possible situation that may occur. Every rule consists of linguistic values for the input and linguistic values for the output. Using values of how well the inputs fit the red, yellow and green range, rule values are being calculated simply by multiplying the values assigned to the linguistic values in the rule. For example rule a consists of all red, so the rule value will be: $ruleValue = leftRedValue * frontRedValue * rigthRedValue$ This way we determine which rule is the most accurate for the given input - we simply take the one with the highest rule value. In our case the simplest way to defuzzyfy was to assign certain values to the consequent values and compute the output by multiplying the rule's consequent and its ruleValue (Fig.4)

```
Consequent left = new Consequent("left", -1f);
Consequent slLeft = new Consequent("slight left", -0.5f);
Consequent front = new Consequent("front", 0f);
Consequent slRight = new Consequent("slight right", 0.5f);
Consequent right = new Consequent("right",1f);
Consequent back = new Consequent("back", -2f);
Consequent stop = new Consequent("stop", 0f);
Consequent go = new Consequent("go", 0.8f);
Consequent trouble = new Consequent("trouble", 1f);
```

Figure 4:

```
Rule result = rules[index];
foreach(Consequent c in cons)
{
    if (c.linguisticValue == result.h)
    {
        horizontal = res*1.3f * c.value;
    }
    if (c.linguisticValue == result.v)
    {
        vertical = (res * c.value+c.value)/2;
    }
    if (result.h == "trouble")
    {
        if (r.distance > l.distance)
        {
            horizontal *= -1f;
        }
    }
}
```

Figure 5:

Finally, as we described earlier we compute the final output - horizontal and vertical values. Later it came to our attention, that those values needed to be slightly modified, so we added some more factors to the multiplication (Fig.5)

This certainly is not the most optimal way to do it, but for our needs was good enough

4. Raycast in Unity

Raycast is a technology that allows you to determine the distance of a casting point projecting from a mesh that has crossed its path (Of course, if this mesh has a collider). In order to properly project a raycast, it is necessary to provide it with a starting point and a direction relative to the object associated with it, for example, a car or a rifle. This technology is successfully used in fps and RPG games to detect whether the target of a specific character is in its field of view and is not obstructed by anything (for example, enemies looking for the main character). In the case of the former, raycast is also very often used to determine where the shot fired from the weapon will hit. Although this method is slowly being abandoned as it is

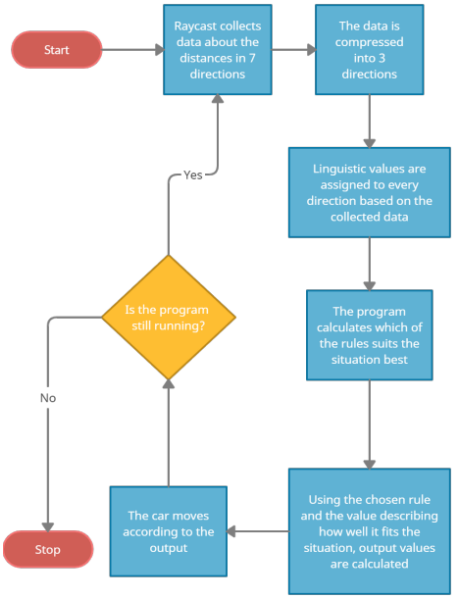

Figure 6: Flowchart of the AI's algorythm

```
motor = maxMotorTorque * v;
steering = maxSteeringAngle * h;

foreach (AxleInfo axleInfo in axleInfos)
{
    if (axleInfo.steering)
    {
        axleInfo.leftWheel.steerAngle = steering;
        axleInfo.rightWheel.steerAngle = steering;
    }
    if (axleInfo.motor)
    {
        axleInfo.leftWheel.motorTorque = motor;
        axleInfo.rightWheel.motorTorque = motor;
    }
}
```

