Utilizing Vague Rules for Process Control: A Fuzzy Rule System Approach

Mohammad Mirzanejad

Mar. 2012

(Translated from a Persian-written text)

Introduction

The aim of the first part of the fuzzy control project is to develop a fuzzy rule system for process controls, where both the conditions and decisions can be formulated vaguely. The usefulness of vague rules in complex process control lies in the fact that a sharp value of a variable can be activated for different rules to different extents, avoiding undesirably large consequences from small differences in parameters.

Part I: Development of a generic fuzzy control system

For the first part of the project, the aim is to develop a fuzzy rule system. Fuzzy rule systems are used for process controls by using fuzzy rules. In such rules, both the conditions and the decision can be formulated vaguely. A general fuzzy rule can be written in its normal form as:

$$IF\ (\ \prod_{i=1}^{n} \coprod_{j=1}^{m} (x_{ij}\ \text{IS}\ L_{ij})\ )\ THEN\ Y\ IS\ L$$

x_{ij} and Y are ordinary variables, and $L_{ij}$and L are linguistic terms represented by fuzzy sets. A simple example of a fuzzy rule with two conditions for controlling a heater is:

$$IF(TEMPERATURE\ LOW)\ AND(PRESSURE\ LOW)\ THEN\ THE\ GAS\ VALVE\ IS\ OPEN$$

The usefulness in complex process control of vague rules is derived from the fact that a sharp value of a variable can be activated for different rules to different extents. With vague rules, it is avoided that a small difference in the parameters to be read has undesirably large consequences. Evaluating a collection of vague rules is a typical five-step process.

1. Evaluation of the conditions

When a variable appears in a condition, the value of that variable is applied to the membership function of the linguistic term from that condition and the membership degree is measured. For the condition 'temperature is low' this is shown in Figure 1. In this project it may be assumed that membership functions that appear in the rules specifications always have a point-by-point form.

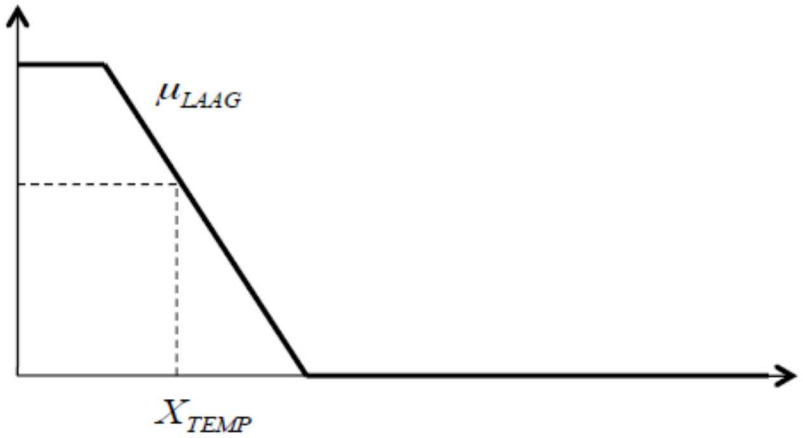

Figure 1: evaluation of the condition 'Temperature is Low'

2. Aggregation of the conditions

If a line contains multiple conditions, then the membership degrees obtained after evaluating the conditions must be aggregated. In fuzzy logic, a t-norm is used for conjunction and a t-conorm for disjunction. Within this project, you may limit yourself to (min, max), the probabilistic pair and the Lukasiewics pair. You may assume that within the same rule, the same pair of norms and co-norms are always used.

3. Determination of the control output

Post-aggregation of conditions is known for each rule an aggregated membership degree that indicates the extent to which the rule applies.

