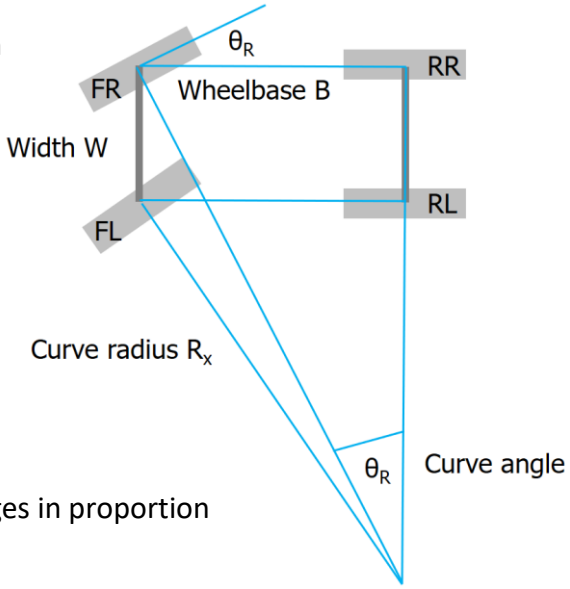
Tire Pressure Monitoring

DHBW Automotive Software Engineering – Stuttgart, © Kai Pinnow 2019

1. Introduction and Overview

The project work focuses on tire pressure monitoring. Detailed requirements below.

As depicted, the following equations govern the motion for a left turn:



$$(1)\quad R_{RR} = W + R_{RL}$$

$$(2)\quad R_{FR}^2 = B^2 + R_{RR}^2$$

$$(3)\quad R_{FL}^2 = B^2 + R_{RL}^2$$

$$(4)\quad B = R_{RR} \tan \theta_R$$

$$(5)\quad B = R_{RL} \tan \theta_L$$

Please note that any pair of velocities changes in proportion to the related curve radiuses

$$(6)\quad V_x / V_y = R_x / R_y \quad \text{for any } x,y \text{ from } \{RR, RL, FR, FL\}$$

A drop in tire pressure of a single wheel results in an increase of the respective velocity of that wheel since it will be a little smaller then.

2. Requirements and Deliverables

Please refer to “ID“ column in your documentation. A * denotes extra tasks / requirements.

ID	Check
R1	A tire pressure monitor allows observation of the four wheel speeds to detect an unexpected imbalance for a vehicle with sizes B=1.53 m and W=2.65 m.
R2	An imbalance of a wheel speed of 0.5 % of one of the wheels concerning the expected consistent wheel speeds will be regarded as an indication of a tire pressure drop.
R3	Detecting a tire pressure drop, some warning lamp shall be switched on and some “SOS” (three times short, three times long, three times short) sound shall appear (base rate 0.8 seconds).
R4*	The system shall allow “re-calibration” after inflation.
R5	Design the solution mainly with appropriate graphical modeling elements (i.e. block diagrams and/or state machines) or with scripts or ESDL and document all your decisions, reasoning, and results clearly with screenshots and text.

D1	Plan all necessary tasks based on three point estimates and monitor progress according to below requirements.
D2	Use the provided example data “curve.mat” to calculate, display, and analyze curve radiuses for selected situations. It contains the wheel speeds (vfl, vfr, vrl, vrr) in [km/h] and the steering wheel signal sw (without direction) in [degree] with time base tv in [s], plus the corresponding lateral acceleration q in [g] (with different time base tq again in [s]).
D3	Create a Simulink model that calculates the driving distance for each wheel and analyze the provided “curve.mat” data in this regard. Remember to analyze and document settings. Are there imbalances according to requirement R2?
D4	Set-up a simple tire pressure monitor in Simulink that detects a deviation according to requirement R1 and R2 by observing driving distances of the individual wheels for straight driving i.e. driving without curves.
D5	Code, configure, and apply a simple “linear congruential” random number generator like `X(i) = (a * X(i-1) + c) mod m` with suitable parameters `a`, `c`, and `m` to test the tire pressure monitoring without the provided “curve.mat” data.
D6	Execute some system tests in Simulink with the number generator from D6 to check the tire pressure monitoring function feasibility.
D7	Transfer the tire pressure monitoring function to ASCET.
D8	Provide unit tests for all designed tire pressure monitoring components.
D9	Design a warning function according to requirement R3.
D10	Provide the random number generator designed in D5 in ASCET with unit tests.
D11	Create a system test with the aid of the number generator and some error model i.e. simulating some pressure drop over a certain time to demonstrate the tire pressure monitoring.
D12*	Think about the way to calibrate the system by means of requirement R4. Which parts of the implementation shall change in order to support such a feature? How long does one need to drive for calibration?
D13*	Shall the analysis incorporate curve driving or just analyze segments driving straight?
D14*	Reflect: Which other observations or comments are in place concerning the model, the requirements, the prescribed functions, or your solution, the testing, and the selected graphical approach.

1 Aufgabe D1

Im Folgenden wird die Tabelle der 3-Punkt-Abschätzung dargestellt. Die Aufgabe D2 „Analyse der einzelnen Signale“ beanspruchte mehr Zeit als geplant. Die Aufgaben mit ASCET konnten häufig nur in der „pessimistisch“ geschätzen Zeit durchgeführt werden. Hier wurde häufig zu wenig Zeit eingeplant, um Probleme mit dem Programm ASCET zu lösen. Die Zusatzaufgabe D13* liegt weit unter der geschätzten Zeit, da hier nur ein Ansatz ausgearbeitet aber nicht umgesetzt wurde.

	A	B	C	D	E	F	G	H
1	**3-Punkt-Abschätzung**							
2	Task	optimistisch	wahrscheinlich	pessimistisch	mean	var	act	ohne extra
3	D1	20	30	60	33.33	44.44	40	33.33
4	D2	90	120	150	120	100	180	120
5	D3	50	70	90	70	44.44	75	70
6	D4	15	30	45	30	25	45	30
7	D5	20	40	60	40	44.44	40	40
8	D6	20	50	70	48.33	69.444	60	48.33333
9	D7	90	120	150	120	100	130	120
10	D8	40	60	80	60	44.44	80	60
11	D9	30	60	90	60	100	90	60
12	D10	20	30	40	30	11.11	30	30
13	D11	70	90	130	93.33	100.00	130	93.33333
14	D12*	60	100	120	96.67	100	120	0
15	D13*	120	150	180	150	100	45	0
16	D14*	40	60	80	60	44.44	45	0
17								
18				Summe	1012	927.78	1110	705.00
19				in h:	16.86	15.46	18.5	11.75

Abbildung 1.1: 3-Punkt-Abschätzung

2 Aufgabe D2

2.1 Berechnung der Radien

Im folgenden werden die in der Aufgabenstellung gegebenen Gleichungen verwendet, um die Kurvenradien der einzelnen Räder zu berechnen.

Mit (6) folgt: $\frac{R_{RR}}{R_{RL}} = \frac{V_{RR}}{V_{RL}}$

$$R_{RL} = R_{RR} * \frac{V_{RR}}{V_{RL}}$$

Mit (1) folgt: $R_{RL} = \frac{V_{RR}}{V_{RL}} * (W + R_{RL})$

$$R_{RL} - \frac{V_{RR}}{V_{RL}} * R_{RL} = \frac{V_{RR}}{V_{RL}} * W$$

$$R_{RL} * (1 - \frac{V_{RR}}{V_{RL}}) = \frac{V_{RR}}{V_{RL}} * W$$

$$R_{RL} = \frac{\frac{V_{RR}}{V_{RL}} * W}{1 - \frac{V_{RR}}{V_{RL}}}$$

$$R_{RL} = \frac{W}{\frac{V_{RR}}{V_{RL}} - 1} \quad (7)$$

Mit (1) folgt: $R_{RR} = W + R_{RL} = W + \frac{W}{\frac{V_{RR}}{V_{RL}} - 1}$ (8)

Mit (2) folgt: ${R_{FR}}^2 = B^2 + (W + \frac{W}{\frac{V_{RR}}{V_{RL}} - 1})^2$

$$R_{FR} = \sqrt{B^2 + (W + \frac{W}{\frac{V_{RR}}{V_{RL}} - 1})^2} \quad (9)$$

Mit (3) folgt: ${R_{FR}}^2 = B^2 + (\frac{W}{\frac{V_{RR}}{V_{RL}} - 1})^2$

$$R_{FR} = \sqrt{B^2 + (\frac{W}{\frac{V_{RR}}{V_{RL}} - 1})^2} \quad (10)$$

Die erstellten Gleichungen stellen die Grundlage für die folgenden Analysen dar.

2.2 Analyse der ausgewählten Situationen

Im Folgenden wird anhand einiger Graphiken das Verhalten bei Kurvenfahrten analysiert.

In der nachfolgenden Abbildung 2.1 werden zuerst die Geschwindigkeiten der vier Räder des Fahrzeugs dargestellt. Es gibt einen schnellen Überblick, wann das Fahrzeug stand, langsam oder schnell fuhr.
Es ist erkennbar, dass es Abweichungen zwischen den Geschwindigkeiten gibt. Die Gründe dahinter werden nachfolgend genauer erläutert.

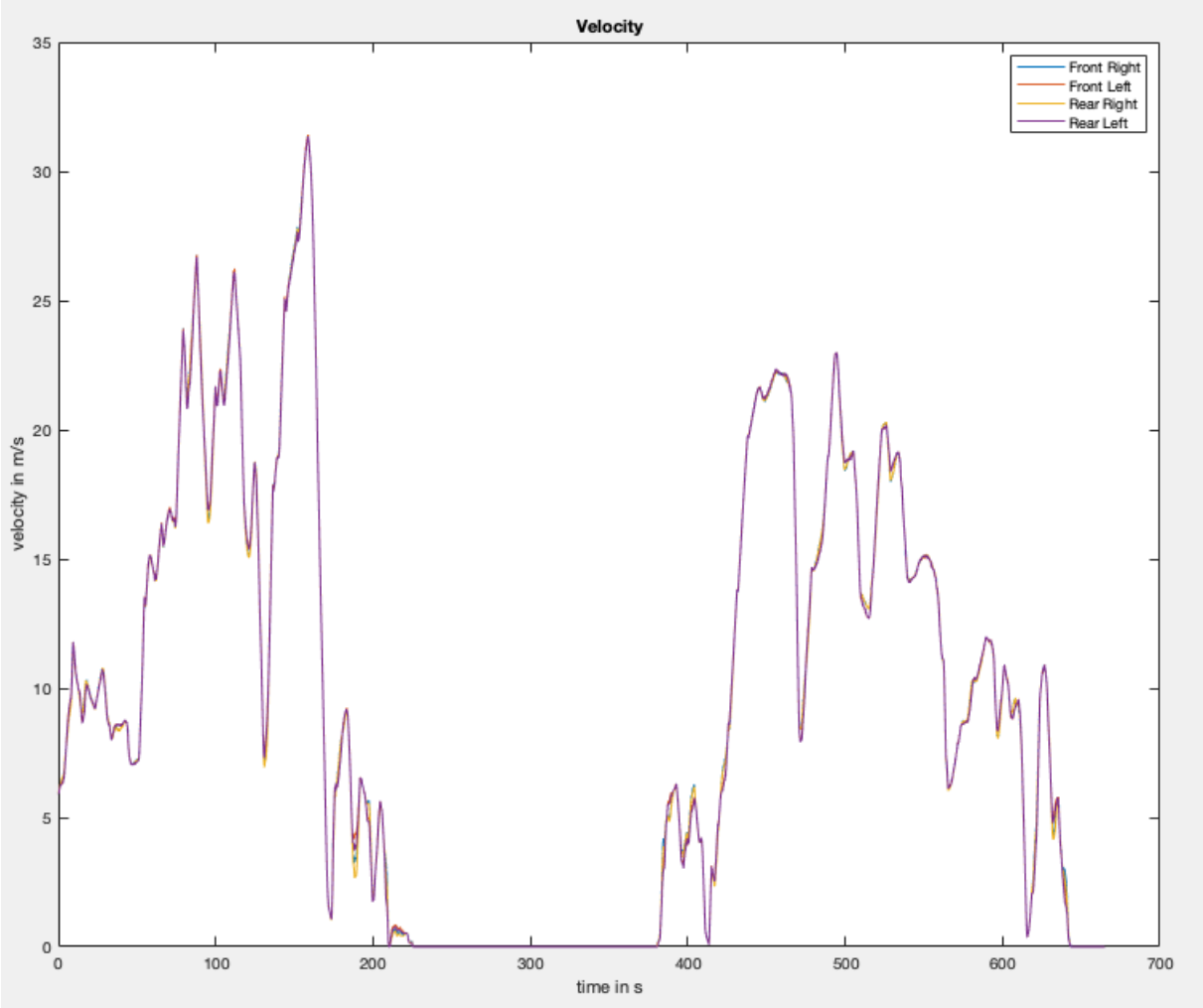


Abbildung 2.1: Übersicht der Geschwindigkeiten der vier Räder

In der nächsten Abbildung 2.2 wird der Radius in Abhängigkeit des Lenkradwinkels abgebildet. Es zeigt, dass mit steigendem Drehwinkel der Radius immer kleiner wird. Stellen, an denen das Fahrzeug nicht fährt, können hierbei vernachlässigt werden.