Figure 7:

not realistic enough - in the real world, the projectile's flight path is not a perfectly straight line and additionally, its speed is limited, so where the raycast would register a hit, the real bullet could miss the target if it is moving. Therefore, the gaming industry is slowly starting to use a different method, which is to generate additional bullets with a given initial velocity and mass. Of course, however, there are exceptions where the raycast has an advantage over simulated ballistics. One such exception is dynamic fps games with relatively small maps. After all, the ballistics and the velocity of a projectile at short distances are not that significant, and raycast consumes much fewer hardware resources. This is because it is only active for one frame (short hit test immediately after the shot), while the position of the bullet with active ballistics must be counted until the hit and, additionally, many such bullets can appear in the memory at the same time. In our case, raycast was the best option for gathering data about the surroundings, because it is a simple and fast way to do it. When driving a car we need as little delay as possible, so the simpler and less resourceconsuming method the better.

To make our car move, we needed a simple script and a bit of Unity's physics. We added round colliders into the car's wheels and we by rotating them we move the car.

The car also has a rigidbody attribute, so the Unity's physics engine can apply gravity to it and detect collisions.

It is very unlikely for cars to float in space so we needed to create a track. Our first idea was simple: pick any premade area. This could be enough if the car would be driven by a user, but not for AI because a standard track would not give the AI any information about its' boundaries. The next guess was to upgrade the course for our car, upgraded version should have some kind of barriers to create a path to the finish line. This plan was quickly discarded and the next concept was to create a track using prepared parts like turns or u-turns. The biggest issue with this solution was the fact, that if the substrate was not perfectly flat, AI did not work properly. The last idea, other than forcing our car to fly, was to create a route ourselves from scratch. We decided on recreating one of the most famous auto and motorcycle raceways: Laguna Seca Raceway. To better suit our needs it was shrunk (to spend less time on every lap) and the driving surface was increased to make the turns simpler. The base structure of the road was made in Blender. Due to problems with mesh colliders, the walls are reinforced with additional blocks made from basic cubes in Unity. This way the raycast could finally work properly. There are two turns that turned out to be difficult both for AI and the players.

The left one (a) looks simple at first glance but with a long straight road before the turn where the car can drastically accelerate, it was enough to make the car unable to pass through without hitting the wall. The right one (b) is a u-turn which requires some skill from the players to drive without a collision, so it was too difficult for AI regardless of the car's speed when entering the turn. The title of this paragraph still seems to be unrelated to the topic, so let us explain. Accidentally we recreated one of the scenes from the series 'Initial D', because the car one of the members of our group picked from Unity Asset Store is a model of Toyota AE86 (main character's car) and the turn (a) was also difficult for the characters of this series. Unfortunately AI we created does not drift

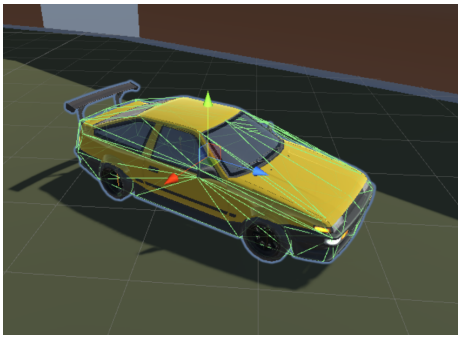

Figure 8: Car with the rigidbody attribute in Unity

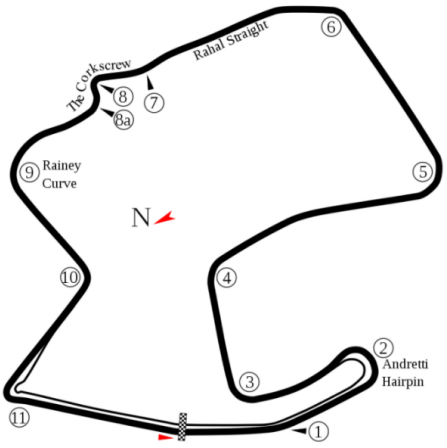

Figure 9: Original Laguna Seca track - layout painted by Alexander Jones

very well. At the moment of writing this article, the model is no longer available, because the package has been deprecated from the Asset Store.