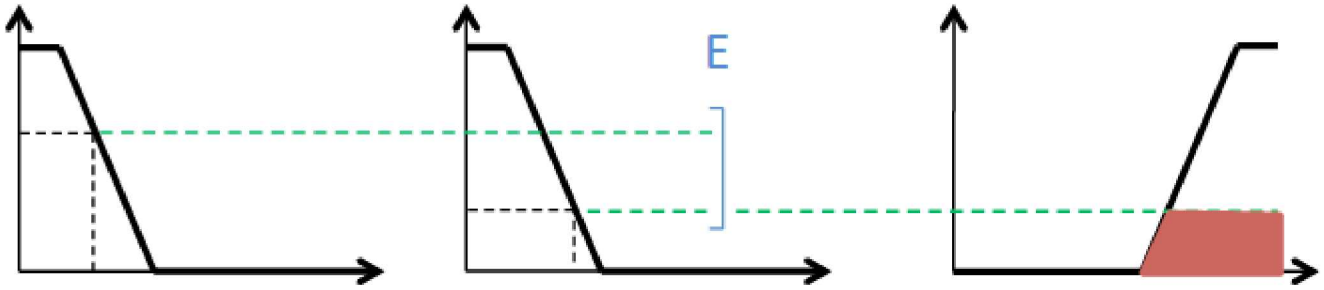

Figure 2: Determination of the rule output

4. Merging the rules

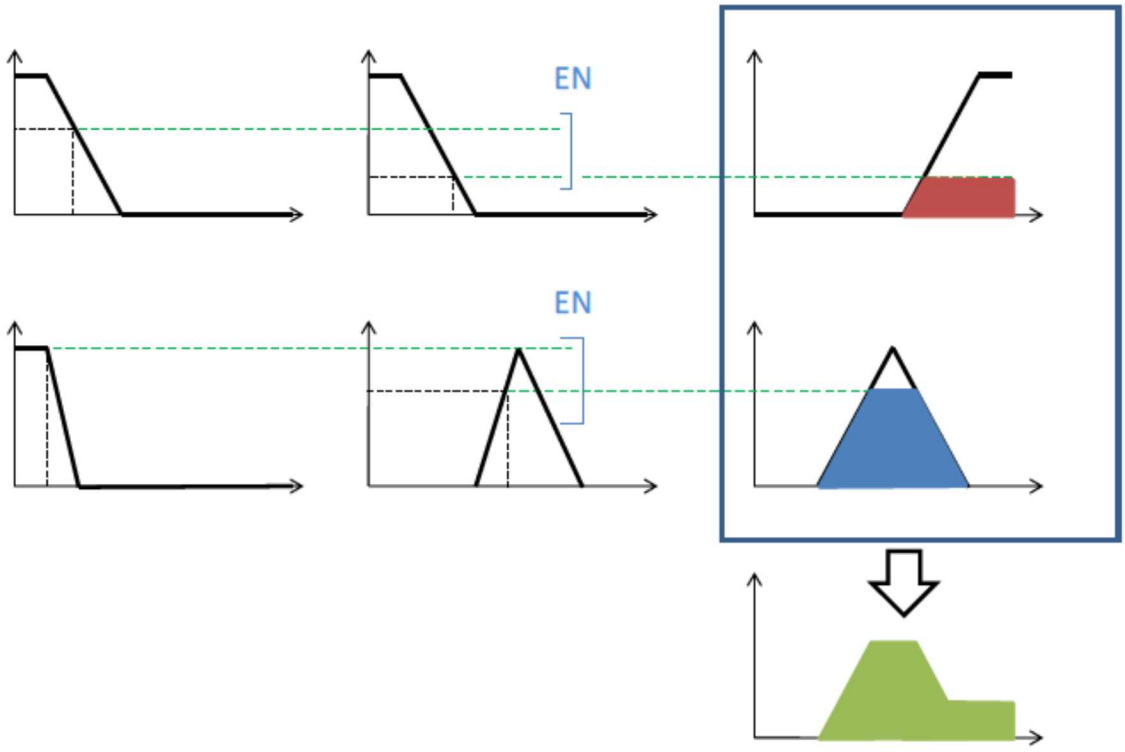

Figure 3: combination of rules

When several rules are present that control the same variable, these rules must be brought together. This operation is done by combining the obtained fuzzy sets with a union operator. As an implementation, the pointwise maximum of membership functions is used. This is shown in Figure 3.