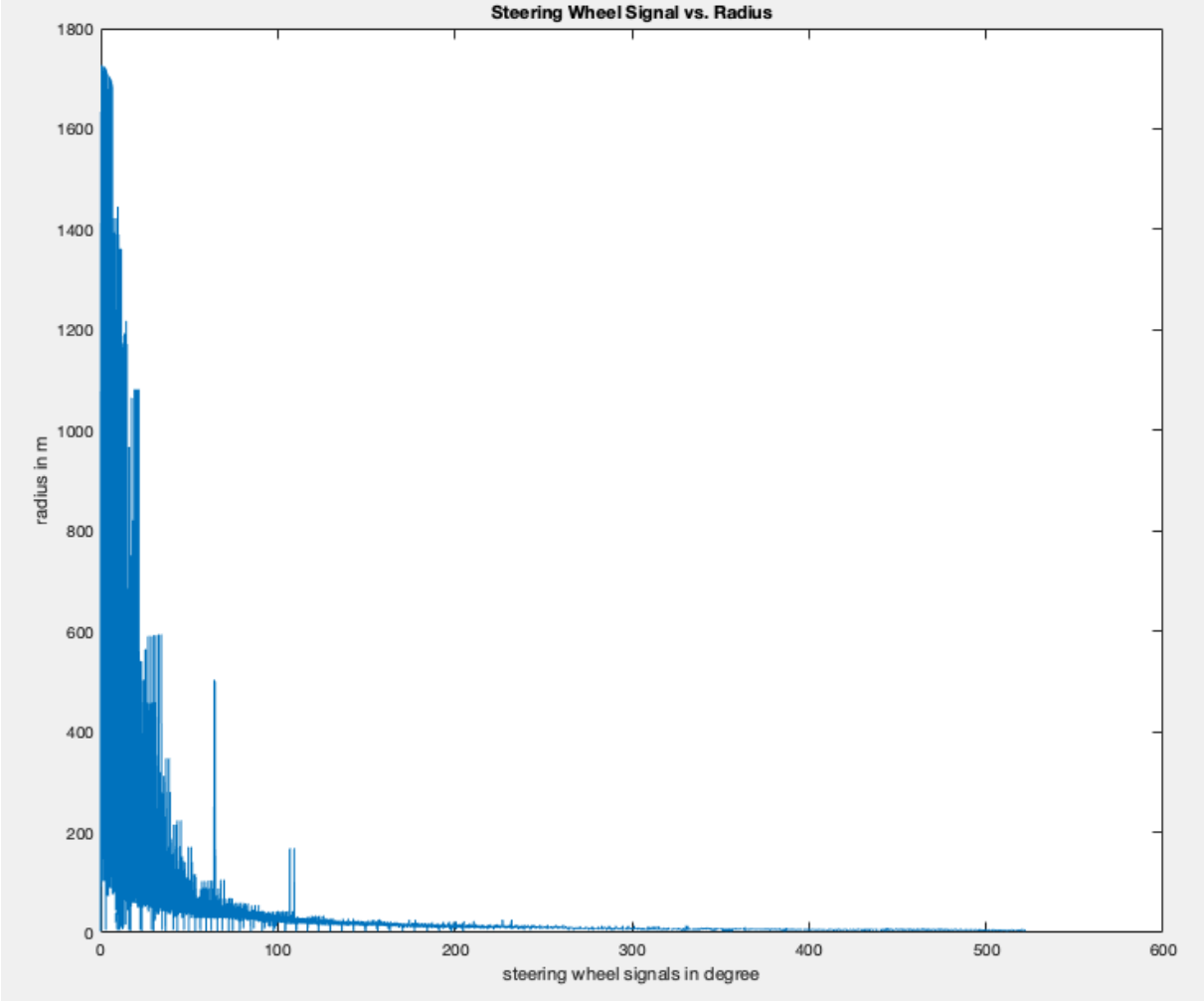


Abbildung 2.2: Lenkraddrehwinkel gegenüber Radius

In der folgenden Abbildung 2.3 wird der Radius in Abhängigkeit der Geschwindigkeit dargestellt. Es zeigt, dass bei gleichem Drehwinkel des Lenkrads bei steigender Geschwindigkeit der Radius immer größer wird.

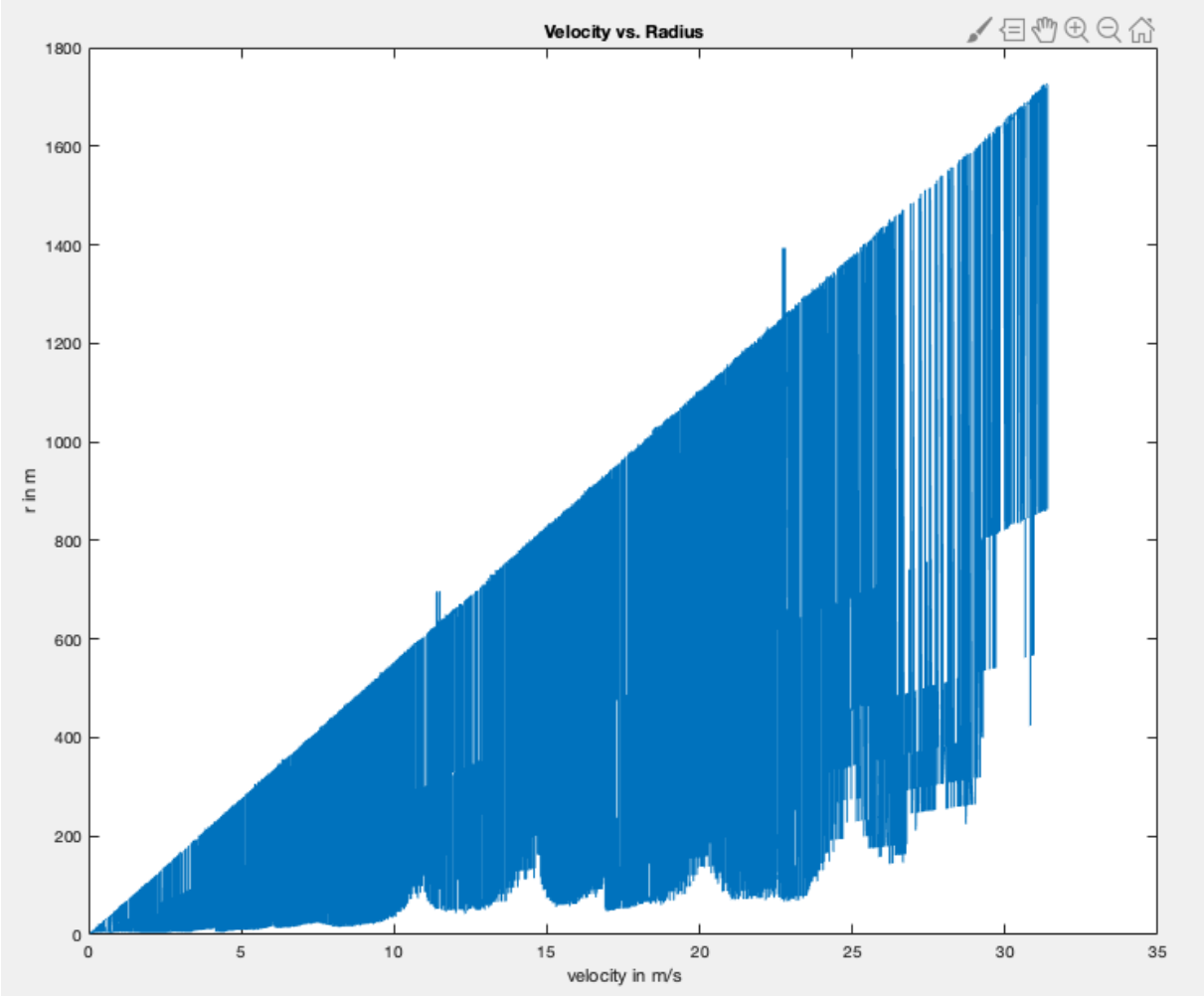


Abbildung 2.3: Geschwindigkeit gegenüber Radius

Die Abbildung 2.4 zeigt einen Ausschnitt zwischen 505s und 555s. Die Abbildung wurde in einzelne Abschnitte zur Analyse unterteilt. Im ersten Abschnitt ist der Lenkradwinkel sehr klein. Man sieht dort keinen Unterschied zwischen den Geschwindigkeiten der linken und rechten Räder. Im zweiten Abschnitt ist der Lenkwinkel wesentlich höher. Das hat zur Folge, dass die Geschwindigkeit der rechten Räder über der Geschwindigkeit der linken Räder liegt. Beim Wechsel vom zweiten in den dritten Abschnitt ist der Winkel kurz Null und steigt dann wieder. Dort liegt die Geschwindigkeit der linken Räder über der Geschwindigkeit der rechten Räder. Daraus lässt sich schließen, dass sich die Fahrt von einer Links- in eine Rechtskurve ändert. Im vorletzten Abschnitt ist wieder eine Linkskurve erkennbar. Im letzten Abschnitt wird kaum gelenkt, aus diesem Grund weichen die Geschwindigkeiten nur sehr gering voneinander ab.

Es ist zu erkennen, dass der Lenkradwinkel eine gravierende Ursache für die Differenz der Geschwindigkeiten ist.

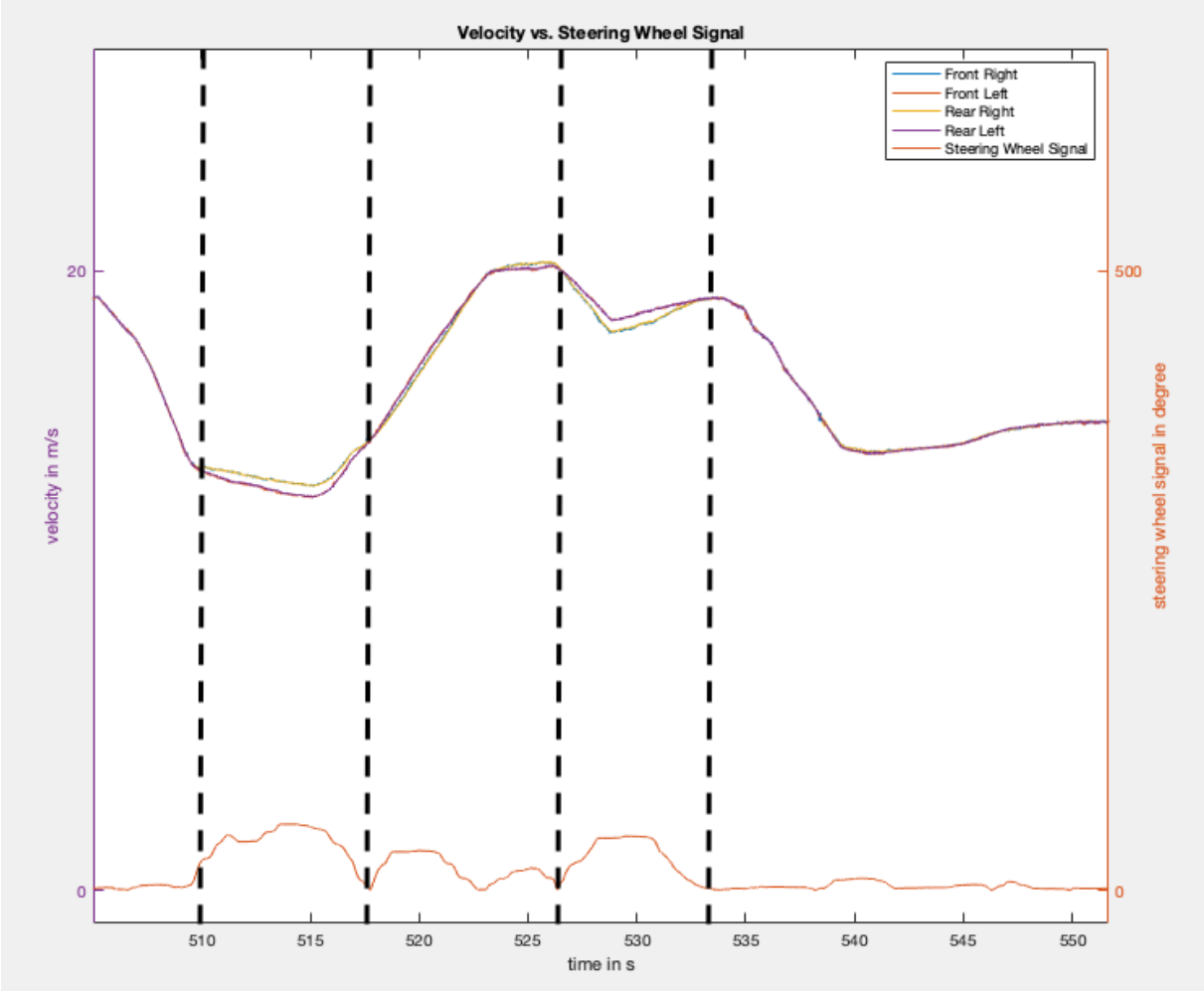


Abbildung 2.4: Übersicht der Geschwindigkeiten mit Drehwinkel des Lenkrads

Die Abbildung 2.5 zeigt den Ausschnitt von ca. 510s bis 517s. Dieser entspricht dem zweiten Abschnitt der Abbildung 2.4. Hier werden die Radien der beiden Vorderräder verglichen. Es ist erkennbar, dass der Radius des Rechten Vorderrads immer über dem Radius des Linken liegt. Dies zeigt nochmal, dass es sich hier um eine Linkskurve handelt.