5. Experiments

In order to optimize the vehicle so that it would lap the track in the best time possible and without collisions, e had to conduct a series of tests consisting of changing selected parameters in Unity and checking which configurations would give the best results. Before conducting more constructive tests, we began with random parameters and started testing how MaxTorque and the wheel drive impact a car's ability to maneuver (simultaneously we were trying to find room for the code's improvement).

We found that as the expected time needed to lap the track decreased as MaxTorque increased until the point, where the car could no longer respond fast enough and started to hit a wall. Some of the values stayed unmeasured because we found them insignificant. In the table

Figure 10: Our version adjusted to better suit our needs in Unity

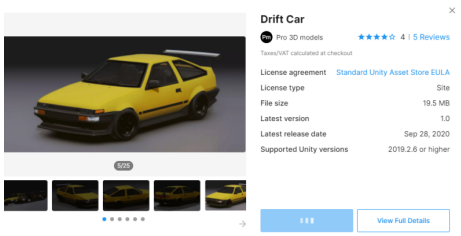

Figure 11: Model of the car in the Asset Store

some of the values are colored, here are their meanings:

- yellow - the best time achieved on those settings,
- orange - the car was really close to hitting one of the walls
- red - the car nudged the wall but it did not stop it for long,
- purple - the car hit the wall and it did have some trouble continuing the lap

With this knowledge we focused on selecting the friction and drag parameters for the physics of the vehicle wheels. Until the expected results were obtained, we modified both the sliding and the static friction. Optimal effects were acquired only when the extreme of static friction was about twice as high as the sliding friction. We also needed to adjust the weights of the wheels and of the car itself to more reasonable amounts. WheelFrictionCurve is used by the WheelCollider to describe the friction properties of the wheel tire. The curve takes a measure of tire slip as an input and gives a force as output. The curve is approximated by a two-piece spline. The first section goes from (0,0) to (extremumSlip,extremumValue), at which point the curve's tangent is zero. The second section goes from (extremumSlip,extremumValue) to (asymptoteSlip,asymptoteValue),