5. Sharpening

In the last step, for each variable to be controlled, a sharp value is calculated from the obtained fuzzy set. A commonly used method for sharpening in the context of fuzzy control systems is to calculate the x-coordinate of the centroid of the surface under the obtained membership function. The x-coordinate of the centroid of the surface under a function f(x) is calculated as:

$$x_Z = \frac{\int_{-\infty}^{\infty} x.f(x).dx}{\int_{-\infty}^{\infty} f(x).dx}$$

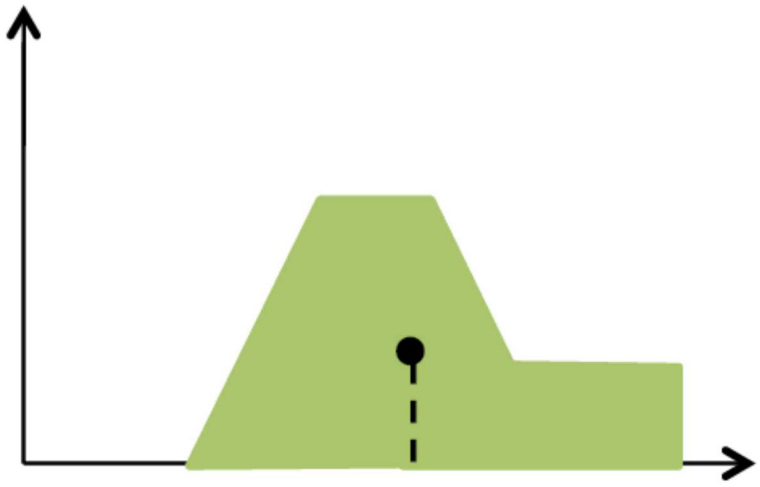

Figure 4: Tightening of the rule output

Part II: Design of a fuzzy controller

In the second part of this project, a fuzzy controller is required to be designed and controlled for a race car. The car initially recognizes fixed starting positions and must complete a lap as quickly as possible on the given circuits.

Figure 5: Texas Speedway (left) - Spa Francorchamps (right)

To render this project, we use the JMonkey Game Engine, an environment provided by them. This allows us to provide a more realistic environment where the car slips, has wheelspin, can crash, can be adjusted by counter-steering, etc.

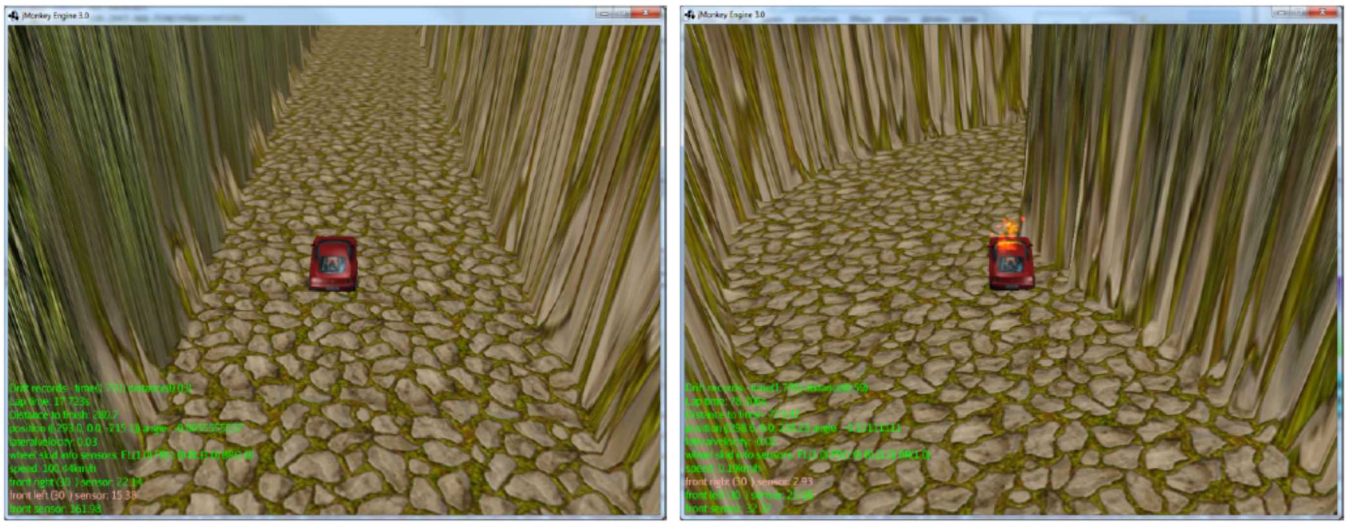

Figure 6: Race in Jmonkey Environment (left) – crash (right)

This project is provided with a few courses (lnterlagos, Silverstone, SpaFrancorchamps, TexasSpeedway) and a jar that contains them as well as the jmonkey engine and all necessary libraries. This requires Opengl 2.0.