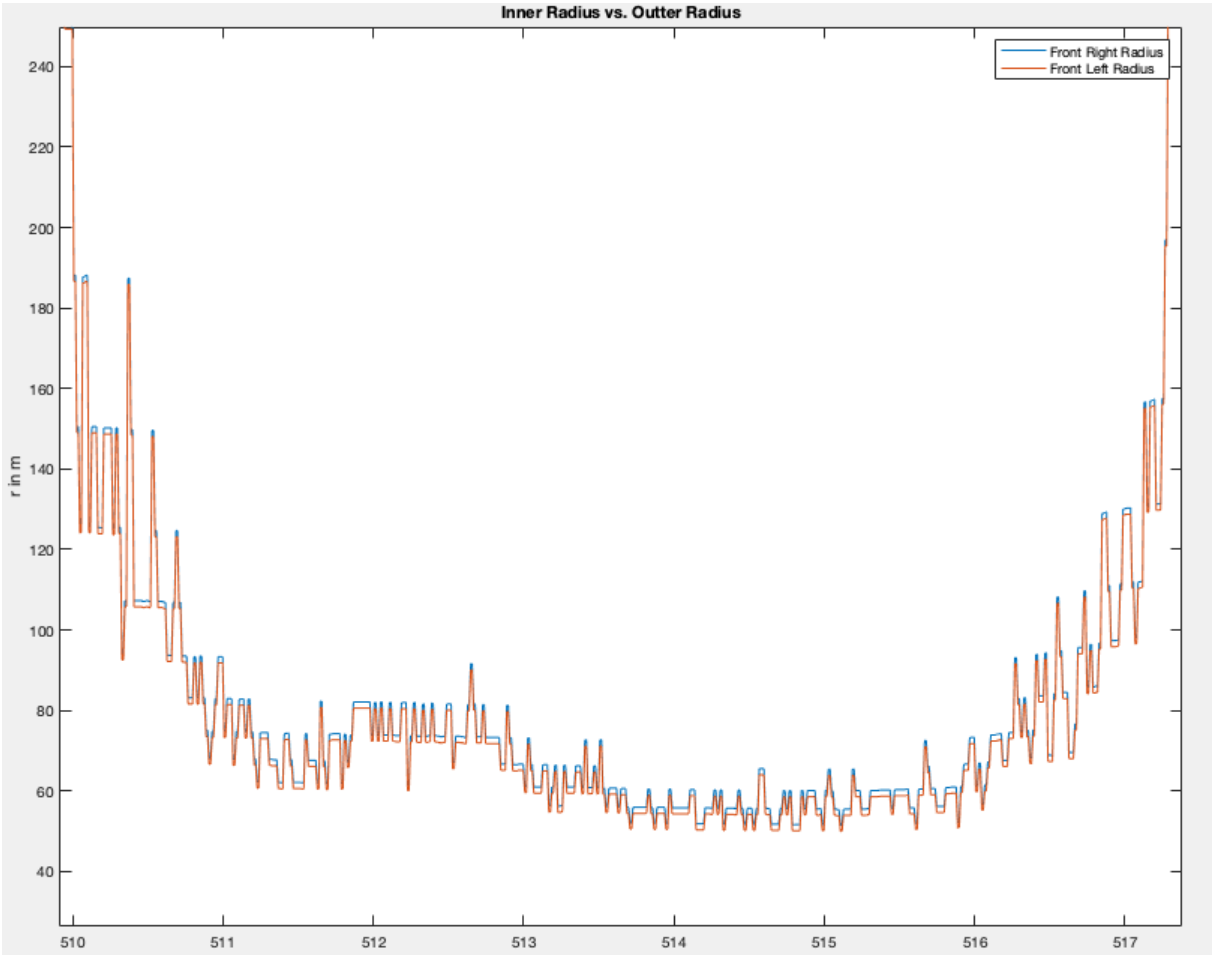


Abbildung 2.5: Vergleich des linken und rechten Radius bei Kurvenfahrt

3 Aufgabe D3

3.1 Simulink-Modell

Die nachfolgenden Abbildungen zeigen das Simulink-Modell zur Berechnung der Distanzen der einzelnen Räder.
Das Modell in Abbildung 3.1 hat die vier Geschwindigkeiten (in km/h) der Räder als Eingänge. Diese werden mit dem „From Workspace“ Block mit ihrer zugehöhrigen Referenzzeit tv in das Model importiert.
Danach werden sie in m/s umgerechnet und integriert, um die Strecke zu erhalten.
Zur Simulation wird ein FixedStep von 0.01 mit dem ode3-Solver verwendet. Der ode-Solver wurde durch die Simulink „automatic solver selection“ ausgewählt. Das Ergebnis

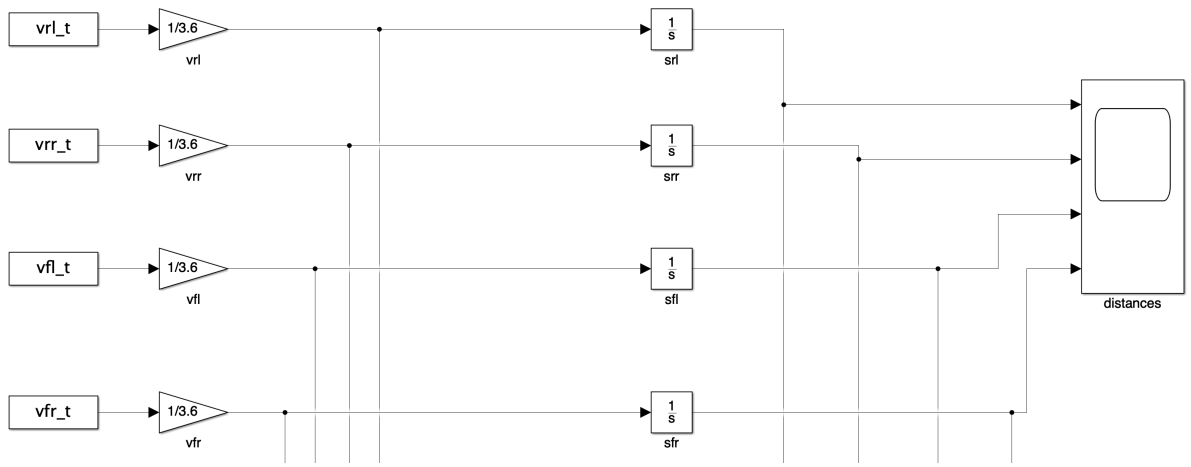


Abbildung 3.1: Tire Driving distances

der Integration sind die Distanzen der einzelnen Reifen, die gegenüber der Zeit aufgetragen sind.

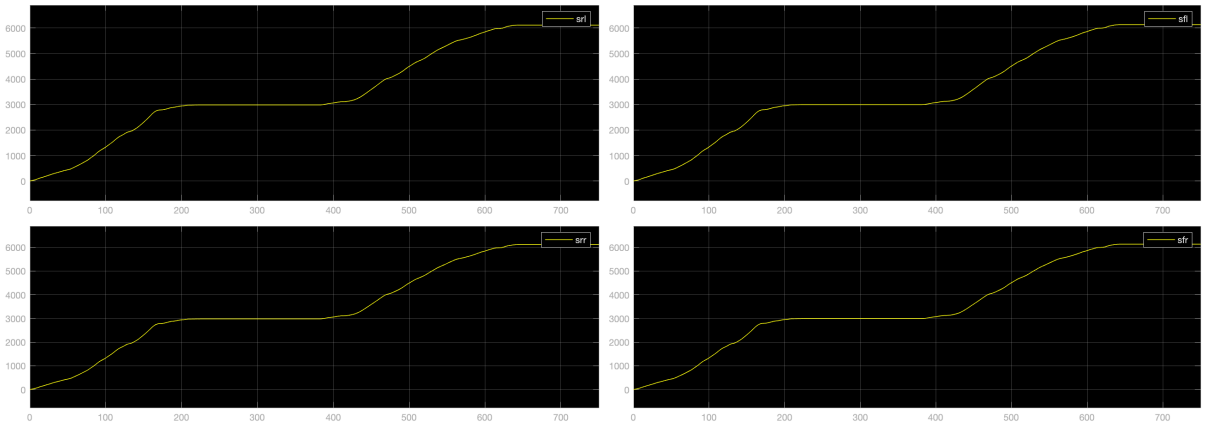


Abbildung 3.2: Tire Driving distances plot

Für die Kurvenfahrt entsteht die folgende Datenaufzeichnung über die Reifendruckabweichungen.

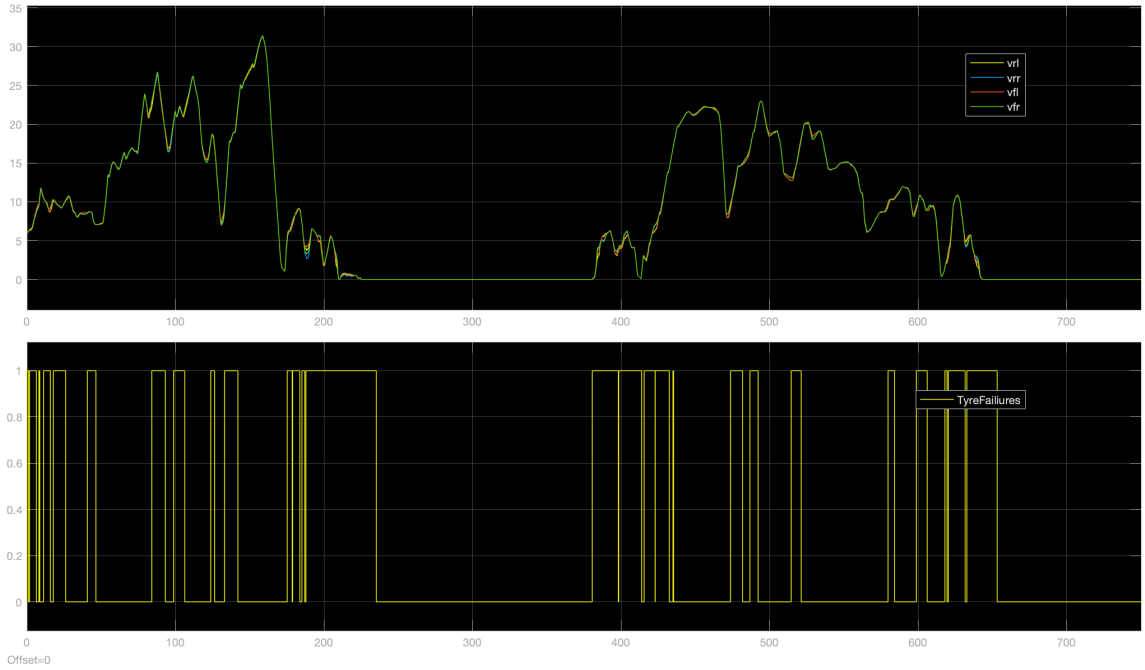


Abbildung 3.3: Tire Monitor für Kurvenfahrt

Ein Wert von 1 entspricht der Beobachtung einer Imbalance und ein Wert von 0 entspricht keiner Anomalie.
Wie berechnet wird, ob Imbalancen vorliegen wird in D4 thematisiert. Es werden viele Imbalancen gemessen, aber diese Implementation ist auch nicht für Kurvenfahrten geeignet. Weitere Ausführungen werden in D13 aufgeführt.

4 Aufgabe D4

Zur Überprüfung, ob es Imbalancen in den Daten gibt, wird das Modell, das in Abbildung 3.1 zu sehen ist, erweitert. Der Mittelwert der vier integrierten Distanzen gilt als Referenzwert um festzustellen, ob ein Reifendruckabfall vorliegt. Die Berechnung ist ebenfalls im erweiterten Modell Abbildung 4.2 zusehen. Anschließend wird die prozentuale Abweichung vom Einzelsignal zum Mittelwert berechnet, weicht die Abweichung um mehr als 0.5% ab handelt es sich um einen Reifendruckabfall. Die Berechnung ist in Abbildung 4.1 dargestellt.