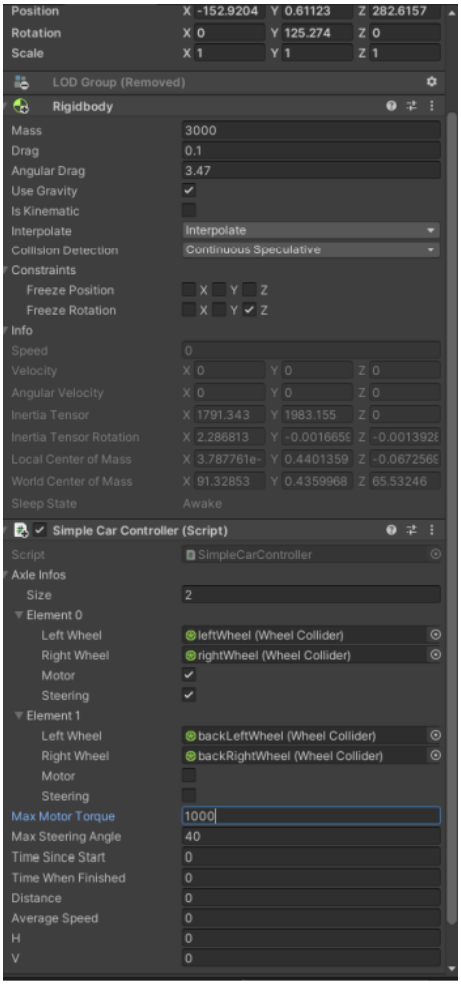

Figure 12: Starting variables in car's attributes in Unity

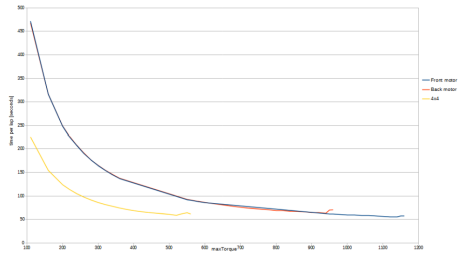

Figure 13: Time per lap depending on MaxTorque value for all wheel drives

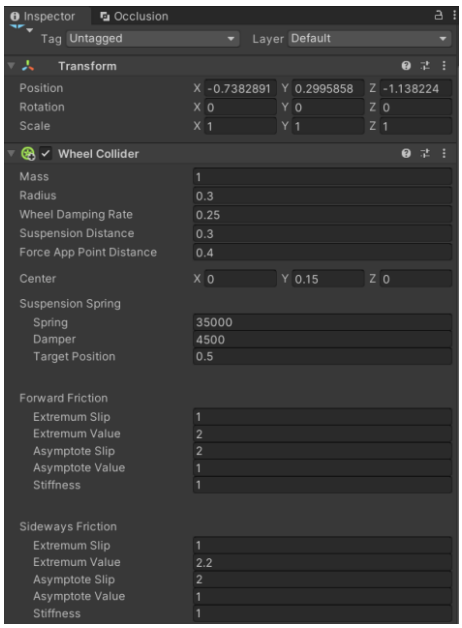

Figure 14: Wheel settings after adjusting in Unity

where curve's tangent is again zero. Wheel collider computes friction separately from the rest of the physics engine, using a slip based friction

model. It separates the overall friction force into a "forwards" component (in the direction of rolling, and responsible for acceleration and braking) and "sideways" component (orthogonal to rolling, responsible for keeping the car-oriented). Tire friction is described separately in these directions using WheelCollider.forwardFriction and WheelCollider.sidewaysFriction. In both directions it is first determined how much the tire is slipping. Then the slip value is used to find out the tire force exerted on the contact. Finally, after making many adjustments to the parameters, we came to the conclusion that the best results were obtained when both of the drag parameters were set to the minimum. Then the center of friction force application acting on the wheels was lowered in relation to the center of mass of the wheel with a radius of 30 centimeters. As a result, we achieved not only the lack of the vehicle overturning but also the curb weight and the engine torque was significantly reduced. After establishing which friction and drag parameters were the most effective, we moved onto the next text in which we wanted to see how changing the maximum torque parameters will affect the time the car will finish the race. The table below shows the dependence of max torque on time. Changing max torque values from 500 to 1000 in steps of 100 with the Front Wheel Drive. Then we decided to check which drivetrain setting would be the most effective for the fastest track completion. In this test, the maximum torque remained 800 for both the front-wheel-drive and the rear-wheel drive, but 400 for the all-wheel drive. It was measured for 3 attempts. The table below shows the dependence of different drive layouts on time

max motor torque	time [s]		
	Front motor	Back motor	4x4
110	471.6593	467.9214	225.2513
160	316.3647	316.3847	154.3362
200	248.6163	249.1164	124.1112
220	225.2313	226.6116	113.7329
240	207.3075	207.8276	104.9344
260	190.2638	191.3441	97.61563
280	176.5609	176.5609	91.59664
300	164.5383	165.0584	86.19755
320	154.3762	154.8163	81.83828
340	145.0742	146.0944	78.05891
360	137.1325	137.7726	74.45952
380	x	x	71.52001
400	x	x	68.90045
420	x	x	66.92078
440	x	x	65.0411
460	x	x	63.72127
480	x	x	62.30124
500	x	x	60.8612
520	x	x	58.94116
530	x	x	61.10121
540	x	x	62.94125
550	91.89659	92.77644	64.4212
560	90.89676	91.57664	61.60122
580	88.21721	88.77711	
600	85.8976	86.43751	
620	x	84.0979	
640	x	81.99825	
660	x	80.05858	
680	x	77.95893	
700	x	76.13924	
720	x	74.41953	
740	x	73.09975	
760	x	71.69998	
780	x	70.60017	
800	x	69.40037	
820	x	69.2004	
840	x	67.46069	
860	x	67.00077	
880	x	66.2209	
900	x	65.0211	
920	x	64.62117	
940	62.44124	63.52126	
950	61.88123	70.24023	
960	61.54122	70.70015	
980	60.6412		
1000	60.04118		
1020	59.52117		
1040	58.66115		
1060	58.10114		
1080	57.48112		
1100	56.5211		
1120	55.98109		
1140	55.48108		
1150	57.34112		
1160	58.02114		