There is one (or more) class(es) that implements theController interface. The method getFrameControl(VehiclePropertiesvp) is the only method to be implemented and must return a Framecontrol object newFrameControl(steering, acceleration, brake) in which the user specifies, based on the current vehicle properties, how much they want the vehicle to accelerate (acceleration 0...16 00), brakes (brake 0...40) and what turning angle they should try to take.

The vehicle properties available are:

- private float carSpeedKph;(0- 500)
- Sensors center, left, and right in pre-adjustable angle in frame control

  private float frontSensorDistance;

  private float leftSensorDistance;

  private float rightSensorDistance;
- private float roadWidth;
- friction of the wheels (if less than 0.1 you will start to slip):

  privatefloatfrontleftFriction;

  privatefloatbackRightFriction;

  privatefloatbackleftFriction;

  privatefloatfrontRightFriction;
- private float lateralVelocity;(allows you to measure when car actually starts to slip when it deviates from 0))

  private float frontwheelangle;(returns the angle of 1 of the front wheels)

This method aims to implement some vague control systems in order to obtain multiple vague controllers.

- A controller that focuses on reaching the finish line as quickly as possible
- A controller who gives priority to safety and must therefore ensure that the car doesn't crash.
- A controller that approaches corners rally-style, sends in with oversteer so that the rear of the car goes too far and you have to counter-steer to make the corner. (Which is also possible without a handbrake)

The intention is to generally develop the above controllers so that they work on all courses, but it is also allowed to write additional different setups (one per course) for the car (other controllers) such that the car is competitive.

Part III: Generic fuzzy control system

The project used the following additional libraries: the Apache Commons Mathematics library and the JSON.simple library. The Apache Commons Math library provides a stable numerical implementation of several integration methods. Because integration performance is more important than precision, the simplest implementation was chosen - the MidPointlntegrator - with a relatively small precision. All parameters that determine the shape of the membership functions are stored in the resources folder, and serialized in JSON. These files are read via the JSON.simple library and entered the system based on the given rules. A problem that arose early in development concerns the unconditional limiting of an aspect in the output. Suppose there is a condition for which it must hold unconditionally that an output variable is zero.

$$IF\ distanceFront\ IS\ minimal\ THEN\ acceleration\ IS\ none$$

Where the membership function of acceleration IS none involves a sparse clustering around 0. This rule cannot be expressed independently as it is the matching condition distanceFront IS minimal does not ensure with certainty that the centroid of the remaining aggregated membership function ends up at or below 0.

- **Possible solutions**

This problem involves combining each rule that affects acceleration with a distanceFront IS minimal clause, or a set of equivalent but negative rules for the positive acceleration-determining rules so that the center of gravity falls exactly at 0. Both solutions are cumbersome and unsealable. Analogously, the maximum of a membership function also poses a problem. As its fuzzy control system is defined, it is impossible to obtain the maximum value of the membership function of an output variable. If the Consequence of the rule is not limited becomes (limit = 1) and the rule is therefore fully valid, the output variable will eventually be the focus of the membership function. This will rarely be the maximum value, but more frequently approximately in the middle between the initial and final parameters (y and y in trapezoid functions).

The system - included in the fuzzy/FuzzySystem class - is first instantiated empty, and only then are the rules specified. These rules are stored internally in an Arraylist which is taken into consideration during evaluation. These rules only need to be entered once - in the constructor of the controller (in this case the BaseController takes care of this).