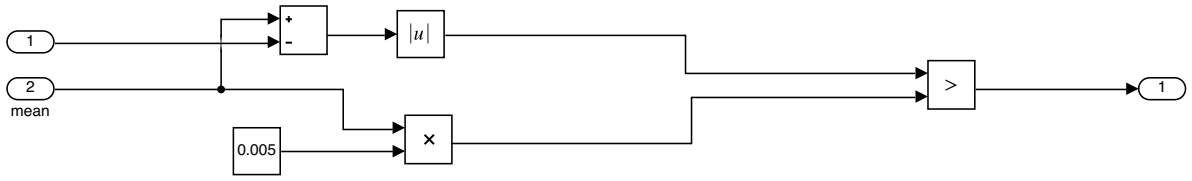


Abbildung 4.1: Prozentuale Abweichung nach R2

Damit die Messungen nicht durch Rauschen im Geschwindigkeitssignal verfälscht werden, wird das integrierte Signal zur Messung verwendet. Durch die Integration wird kurzzeitiges Rauschen sozusagen gemittelt, sodass es die Messung nicht verfälscht. Damit jedoch ein Reifendruckabfall nicht untergeht, da das bereits integrierte Signal zu lange kein Reifenfehler enthielt, wird ein „Transport Delay“ Block genutzt der das Signal um eine beliebige Zeit (hier 10 Sekunden) verzögert. Das verzögerte Signal kann dann vom Originalsignal abgezogen werden, sodass ausschließlich das Integral der letzten 10 Sekunden betrachtet wird.[1]
Die Berechnungen für die Abweichungen werden für jeden Reifen einzeln durchgeführt und anschließend, wie in Abbildung 4.2 zu sehen, „verodert“, um ein Überblick über das Gesamtsystem zu erhalten.

[1] In den ersten 10 Sekunden, ist der Output des Transport Delays 0, sodass in den 10 Sekunden Rauschen durchaus die Messungen verfälschen kann. Deswegen wird der Output des Transport Delays in den ersten 10 Sekunden manuell auf -1000 gesetzt, sodass die Messung in den ersten 10 Sekunden bewusst so manipuliert wird, dass es keine Imbalancen gibt

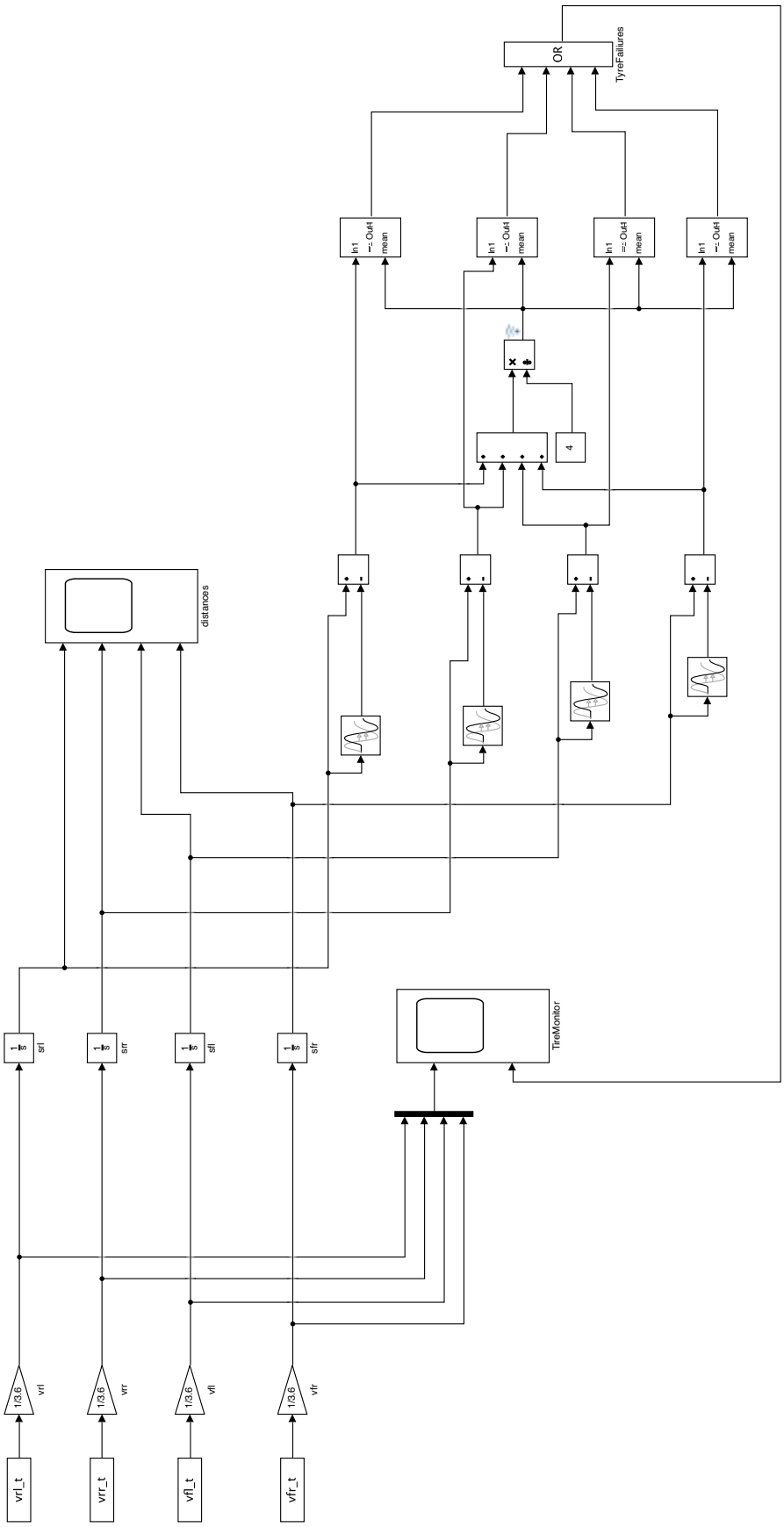


Abbildung 4.2: Simulink-Gesamtmodell

5 Aufgabe D5

Die untenstehende Abbildung 5.1 zeigt den "Random Number Generator".
Die Daten werden nach der Vorschrift

$$X(i) = (a * X(i-1) + c) \mod m$$

generiert.
Die Parameter a, c und m sowie X(0) sind hierfür frei wählbar, um optimale Zufallsazahlen zu erhalten.
Dafür sind einige Vorschriften zu beachten:
$Modul\, m \in \{2, 3, 4, \ldots\}$
$Faktoren\, a_1, \ldots, a_n \in \{0, \ldots, m-1\}\, mit\, a_n > 0$
$Inkrement\, b \in \{0, \ldots, m-1\}$
$Startwerte\, y_1, \ldots, y_n \in \{0, \ldots, m-1\} \qquad (nicht\, alle\ 0,\ wenn\, b = 0)$

Nach diesen Vorschriften wurde a = 89 und c = 253 gewählt. Beide Zahlen sind Primzahlen und haben somit keinen gemeinsamen Teiler mit m = 2^{31}. m als Zweierpotenz zu wählen ist besonders effizient, da so die Modulooperation durch das abschneiden der Zahl in Binärdarstellung berechnet werden kann.
Für $c \neq 0$ muss außerdem folgendes gelten, damit m Zufallszahlen bei einem beliebigen Seed generiert werden:
1. m und c haben keinen gemeinsamen Teiler
2. a - 1 ist durch alle Primfaktoren von m teilbar
3. a - 1 durch 4 teilbar wenn m durch 4 teilbar ist
Die zusätzlichen Anforderungen werden von den gewählten Parametern erfüllt. Zur Generierung von 4 Datensätzen werden unterschiedliche Startwerte X(0) gewählt. Die Rechenoperation wurde darüber hinaus noch ergänzt. Wie in Abbildung 5.1 dargestellt wird die nach der Generierung der Zufallszahl nach obengenannter Vorschrift, noch die Hälfte von m subtrahiert, um auch negative Zufallswerte zu erhalten. Des Weiteren wird danach die Zufallszahl noch mit einem Wert multipliziert, um die Amplitude, um die die Zufallszahlen um 0 oszillieren zu setzen. Abschließend wird noch ein Zielwert addiert, um die Zufallszahlen um einen Zielwert oszillieren zu lassen.

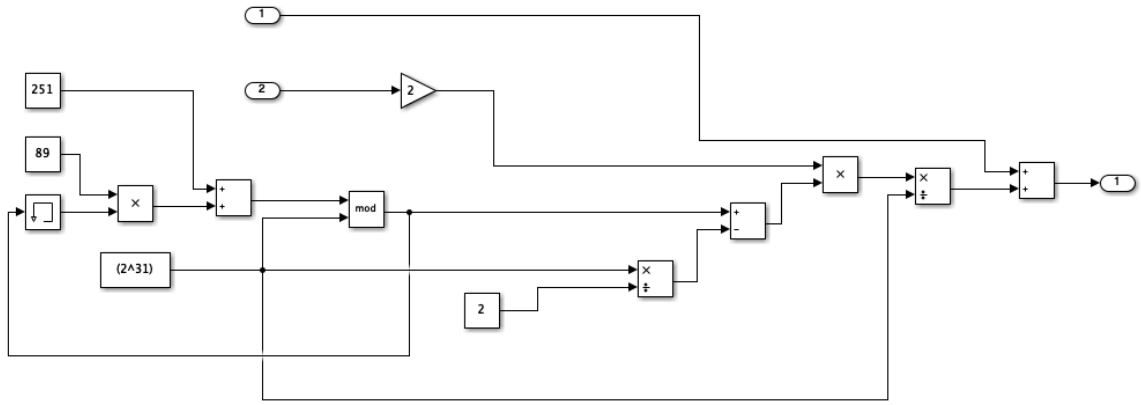


Abbildung 5.1: Random Number Generator

6 Aufgabe D6

Statt der aufgezeichneten Daten werden nun die Geschwindigkeiten aus dem Randomgenerator genutzt, um das Modell zu prüfen.
Bei dem Randomgenerator kann die Basisgeschwindigkeit eingestellt werden, die anschließend mit Rauschen, das durch den Randomgenerator generiert wurde, aufaddiert wird. Sowie das Noiselevel/Amplitude, welches das generierte Signal haben soll. Im Folgenden ist ein Ausschnitt zu sehen, wie der Randomgenerator in das bestehende Modell integriert wird, sowie die erzeugten Daten des Randomgenerators und ob es Imbalancen im System mit dem Randomgenerator gibt.

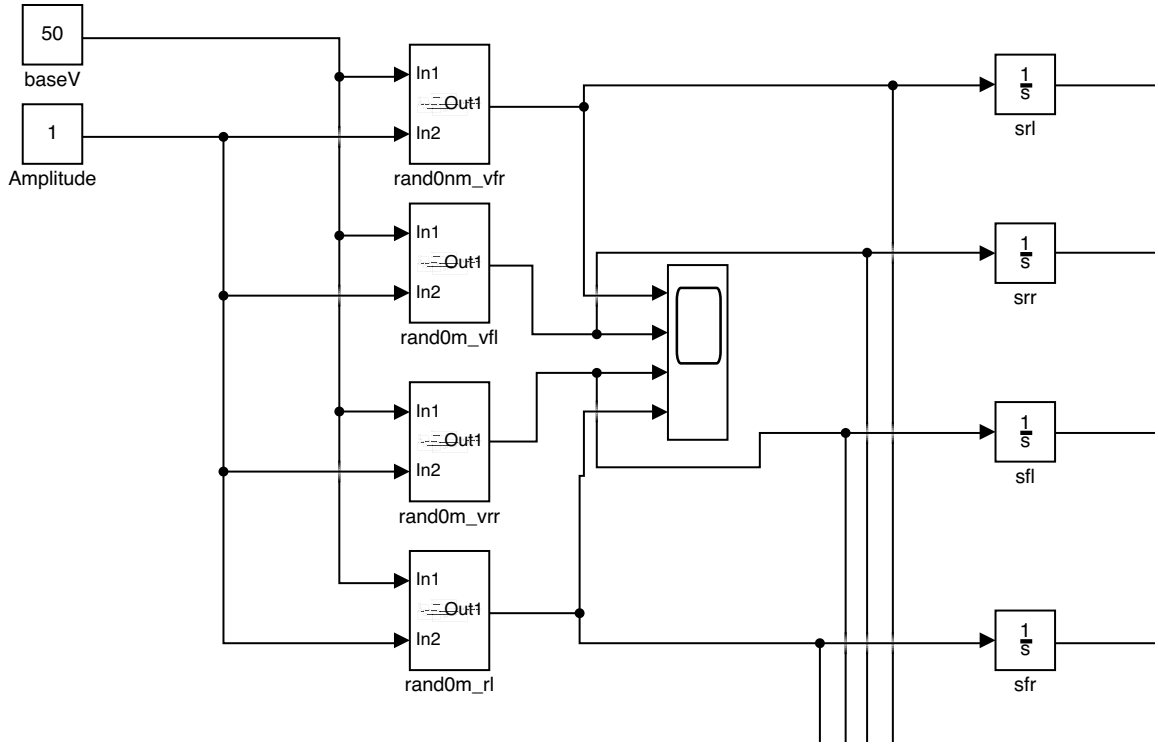


Abbildung 6.1: Include Random Generator

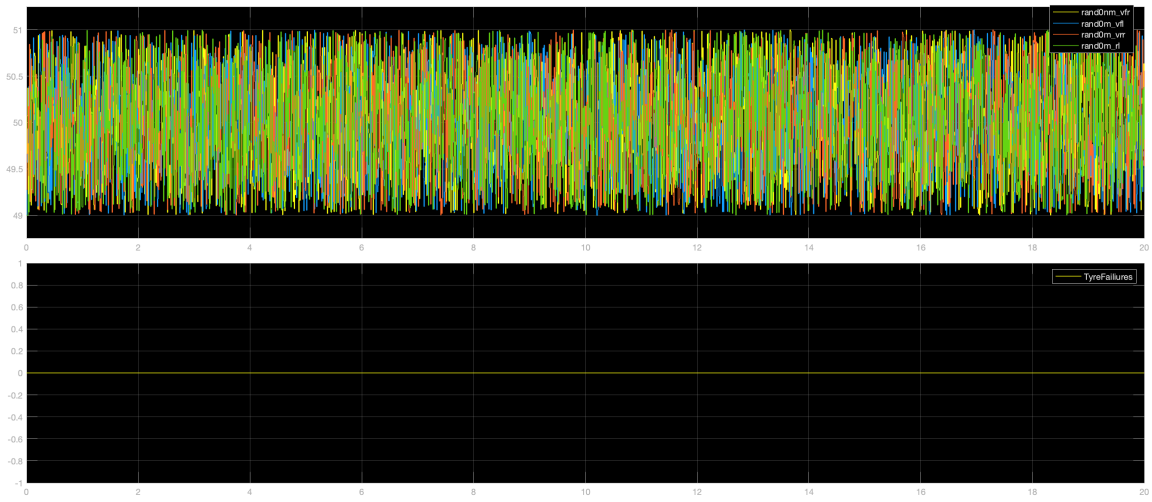


Abbildung 6.2: Generated Data

Es wird deutlich, dass es keine Imbalancen bei diesem Beispiel gibt. Auch bei anderen Basisgeschwindigkeiten und Noiseleveln, die einer realistischen Geradeausfahrt entsprechen, treten keine Imbalancen auf.
Damit ist der konstruierte Randomgenerator gut geeignet, um Geradeausfahrten zu simulieren.