Table 1
Obtained results

max torque	500	600	700	800	900	1000
time	60.6412	57.62113	50.82097	48.66092	49.32094	50.46096

Table 2
Max torque value and obtained time

To summarize, by setting the max torque value to 800, the best time to complete the track was achieved. As for the drivetrain, the front wheel drive turned out to be the best choice, also achieving the fastest completion times. Taking into consideration all the tests performed,

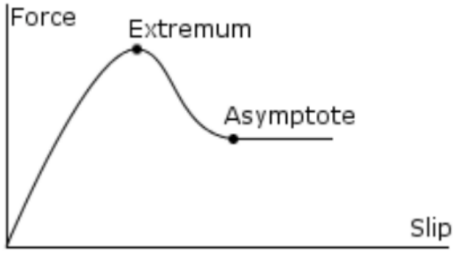

Figure 15: WheelFriction graph

	Front Wheel Drive	Rear Wheel Drive	4 Wheel Drive
Time1	50.72097	55.94109	50.34096
Time 2	49.96095	54.18105	49.72095
Time3	49.64095	56.4411	53.08102

Table 3
Time needed for a different wheel drive

it can be seen that in order to obtain a collision-free test with the fastest possible time to complete the race, the following parameters turned out to be the best choice:

- the extreme of static friction about twice as high as the sliding friction
- both of the drag parameters set to the minimum
- the center of friction force application acting on the wheels lowered in relation to the center of mass of the wheel
- front-wheel drive
- max torque value of 800

6. Conclusion

In conclusion, AI works properly up to a certain velocity. However, after reaching higher speeds, it is not able to respond fast enough, which results in a collision with the wall. The car is also not capable of coping with an obstacle placed closely in front of it, falling into the path selection loop of going slightly forwards and slightly backward. Not surprisingly, when it comes to checking whether the player or artificial intelligence is doing a better job, as long as the best technique to achieve the best time will be drifting, AI will fall far behind the players or even not finish the lap. Despite the difficulties encountered, the following situation was achieved: the car, regardless of its starting point, is able to move and complete the given track without major issues. However, the results achieved by a human driver are still noticeably better than those achieved by artificial intelligence. Ultimately, the chosen goal was accomplished because the car can indeed successfully drive without human intervention and without the need to be trained beforehand.

References

[1] M. V. De Assis, A. H. Hamamoto, T. Abrao, M. L. Proenca, A game theoretical based system using holt-winters and genetic algorithm with fuzzy logic for dos/ddos mitigation on sdn networks, IEEE Access 5 (2017) 9485–9496.

[2] A. Chhabra, V. Vashishth, D. K. Sharma, A fuzzy logic and game theory based adaptive approach for securing opportunistic networks against black hole attacks, International Journal of Communication Systems 31 (2018) e3487.

[3] Q. Zhang, M. Gao, F. Zhao, G. Wang, Fuzzy-entropybased game theoretic shadowed sets: A novel game perspective from uncertainty, IEEE Transactions on Fuzzy Systems (2020).

[4] J. Jana, S. Kumar Roy, Soft matrix game: A hesitant fuzzy mcdm approach, American Journal of Mathematical and Management Sciences 40 (2021) 107–119.

[5] S. S. Esfahlani, S. Cirstea, A. Sanaei, G. Wilson, An adaptive self-organizing fuzzy logic controller in a serious game for motor impairment rehabilitation, in: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), IEEE, 2017, pp. 1311–1318.