In addition, the system also needs input variables and values. These are every frame entered into the getFrameControl() method. Internally, the variables and their associated values are stored in a HashMap, sorted by the name of the variable. Later then the aggregated membership function is determined from an entry in the map. Thereafter the entire system is evaluated using the evaluate() method.
Although the specified control system natively uses the float data type, the choice was made to work with double, mainly because the Apache Commons Math library has double as input and output type.

- **Controllers**

1. SafeController

SafeController focuses on reaching the finish line as safely as possible. The biggest limiting factor in this controller is the speed. The design of the system limits the speed to approximately 91 km/h. As additional constraints, a minimum speed has also been imposed, as well as a braking effect when the car drives backward (for example after a frontal collision). The control of the car is governed by a set of rules that gradually change depending on the position of the car on the course. This location is calculated by the following formula.

$$ratio = \frac{sensor_{left}}{sensor_{right}}$$

Adjustments are then made depending on the proportion. This method has the advantage over a simple position determination ($sensor_{left}$ - $sensor_{right}$) that the width of the road has less influence on the course correction.
The output of the function (the Consequence) has an S-curve as a membership function. In this way, a small deviation (ratio $\approx$ 1) will have very little influence, which increases as the ratio deviates from 1. A conscious decision was made not to let the output variable steering within the domain become too high so that less or no abrupt movements occur.
Also note that SafeController is the least advanced controller in the series. It is used as a basis for SpeedController and RalyController, and there are therefore also situations that are less well handled by SafeController.

2. Speed Controller

SpeedController will try to reach the finish as quickly as possible. An extended version of SafeController's speed rules is used for this. The granularity of these rules is greater: more rules are required to give the car a controlled speed. Braking also scales, as a larger number of rules over a component also requires greater control over the complementary aspect.
The course correction algorithm from SafeController was also adjusted. At lower speeds (below 100 km/h) the SafeController system is used which provides a distribution of approximately ±0.17 to ±0.44 - but at higher speeds a less precise or large movement can have catastrophic consequences. Therefore, at higher speeds the effective threshold is higher and the movements are more precise.
A countersteering system was initially developed to prevent slippage, but it later turned out that by adjusting the sensitivity versus speed of the steering rules, this was no longer necessary. Countersteering was later expanded in RallyController.
This is a measure for worst-case scenarios and in the real world this system could never gain the upper hand.

3. Rally Controller

RallyController in turn expands on SpeedController. In RallyController the emphasis is on drifting. This phenomenon can be controlled by approaching a bend via oversteer, which causes the car to start to slip and requires counter-steering to continue through the bend

correctly. However, this approach is more theoretical than practical, due to the irregularities in the course - and in the real world - that occasionally cause a small depression in the walls of the road just before a bend. However, this is already noticed by the sensors as a measurable difference between the left and right sensors, which ultimately causes the car to drive in the wrong direction. However, tweaking the parameters of the rules cannot provide a viable solution to this. Moreover, a working combination of brakes and oversteer to get the car horizontal is practically too complex, partly due to the fact that maximum steering and braking capacity cannot be achieved.
Due to these theoretical and practical considerations, a different approach was chosen. Instead of deliberately causing the car to skid by approaching a corner in oversteer, the corners are now taken at a higher speed. By driving at speeds around 120-130 km/h, the car will start to skid anyway, after which you only need to countersteer to get the car back under control. Drifting can be detected by the combination of lateral speed and the loss of grip or friction on the tires. Once the car is drifting, the lateral velocity will indicate the direction and speed - how much counter-steering will ultimately be required. This approach has the advantage that it is independent of the position on the road or the further course. The car will counter-steer so that it can continue its trajectory in a straight line, regardless of what the left and right sensors dictate.

- **Performance**

1. Choice oft-norm and t-conorm

The difference between the use of the various t-norms and t-conorms is surprisingly small. A small test shows that the relative difference between calculations with different t-norms is typically less than 2-3%. Ultimately, this is practically impossible to measure in terms of the performance of the controllers. This is due to the definition of the membership functions used for the premises in the rules. These are mainly determined by trapezoidal pi functions with boundaries that are relatively close together. This makes the situation $\mu = 0.0$ or $\mu = 1.0$ more common. If one of the parameters of such conjunction or disjunction is 1.0 or 0.0, then the output of each t-norm or t-conorm is the same.