Wird nun eine Basisgeschwindigkeit so manipuliert, dass die generierte Geschwindigkeit im 10 Sekundendurchschnitt über den 0.5% liegt, so signalisiert der Tire Monitor eine Reifenimbalance. Als Beispiel haben hier drei Reifen eine Basisgeschwindigkeit von 100km/h, der abweichende Reifen hat eine Basisgeschwindigkeit von 100.7 km/h. Alle Signale Rauschen um 0.5 km/h in die positive und negative Richtung.
Im Durchschnitt sollte also eine Geschwindigkeit aller vier Reifen von 100.175 km/h erreicht werden. Die prozentuale Abweichung des abweichenden Reifen ist somit:

$$\frac{|100.175 - 100.7|}{100.175} = 0.524\%$$

Der Tire Monitor bestätigt dieses und zeigt eine Imbalance an.

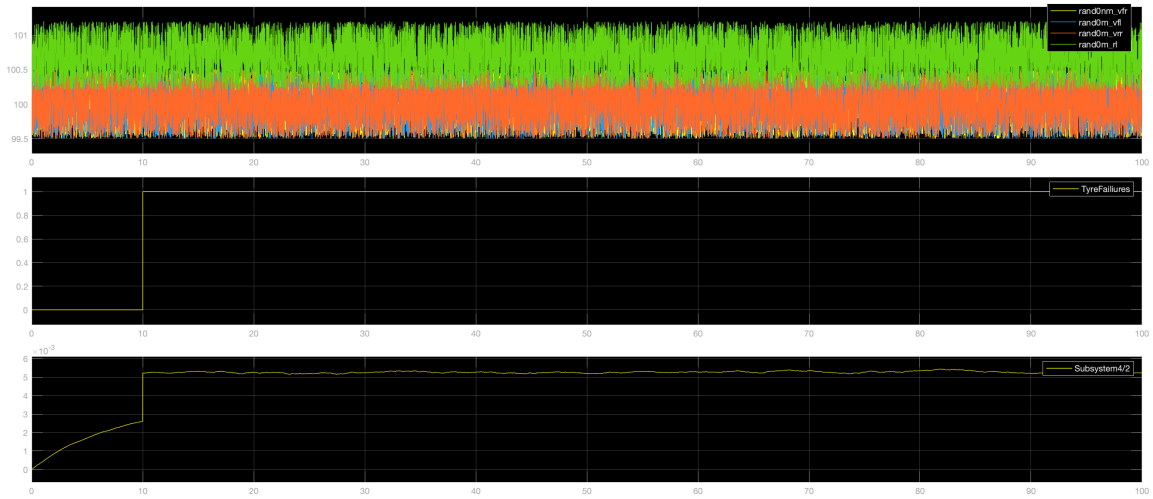


Abbildung 6.3: Systemtest Funktionalität

Der Bereich bis 10s ist bewusst manipuliert wie in D4 angesprochen wurde.

7 Aufgabe D7

Die folgenden Graphiken zeigen die Umsetzung der Simulink-Simulation in ASCET. Die

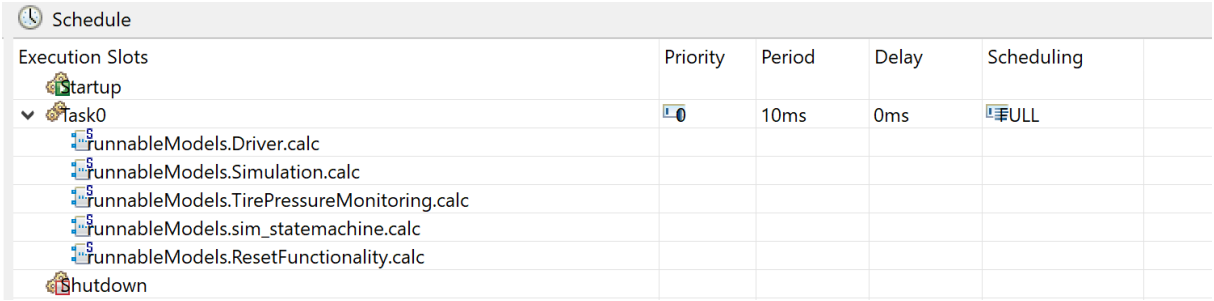


Abbildung 7.1: ASCET Schedule

Simulink-Simulation ist in ASCET in drei Teile geteilt:

- Driver
- Simulation
- TirePressureMonitoring

Die weiteren Teile werden in folgenden Aufgaben weiter erläutert.

Driver

Der Driver ist ein Containermodell und enthält den Randomgenrator als Submodul.

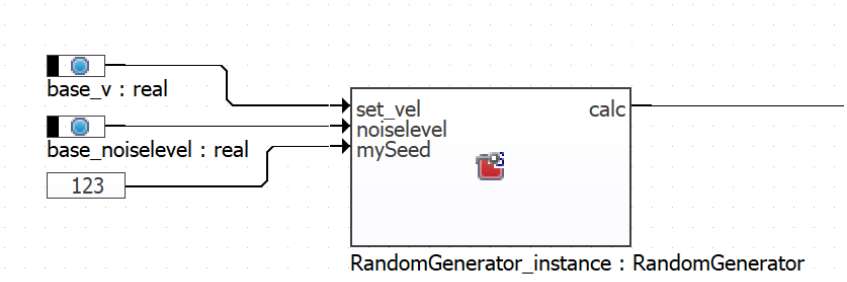


Abbildung 7.2: ASCET Randomgenerator

Dieser ist für jedes Rad einmal vorhanden. Eine Message sendet das jeweilige Signal weiter an andere Module.
Die Implementierung unterscheidet sich kaum von Simulink. Eine setSeed Funktionalität wurde ergänzt.

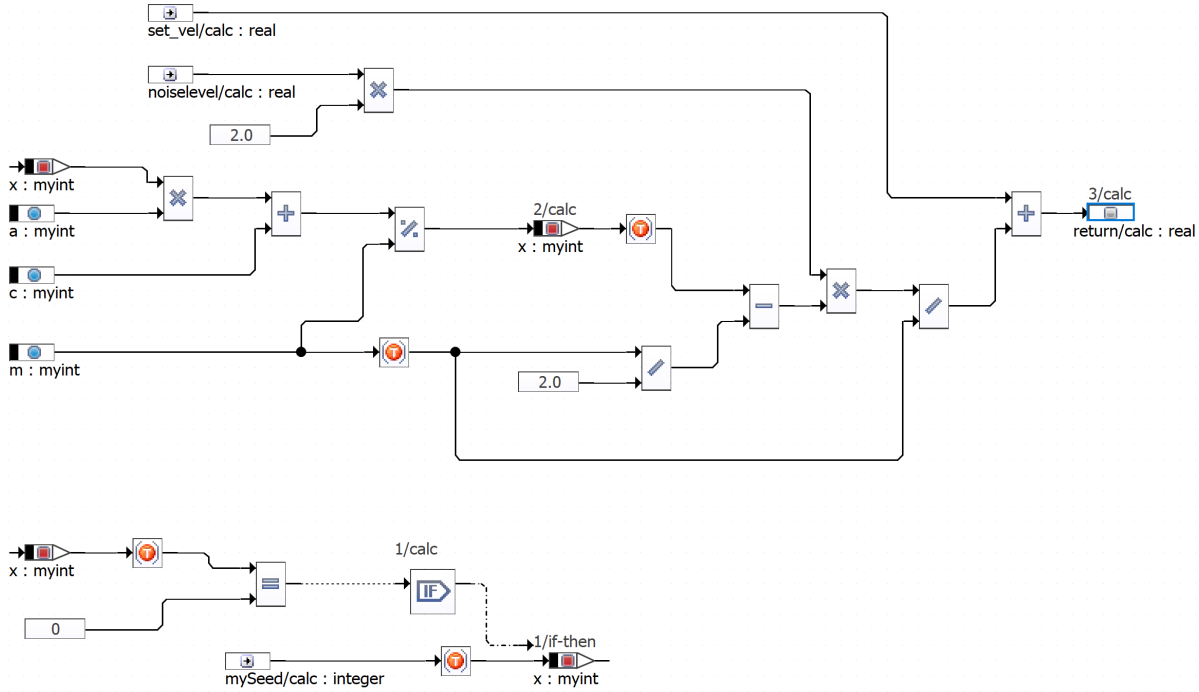


Abbildung 7.3: Random Number Generator

Es wird auch ein eigener Typ für die Modulooperation benötigt.

```
type myint is integer 0 .. 1024;
```

Die Parameter sind gleich implementiert. Bei der Codegenerierung mit m = 2^{31} ist ein Fehler aufgetreten. Die Vermutung ist, dass dieser Wert für die Integerdarstellung zu groß ist. Deshalb wurde m auf 2^{10} gesetzt.

```
class RandomGenerator {
characteristic myint a = 89;
characteristic myint m = 1024;
characteristic myint c = 251;
myint seed;
myint x = 0;
@generated("blockdiagram")
public real calc(real in set_vel, real in noiselevel, integer in mySeed)
     {
...
}
}
```

Simulation

Die Simulation berechnet die Distanzen und Deltas in einem Submodul und schickt diese dann per Message weiter.

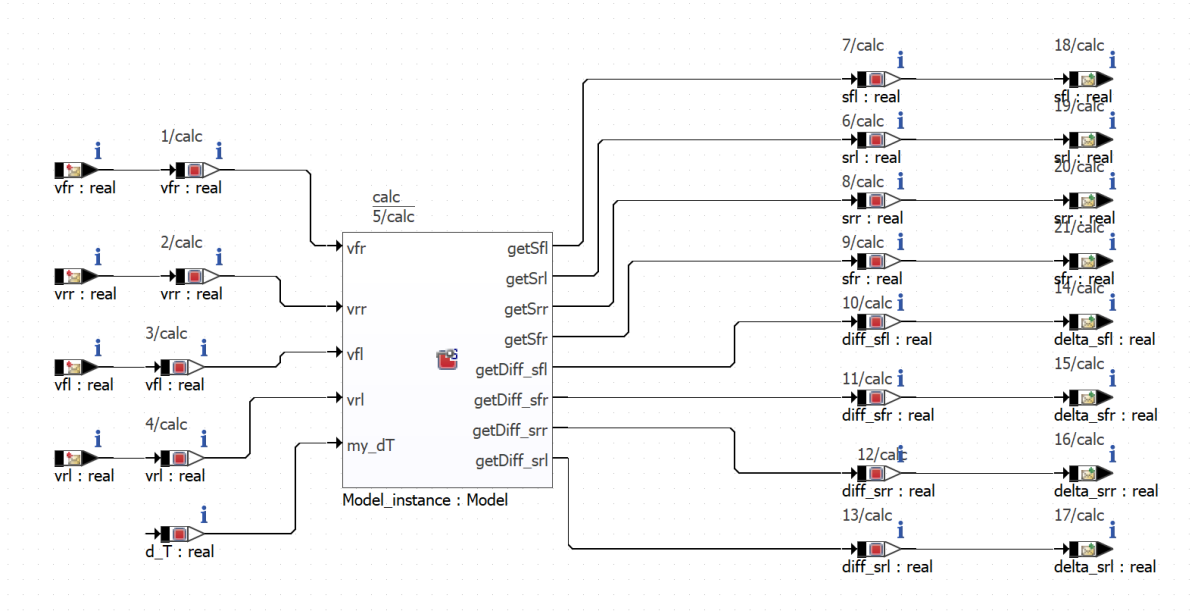


Abbildung 7.4: Simulation

Im Submodul wird die Strecke durch Integration mit dem dT-Element für alle vier Räder berechnet.