[6] D. Polap, K. Kesik, A. Winnicka, M. Wozniak, Strengthening the perception of the virtual worlds in a virtual reality environment, ISA transactions 102 (2020) 397–406.

[7] X. Shi, A. Emrouznejad, M. Jin, F. Yang, A new parallel fuzzy data envelopment analysis model for parallel systems with two components based on stackelberg game theory, Fuzzy Optimization and Decision Making 19 (2020) 311–332.

[8] Brandizzi N., Bianco V., Castro G., Russo S., Wajda A., Automatic RGB Inference Based on Facial Emotion Recognition (2021) CEUR Workshop Proceedings, 3092, pp. 66 - 74.

[9] Brociek R., Magistris G.D., Cardia F., Coppa F., Russo S., Contagion Prevention of COVID-19 by means of Touch Detection for Retail Stores (2021) CEUR Workshop Proceedings, 3092, pp. 89 - 94

[10] D. Połap, M. Wlodarczyk-Sielicka, N. Wawrzyniak, Automatic ship classification for a riverside monitoring system using a cascade of artificial intelligence techniques including penalties and rewards, ISA transactions (2021).

[11] Capizzi G., Napoli C., Paternò L., An innovative hybrid neuro-wavelet method for reconstruction of missing data in astronomical photometric surveys (2012) Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7267 LNAI (PART 1), pp. 21 - 29, DOI: 10.1007/978-3-642-29347-4_3.

[12] A. Ozdemir, K. F. Balbal, Fuzzy logic based performance analysis of educational mobile game for engineering students, Computer Applications in Engineering Education 28 (2020) 1536–1548.

[13] C. Troussas, A. Krouska, C. Sgouropoulou, Collaboration and fuzzy-modeled personalization for mobile game-based learning in higher education, Computers & Education 144 (2020) 103698.

[14] G. Capizzi, G. Lo Sciuto, C. Napoli, E., Tramontana, A multithread nested neural network architecture to model surface plasmon polaritons propagation (2016) Micromachines, 7 (7), art. no. 110

[15] De Magistris G., Russo S., Roma P., Starczewski J.T., Napoli C., An Explainable Fake News Detector Based on Named Entity Recognition and Stance Classification Applied to COVID-19 (2022) Information, 13 (3), art. no. 137, DOI: 10.3390/info13030137.

[16] G. Lo Sciuto, G. Capizzi, S. Coco, R. Shikler, Geometric shape optimization of organic solar cells for efficiency enhancement by neural networks (2017) Lecture Notes in Mechanical Engineering.

[17] G. Capizzi, F. Bonanno, C. Napoli, Hybrid neural networks architectures for SOC and voltage prediction of new generation batteries storage (2011) 3rd International Conference on Clean Electrical Power: Renewable Energy Resources Impact, ICCEP 2011, art. no. 6036301, pp. 341 - 344

[18] D. Polap, M. Wozniak, Meta-heuristic as manager in federated learning approaches for image processing purposes, Applied Soft Computing (2021) 107872.

[19] Połap D., Wózniak M., Napoli C., Tramontana E., Is Swarm Intelligence Able to Create Mazes? (2015) International Journal of Electronics and Telecommunications, 61 (4), pp. 305 - 310, DOI: 10.1515/eletel-2015-0039.

[20] Cardarilli, G.C., Nunzio, L.D., Fazzolari, R., Panella, M., Re, M., Rosato, A., Spano, S., A Parallel Hardware Implementation for 2-D Hierarchical Clustering Based on Fuzzy Logic (2021) IEEE Transactions on Circuits and Systems II: Express Briefs, 68 (4), art. no. 9234481, pp. 1428-1432.

[21] K. Bhattacharya, S. K. De, A robust two layer green supply chain modelling under performance based fuzzy game theoretic approach, Computers & Industrial Engineering 152 (2021) 10700