2. Robustness of control system

SafeController is inherently limited by speed, and it can also be effectively observed that both the average speed and the peak speed are close to each other. This phenomenon occurs on all courses, because the distanceHigh is met quite continuously, while distancelow (and therefore brakeHigh) is less dominant. It follows from this fact that multiple journeys with this controller will have practically the same effective average and top speed. With SpeedController, the speed varies slightly more, because the boundaries that determine the distance are more sensitive. On straight stretches the controller will try to achieve the greatest possible speed (within its power), while it will quickly brake and correct obstacles in the event of obstacles. The performance is virtually the same within different runs of this controller.
The RallyController is slightly less robust. Since the sensors that detect drifting are adjusted very sensitively, the smallest change in conditions can cause catastrophic consequences. Countersteering in a corner also usually happens very close to a wall, so a small change in how much the car countersteers can cause a collision with the wall. After extensive testing

and adjustment of these parameters, this rarely happens on the given course. These situations cannot of course be completely excluded according to the nondeterminism of the game engine.

3. Less performing course

Some courses - such as Spa Francorchamps and Silverstone - have a certain structure that can give treacherous measurements. Sharp bends can lead to incorrect measurements by the forward-looking sensors that ensure correct steering. In such a sharp bend, the sensors may not notice the physical bend (sufficiently) and immediately provide a measurement for the wall opposite the bend. This increases the ratio, but not enough to correct the car in the bend. This effect is enhanced by high speeds.
In itself, this problem could be solved by continuously adapting the angle of the sensors - So that a sharp bend is still noticed - were it not for the fact that the output variables of a single iteration have no (measurable) influence on the next iteration -tie. Again, this problem can be solved suboptimally by making intelligent use of dampening, which decides whether the output of the current iteration is implemented authoritatively, or whether the previous iteration has an influence on it. A good optimization for this character is a local cache of sensor values, so that certain problem patterns and obstacles can be detected. However, this system does not fall within the scope of this project and is therefore only cited as a thought experiment.

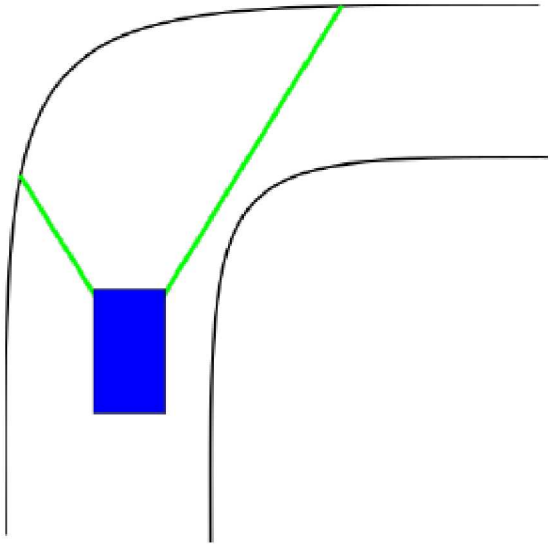

Figure 1: Incorrect measurement on sharp bends

Ultimately, it was decided to use a fixed sensor angle - scanAngle = 0.9 = 30^0- determined from empirical experiments on different courses with different controllers.

The most complex course appears to be Spa Francorchamps. Shortly after the start, this course has an apparent bend to the left, which is followed by a sharp bend to the right.
However, because the front sensor detects the end of the physical bend to the right, both the SpeedController and the RallyController will command too high a gear.
Once you arrive at the unexpected bend, the high speed and sudden adjustment in ratio will cause both controllers to make a course correction that is too abrupt, which in most cases will result in an irrecoverable frontal collision. Given the fact that the controllers were designed in general terms, it was not possible to provide additional rules for this specific (and unique) situation that would not cause undesirable effects on the other courses. However, this could

be solved in a controller specifically designed for this course where no steering adjustment occurs if the ratio exceeds a certain limit.
Amazingly, the least geographically complex course - the Texas Speedway-was also one of the more difficult courses. This circuit, like Spa Francorchamps, has a certain situation that makes it difficult for generally designed controllers. This circuit is very wide and long, which makes high speed dominant with the SpeedController. However, these extensive stretches are followed by very abrupt bends, causing insufficient braking and causing the car to spin if it turns too sharply. This can be beneficial in some situations, after which the car continues to drive, but this is certainly not the norm. However, if the faster controllers were adapted to this course, this would result in suboptimal or even disappointing performance on more complex courses such as Silverstone or lnterlagos.