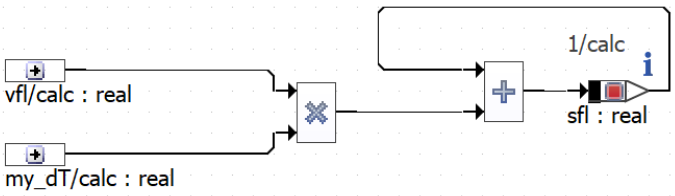


Abbildung 7.5: Streckenberechnung für Vornelinks

Um das 10 Sekundendelta zu erhalten wird der gespeicherte Wert von vor 10 Sekunden abgezogen (auch für alle vier Reifen).

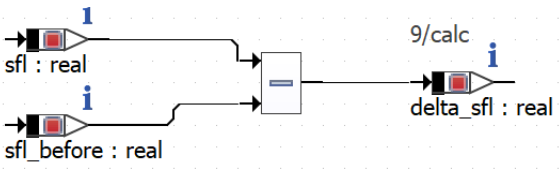


Abbildung 7.6: Deltaberechnung

Das Speichern vergangener Werte wird durch einen Buffer umgesetzt. Jeder Reifen besitzt einen eigenen Buffer.

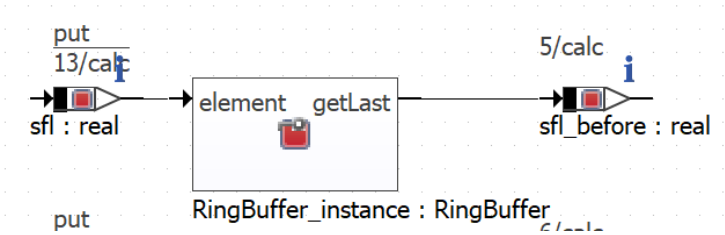


Abbildung 7.7: Buffer

Der Buffer wird als Code implementiert.

```
package components;
type s_array is array [] of real;

class RingBuffer{

  s_array buffer[1000];
  real c;
  real swap;

  public void put(real element){
    swap = element;
    for(i in 0 .. 999){
    c = buffer[i];
    buffer[i] = swap;
    swap = c;
    }
  }

  public real getLast(){
    return buffer[999];
  }

  public real getIndex(integer i){
    return buffer[i];
  }
}
```

Die Codenotation ist einfacher zu implementieren und für diesen Fall auch übersichtlicher. Um die berechneten Größen im Topmodul zugänglich zu machen, werden für die Strecken und Deltas getter-Funktionen im Code implementiert.

TirePressureMonitoring

Im Topmodul wird der Mittelwert,

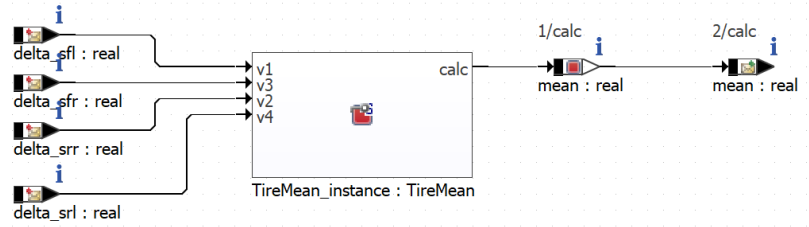


Abbildung 7.8: Berechnung des Mittelwerts

und die Prozentuale Abweichung für alle Räder berechnet.

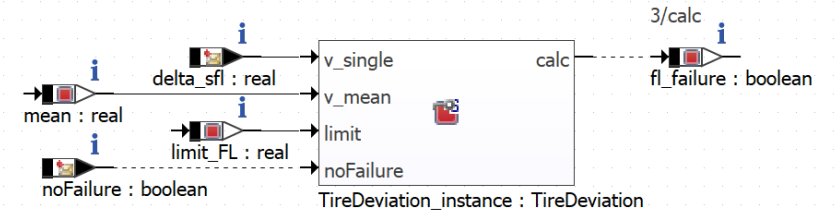


Abbildung 7.9: Druckabweichungsberechnung und Fehlererkennung

Abschließend werden die Ergebnisse aller vier Reifen verodert.

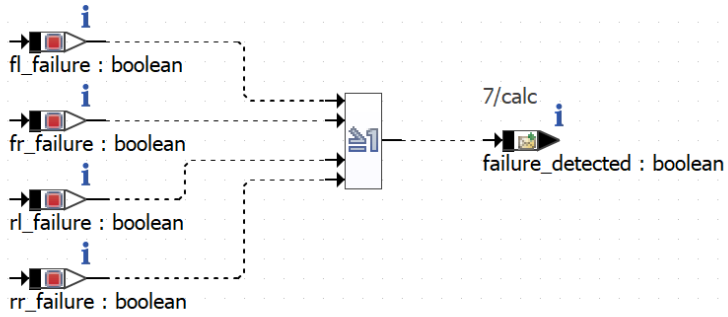


Abbildung 7.10: Fehlererkennung

Die Mittelwertberechnung unterscheidet sich gar nicht vom Simulinkmodel und ist wie in Abbildung 7.8 in einem Submodul, das in der nachfolgenden Abbildung dargestellt ist, umgesetzt.

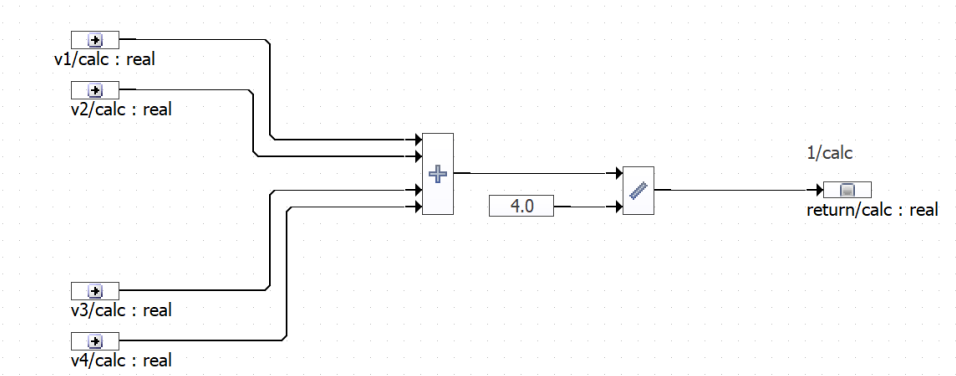


Abbildung 7.11: Mittelwertberechnung

Die Abweichungsberechnung unterscheidet sich ebenfalls nicht vom Simulink Modell und ist ebenfalls wie in Abbildung 7.12 in einem Submodul, das in der nachfolgenden Abbildung dargestellt ist, umgesetzt.

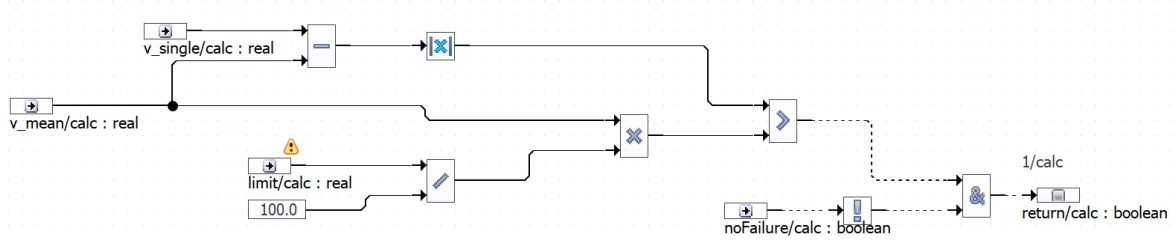


Abbildung 7.12: Abweichungsberechnung

Auf den Parameter noFailure wird in der Reset Funktionalität eingegangen.

8 Aufgabe D8

In diesem Abschnitt werden folgende Module getestet:

- Simulation
    - Model
- TirePressureMonitoring
    - Mean
    - Deviation

Simulation

Anforderungen:

- Korrekte Integration durch wachsende Strecke bei Geradeausfahrt
- Buffer wird korrekt beschrieben/gelesen
- Deltas werden korrekt berechnet

Korrekte Integration:

```
1  @Test
2  public void distanceIncreasing(){...}
3  @Test
4  public void predictDistance(){...}
```

Diese Tests testen erfolgreich, ob die Distanzen der Räder wachsen und somit die Integration korrekt durchgeführt wurde. Außerdem überprüfen sie, ob die Räder die richtige Distanz nach einer bestimmten Zeit erreicht haben.

Korrekte Implementierung des Buffers:

```
1  @Test
2  public void start_with_empty_buffer(){...}
3
4  @Test
5  public void fill_buffer_decending(){...}
6
7  @Test
8  public void read_last_element(){...}
```

Diese Tests testen erfolgreich, ob der Buffer richtig beschrieben und ausgelesen wird.

Deltas korrekt berechnet:

```
1  @Test
2  public void deltasConsistentForSameVelocity(){...}
```

Dieser Test bestimmt, ob die Deltas richtig berechnet werden. Zum Test wird einfachheitshalber ausschließlich die gleiche Geschwindigkeit verwendet. Eine variable Geschwindigkeit ist nicht notwendig, da die Deltas auf der gefahren Strecke beruhen, die sich auch bei gleichbleibender Geschwindigkeit ändert.

TirePressure Monitoring

Anforderungen:

- Korrekte Durchschnittsberechnung
- Korrekte Berechnung der Abweichung
- Korrekte Signalisierung der Abweichung

Korrekte Mittelwerte:

```
1  @Test
2    public void calcMeanAllValuesTheSame(){...}
3
4    @Test
5    public void calcMean(){...}
6
7    @Test
8    public void calcMeanNegativ(){...}
9
10   @Test
11   public void calcMeanMixed(){...}
```

Diese Test zeigen erfolgreich, dass die Mittelwertberechnung mit unterschiedlichsten Werten funktioniert.

Korrekte Signalisierung der Abweichung:

```
1   @Test
2   public void noDeviation(){...}
3
4   @Test
5   public void highDeviation(){...}
6
7   @Test
8   public void lowDeviation(){...}
```

Diese Tests zeigen erfolgreich, dass Abweichungen signalisiert werden, unabhängig davon, ob der Mittelwert über- oder unterschritten wird. Dies zeigt auch, dass die Abweichungen korrekt berechnet werden.

Keine Signalisierung der Abweichung bei Kalibrierung:

```
1 @Test
2 public void noDeviationWhileCalibration(){...}
3
4 @Test
5 public void highDeviationWhileCalibration(){...}
6
7 @Test
8 public void lowDeviationWhileCalibration(){...}
```

Im Rahmen von R4/D12 wird hier getestet, ob Signalisierungen von Abweichungen während der Kalibrierung unterdrückt werden. Weiteres dazu wird in Aufgabe D12 angesprochen.

9 Aufgabe D9

Für die Umsetzung der Warnfunktion durch eine LED, die einen Druckabfall-/anstieg signalisiert, wurde eine Statemachine verwendet. Diese besteht aus insgesamt fünf Stati, vier die jeweils 0,8 bzw. 1,6s dauern und die LED ein- bzw. ausschalten und dem Off-Status. Die Variable „status“ steuert die LED. Ein Ausgeben eines Tons wurde nicht umgesetzt, da im Experiment Environment, keine Tonausgabe möglich ist. Sollte dies in zukünftigen Versionen implementiert werden, kann die Variable „status“ genauso den Ton steuern. Weitere Variablen sind ausschließlich Hilfsvariablen oder zum Testen.