4. Safety

In this model, the concept of safety only consists of contact with the wall, which increases the collisionSum. The goal is of course to reach the finish with such a low possible collisionSum. SafeController is designed to reach the finish line as safely as possible. By controlled use of speed, this is possible on most courses. Only on the Spa Francorchamps course can SafeController not reach the last sharp bend of approximately 32^0 just before the finish.
The SpeedController will reach higher speeds but will not take unnecessary risks in bends by applying the brakes in time. This should not be allowed as this controller is only equipped with minimal course correction in the event of slipping. However, this rarely happens - unless on the Texas course as indicated earlier - so this controller can be seen as sufficiently safe.
As previously indicated when discussing the robustness of the RallyController, this controller is less safe. This controller can drive into the wall, but this is usually a side collision, which allows the car to travel further. However, in exceptional cases, a frontal collision will occur, which only confirms the potential unsafety of this controller.
3.5 Exact controller
An exact controller responds to the input according to well-defined sharp limits. The big difference with these systems is that there is no gradual approach. With a vague control system, the granularity of the response can be adjusted according to constants or other (input) parameters. There may be several rules that relate to one aspect, each of which influences the outcome of the system to a certain extent.
Consider the following system with acceleration as output: If there are no obstacles in front of the car, it can accelerate. But the acceleration should be moderate when the car is drifting (i.e. when the lateral velocity is different from 0). This can be easily modeled by the following rules.

$$IF\ (\ distanceFront\ IS\ SMALL)\ THEN\ acceleration\ IS\ high$$

$$IF\ (lateral\ Velocity\ IS\ NOT\ ZERO)\ THEN\ acceleration\ IS\ low$$

By combining these rules, both acceleration IS high and acceleration IS low are taken into account in a drift, so that a balanced equilibrium is ultimately chosen as acceleration. If one wants to implement this in an exact controller, the two rules must be multiplexed into one formula. This can be done, for example, as follows.

```
int factor = 1;
if ( lateral Velocity > 0 )
        factor = 0.5;
```

```
return ( 1400 * ( distanceFront / 200 ) * factor ) ;
```

This formula contains significantly more complexity and is also more sensitive than a vague rule system. Even if (incorrect) values occur that fall outside the definition area, the output can assume an undesirable value. Fuzzy control systems have an intrinsic protection against this, since $\mu = 0$ outside the definition region of the membership function.

Final remarks

The specified system is inherently difficult to debug. The car always starts in the same place, which makes it difficult to identify and correct bugs in a later part of the course.
A form of temporal control is also missing, making bug tracking a passive and time-intensive activity. The lack of sensors makes detecting a catastrophic error virtually impossible. If the car crashes at a high enough speed, the physics engine can cause it to end up upside down. The controllers are unaware of this development and naively continue driving in the wrong direction.
There is also an appearance of non-determinism, which means that, according to the butterfly effect, every movement of, for example, the screen (going out-of-focus) or camera adjustment, leads to a completely different end result. This makes developing a general controller - which has the same acceptable performance on all courses, in all races - difficult or even impossible. Ultimately, the controllers can be optimized for each course, because each course has its own characteristics. The Texas Speedway, for example, benefits from driving very fast and only making minor course corrections. In principle, braking is not necessary. Because this project focuses on developing a general controller, this is not elaborated in detail.
Although a fuzzy control system is excellent for designing practical applications such as soft controllers,
the specified system contains too little input data to produce an acceptable result. Certain situations cannot be modeled and therefore cannot be detected and prevented. For certain situations - such as the use of
an adaptive scanAngle - an exact control system can offer a good solution. The optimal solution is therefore the golden mean: a hybrid combination of a vague and an exact control system that ensures that the controller has varied behavior and that intelligently takes the environment and the input and output variables into account.