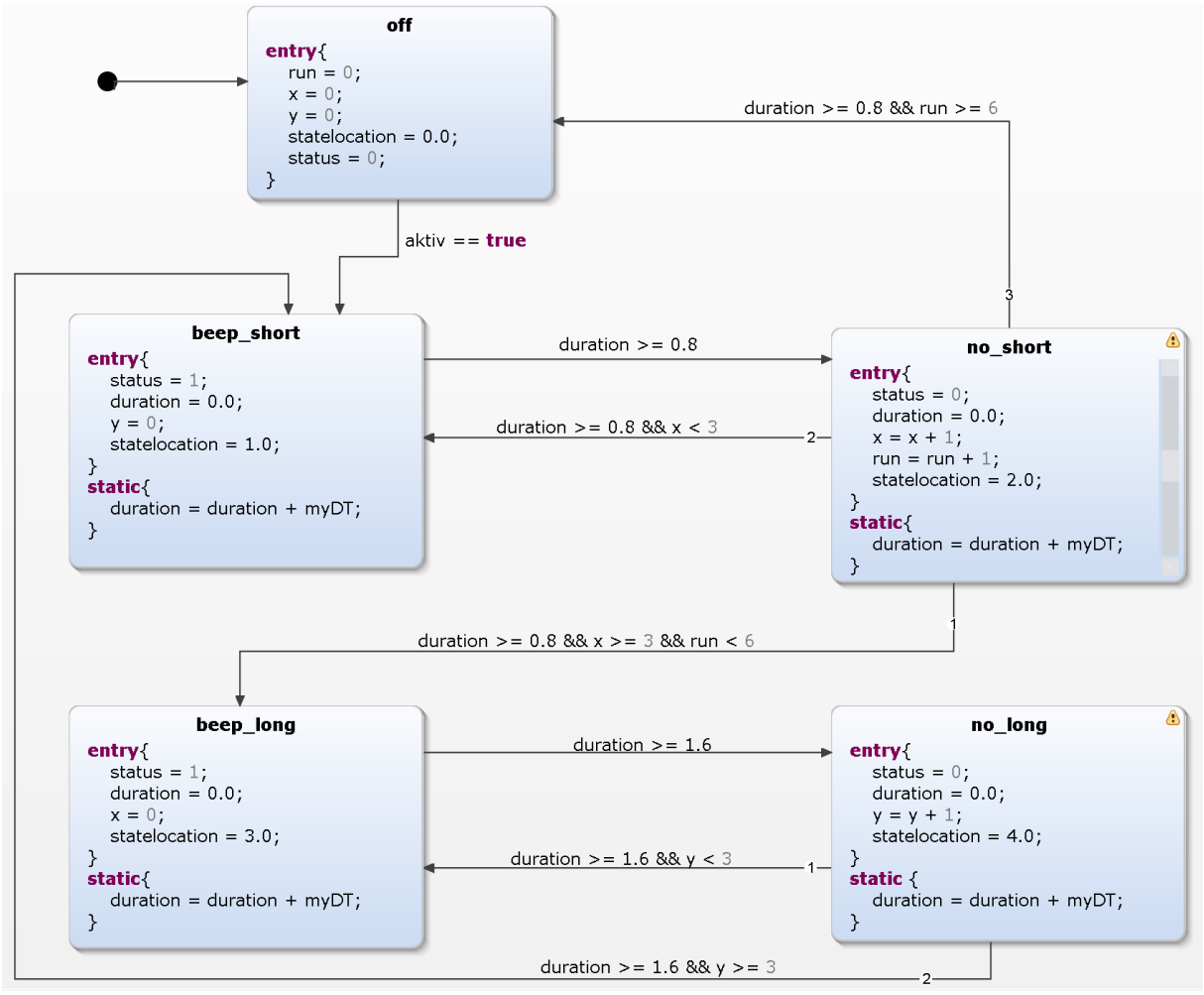


Abbildung 9.1: SOS Statemachine

Für die Statemachine „SOS“ wurden ebenfalls Unittests durchgeführt. Es wird getestet, ob die Statemachine zu jedem Zeitpunkt x im richtigen State ist. Dafür wurde eine Debugvaribale „statelocation“ eingeführt.
Des Weiteren wird erfolgreich getestet, ob die Statemachine bei einem dauerhaften Failure aktiv bleibt.

```
1   @Test
2     public void checkAllStatelocationsAndStatesActiveContinues(){}
```

Außerdem wird überprüft, ob die Statemachine in den Off-State wechselt, wenn nach einem vollständigem Durchlauf kein Fehler mehr vorliegt.

```
1     @Test
2     public void checkAllStatelocationsActiveContinuesNot(){}
```

Zuletzt wird getestet, ob die Statemachine nicht dauerhaft läuft und solange kein Fehler anliegt im Off-State verweilt.

```
1     @Test
2     public void checkAllStatesDeactiv(){}
```

10 Aufgabe D10

Hier wird beschrieben, wie der System Driver getestet wurde.
Anforderungen:

- Zufallsgenerator führt einen Rechenschritt korrekt durch
- Der Zufallsgenerator generiert verschiedene Zufallszahlen
- Der Zufallsgenerator generiert mit dem gleichen Seed die gleichen Zufallszahlen

Korrekte Berechnung:

```
1    @Test
2    public void generateFirstDataStep() { }
```

Ein Testen jedes Datenschritts wäre zu aufwendig, deswegen wird zum einen getestet, ob der erste Rechenschritt korrekt durchgeführt wird.

Verschiedene Zufallszahlen:

```
1    @Test
2    public void generateDifferentNumbers() {    }
```

Per Definition sollen m unterschiedliche Zufallszahlen generiert werden. Dies wird mit diesem Test erfolgreich getestet.

Reproduzierbarkeit

```
1    @Test
2    public void reproducible(){}
```

In diesem Test wird erfolgreich getestet, ob die gleichen Zufallszahlen mit dem gleichen Seed mehrfach produziert werden können.

11 Aufgabe D11

Nachfolgend wird beschrieben, wie dem Driver Model ein Error Model hinzugefügt wird. Dieses Model wird in Abbildung 11.1 abgebildet.

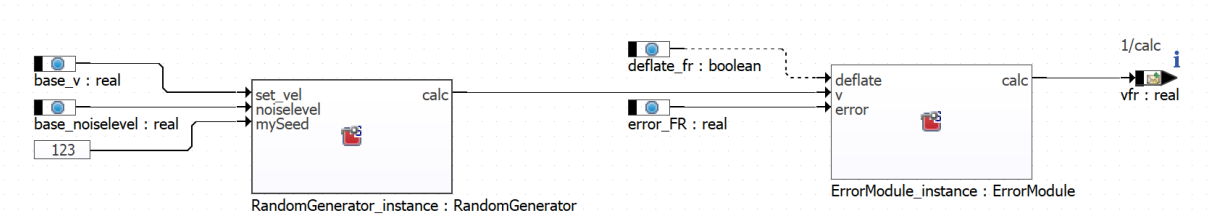


Abbildung 11.1: Error Module Integration

Ist das Error Modul inaktiv leitet es die generierte Geschwindigkeit durch. Wird es aktiviert wird der generierten Geschwindigkeit ein prozentualer Fehler abgezogen.

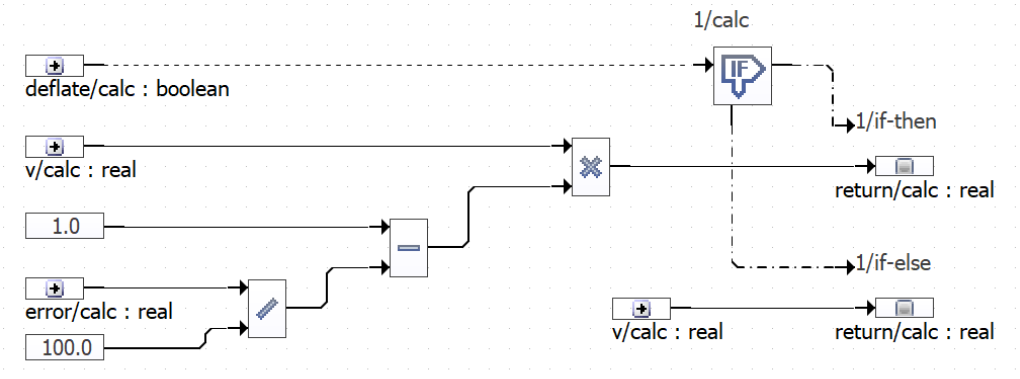


Abbildung 11.2: Error Module

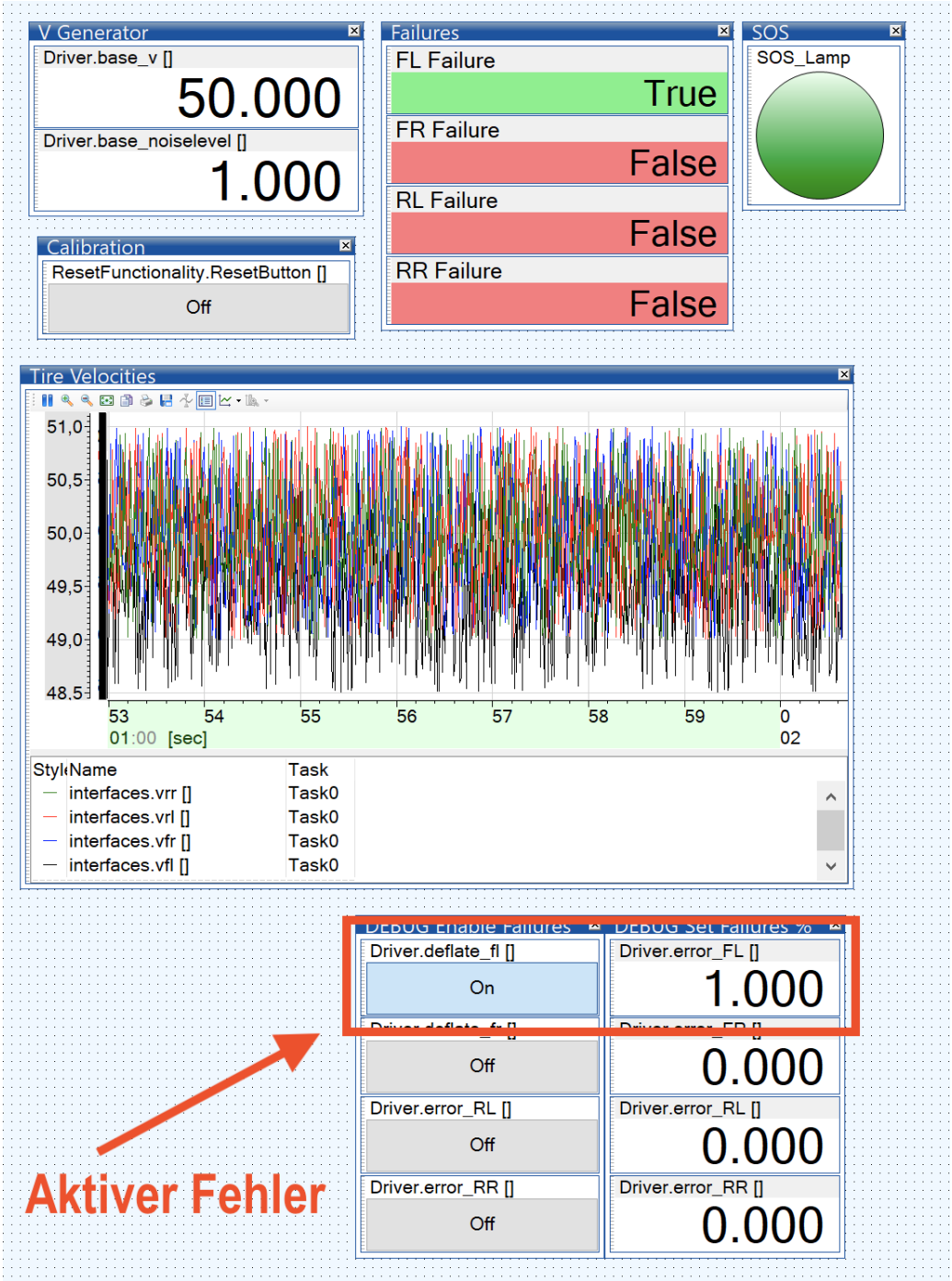


Abbildung 11.3: Reifendruckabfall aktiv

Hier wird nur ein aktiver Fehler gezeigt, der länger als 10 Sekunden aktiv ist.

12 Aufgabe D12

Füllt der Fahrer seine Reifen wieder auf ist es unwahrscheinlich, dass alle Reifen den exakt gleichen Druck haben. Deswegen wird eine Resetfunktionalität implementiert, die diese Ungenauigkeiten in der Berechnung für Reifendruckabfälle mit einbezieht.
Dieser Reset soll vom Fahrer manuell betätigt werden und kann nur initiiert werden, wenn das Fahrzeug steht. Dafür wird das folgende Modul in Abbildung 12.1 verwendet.

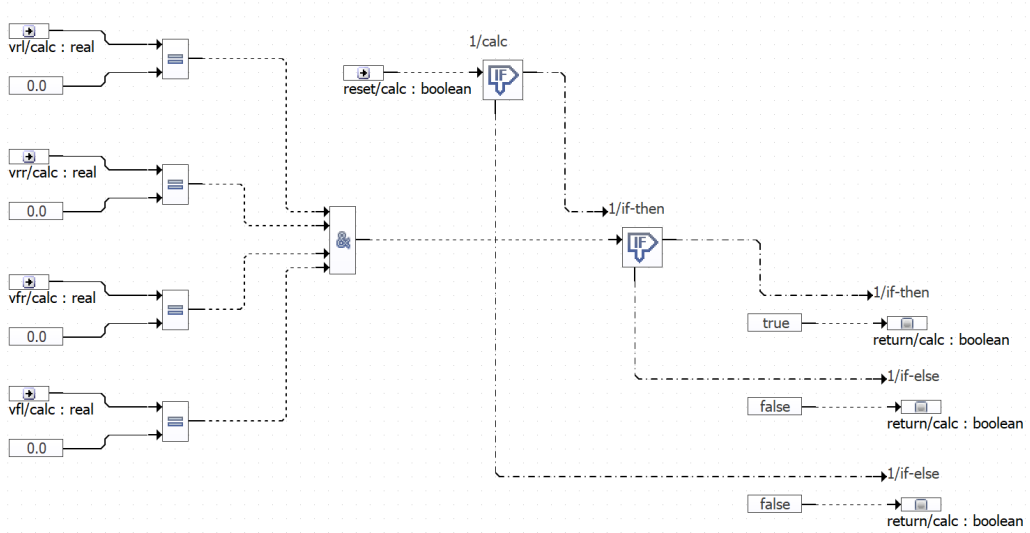


Abbildung 12.1: Reset Funktion

Eine Ebene höher interagiert die Funktion mit einer Statemachine die den Reset bearbeitet.

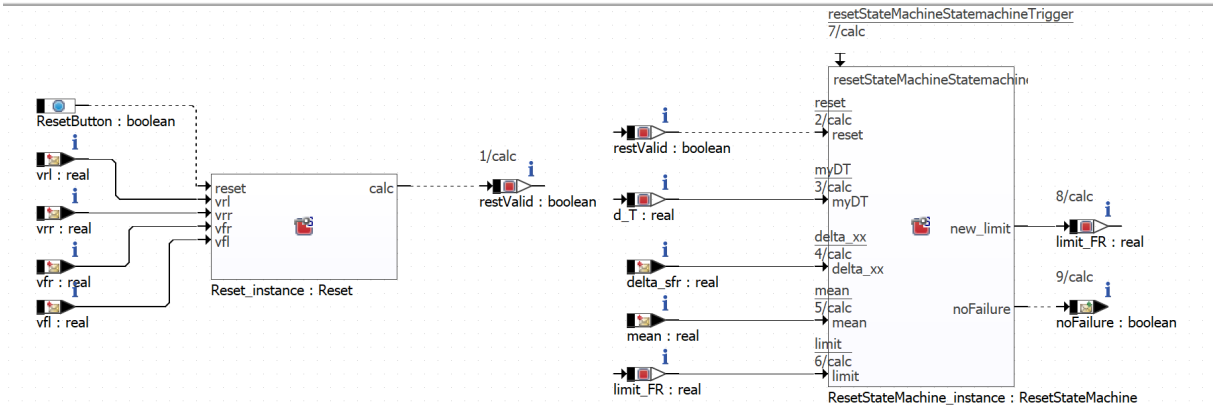


Abbildung 12.2: Reset

Die Statemachine ist vierfach für jeden Reifen ausgeführt. Sie besitzt als Inputs, ob der Reset valid ist, das dT, das Streckendelta eines Reifens, den Durchschnitt aller Reifen, sowie das Limit bei dem ein Failure erkannt wird. Als Output gibt die Statemachine das neue Limit sowie eine Message an, die das Signalisieren eines zu hohen oder zu niedrigen Reifendurcks unterdrücken kann.
Die Reset-Statemachine besitzt vier States.

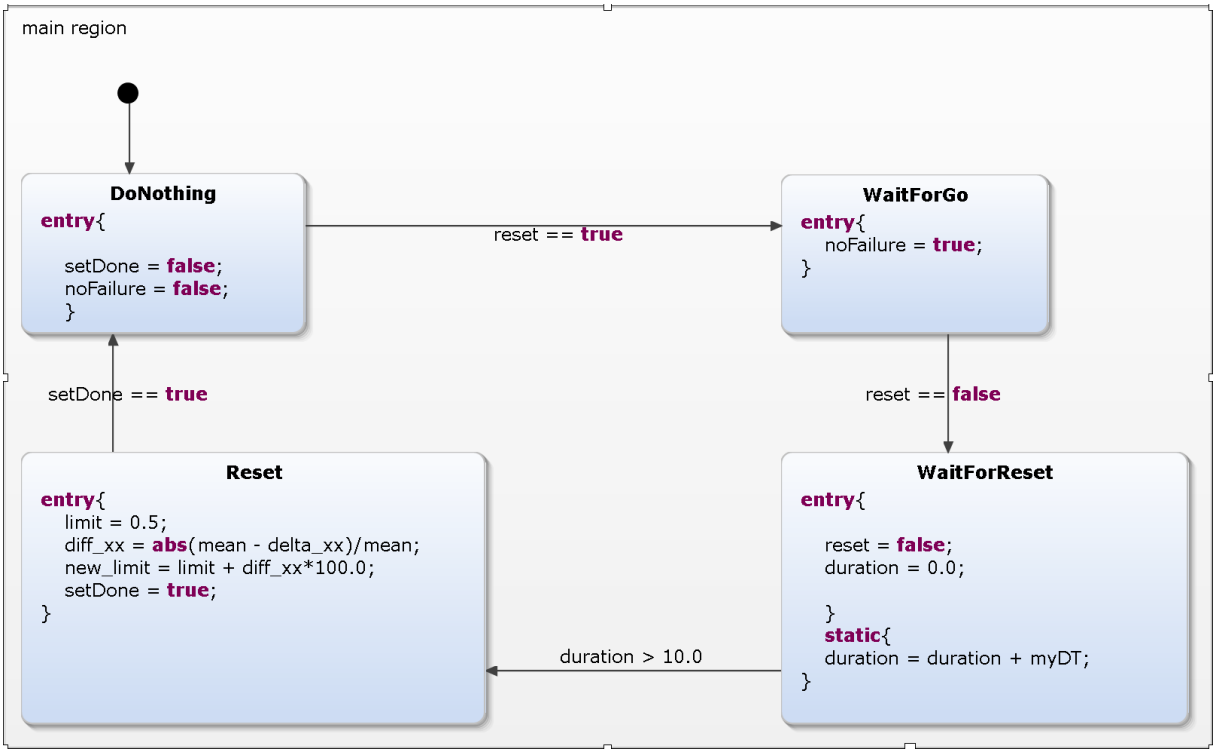


Abbildung 12.3: Reset Statemachine

Im DoNothing-State befindet sich die Statemachine solange die Reset-Taste nicht gedrückt bzw. der Reset nicht akzeptiert wurde. Liegt ein valider Reset vor wechselt die

Statemachine in WaitForGo. Ab diesem State werden vorerst alle Reifenfehlermeldungen unterdrückt, da während einem Reset ja kein Fehler auftauchen soll. Fährt das Auto los, beginnt der eigentliche Reset des Systems. Durch das Losfahren ist der Reset-Parameter nicht mehr aktiv und die Statemachine wechselt in den State WaitForReset. In diesem State wartet das System, bis es sich sich eingeschwungen hat, sodass Noise das System nicht verfälscht. Ab einer gewissen Zeit wechselt die Statemachine in den Reset-Zustand. In diesem wird nun, wie auch in Abbildung 7.12, die momentane prozentuale Abweichung berechnet und auf das Standardlimit addiert. Dadurch wird die Abweichung für diesen Reifen um soviel größer, wie der Reifendruck momentan abweicht. Weicht der Reifendruck dann von dem neu kalibrierten Wert wieder 0.5% ab, wird wieder ein Failure erkannt. Nach dem erfolgreichen Reset wechselt die Statemachine in DoNothing, wo auch wieder das Erkennen von Reifendruckabfällen freigeschaltet wird.
Dieser Reset wird parallel in drei weiteren Statemachines für die anderen Reifen durchgeführt.

Um dies im Environment zu testen, wird vor Start mit Hilfe des Error-Moduls, der Reifendruck der Reifen manipuliert und der Reset aktiviert. Neue Limits werden nach dem Losfahren des Fahrzeugs generiert.

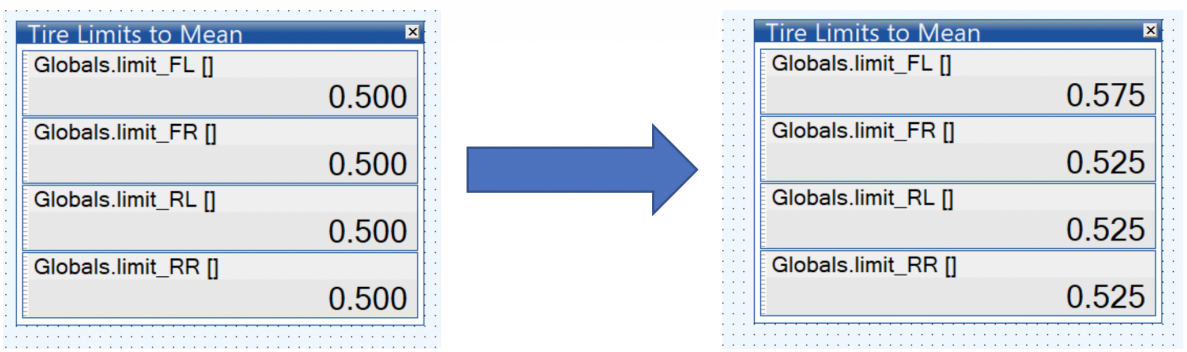


Abbildung 12.4: Neue Limits nach Reset

Die neuen Limits sind keineswegs größer. In ihnen ist nur schon die Abweichung der Reifen zum Normalzustand miteinberechnet, sodass sich bei der Berechnung nichts ändert.

13 Aufgabe D13

Wenn die Analyse nicht nur Geradeausfahrten sondern auch Kurvenfahrten miteinbeziehen soll, müssen wesentlich mehr Einflussfaktoren miteinbezogen werden, als bei der Geradeausfahrt.
Wird eine Kurve durchfahren, so müssen die äußeren Räder eine weitere Strecke fahren als die Inneren. Dies wurde bereits bei Aufgabe D2 in Abbildung 2.4 dargestellt. Das führt dazu, dass die Strecken/Geschwindigkeiten, auch ohne unterschiedlichen Reifendruck, voneinander abweichen und einen nicht vorhandenen Fehler detektieren.
Werden nun auch Kurvenfahrten betrachtet, so müssen die Drehwinkel Θ der Vorderreifen miteinbezogen werden. Anhand dieses Drehwinkels und dem Reifenabstand kann der äußere und innere Radius berechnet werden, der Auswirkung auf die Geschwindigkeiten hat.

Aus (5): $R_{rl} = \dfrac{B}{tan(\theta_L))}$

Aus (7): $R_{rl} = \dfrac{W}{\dfrac{V_{rr}}{Vrl} - 1}$

$$\frac{B}{tan(\theta_L)} = \frac{W}{\frac{V_{rr}}{V_{rl}} - 1}$$

$$\frac{tan(\theta_L)}{B} = \frac{\frac{V_{rr}}{V_{rl}} - 1}{W}$$

$$\frac{tan(\theta_L) * W}{B} = \frac{V_{rr}}{V_{rl}} - 1$$

$$\frac{tan(\theta_L) * W}{B} + 1 = \frac{V_{rr}}{V_{rl}}$$

$$\frac{tan(\theta_L) * W * V_{rl}}{B} + V_{rl} = V'_{rr} \quad (\ 11\ )$$

Die abweichende Radgeschwindigkeit müsste nun in der Berechnung der prozentualen Abweichungen beachtet werden.
Eine Implementierung wurde als zu aufwendig betrachtet, da es das System auf fundamentaler Ebene beeinträchtigt. Deswegen wurde entschieden ein robustes System zur Geradeausfahrt beizubehalten.

14 Aufgabe D14

Deltazeit

In der gesamten Umsetzung wurden 10 Sekunden zur Deltagenerierung genutzt. Durch diese 10 Sekunden wurde kleines Rauschen/Noise bei der Erkennung von Reifendruckabfällen unterdrückt. Auch zum Reset wurde deswegen die Einschwingdauer von 10 Sekunden verwendet. Sollte die Gesamtfunktionalität jedoch tatsächlich im Fahrzeug implementiert werden, sollte eine höhere Deltatime gewählt werden, da Fahrzeiten deutlich höher sind. In der Simulation wurden jedoch nur 10 Sekunden gewählt, um auch ein schnelles Testen zu gewährleisten.

Änderbarkeit des Codes/Modell

Gerade in ASCET sind Größen, wie bspw. die Größe des Buffers, nicht variabel und können nur bei der Definition des Buffers festgelegt werden. Es wurde nach Möglichkeiten gesucht, wie das Modell/Code anpassbarer gemacht werden könnte. Es ist jedoch in ASCET nicht möglich gewesen bspw. ein Array mit variabler Länge zu erstellen.

Effizienz bei dem Bearbeiten der Aufgaben

Die Aufgaben sind teilweise unpräzise gestellt, sodass es viel Interpretationsspielraum gibt. Häufig ist es vorgekommen, dass nach erstem Lösen einer Aufgabe eine neue Interpretation entstanden ist, sodass die Aufgabe nochmal überarbeitet werden musste. Beispielhaft sind dafür die Aufgaben D2/D3/D4.
Das Programmieren mit ASCET geht im Idealfall schnell. Treten jedoch Fehler auf, sprengen diese meist die eingeplante Zeit, da durch ASCET wenig Hilfestellung zur Lösung der Probleme geleistet wird.
Außerdem entstand der Eindruck, dass grafisches Programmieren langsamer und mühsamer ist, als die Implementierung in Skriptsprache. Ein Beispiel ist hierfür der Buffer, der in Skriptsprache umgesetzt wurde, da dies einfacher und übersichtlicher ist.