

Dokumentation – „TirePressureMonitor“

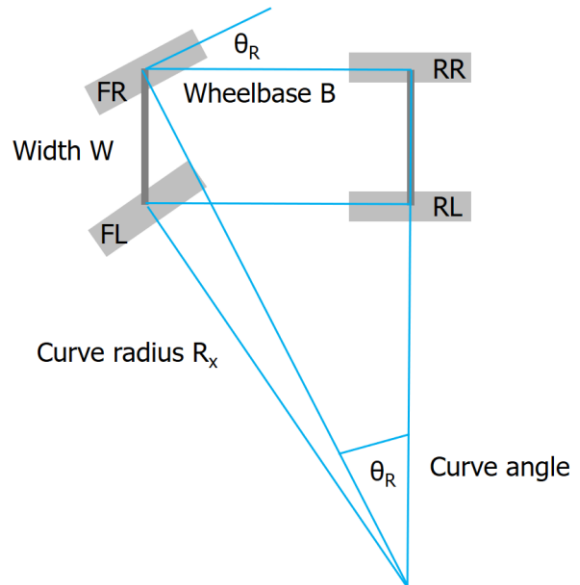
Matrikelnummern: 9234269, 5333856

Inhaltsverzeichnis

Formeln.....	2
Anforderungen.....	2
Aufgaben	3
Implementierung der Aufgaben.....	4
Aufgabe D1	4
Aufgabe D2	5
Aufgaben D3 und D4.....	6
Aufgaben D5 und D6.....	8
Aufgaben D7, D9, D10, D11 und D12.....	10
Aufgabe D8	15

Formeln

1. $R_{RR} = W + R_{RL}$
2. $R_{FR}^2 = B^2 + R_{RR}^2$
3. $R_{FL}^2 = B^2 + R_{RL}^2$
4. $B = R_{RR} \tan \theta_R$
5. $B = R_{RL} \tan \theta_L$
6. $V_x / V_y = R_x / R_y$ for any x, y
from $\{RR, RL, FR, FL\}$



Anforderungen

ID	Description
R1	A tire pressure monitor allows observation of the four wheel speeds to detect an unexpected imbalance for a vehicle with size $W = 1.53$ m and $B = 2.65$ m.
R2	An imbalance of a wheel speed of 0.5% of one of the wheels concerning the expected consistent wheel speed 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 time short, three times long, three times short) sound shall appear (base rate 0.8 s).
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.

Aufgaben

ID	Description
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 $X(i) = (a * X(i-1) + c) \bmod m$ 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 D5 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.

Implementierung der Aufgaben

Aufgabe D1

Nach der 3-Punkt-Schätzung wurde die voraussichtliche Entwicklungszeit berechnet. Hierbei gibt man für die jeweiligen Aufgaben (Anzahl = N) jeweils die Zeit für den best-case (bc), den worst-case (wc) und den likely-case (lc) an.

Der grobe Schätzwert für die Bearbeitungszeit einer Aufgabe wird anschließend mit der Formel

$$estimate = \frac{lc + 4 * lc + wc}{6}$$

berechnet.

Um nun die gesamte Bearbeitungszeit abschätzen zu können, muss man zuerst die Standardabweichung durch die Formel

$$standarddeviation = \frac{wc - bc}{6}$$

und den Standardfehler durch die Formel

$$standarderror = \frac{standarddeviation}{N}$$

berechnen.

Der Schätzwert der gesamten Bearbeitungszeit ergibt sich, indem man die Formel

$$projecttime = \sum estimate \pm 2 * \sqrt{\sum (standarderror)^2}$$

verwendet.

Task	Best Case	Likely Case	Worst Case		Estimate	Standard deviation	Standard Error		Actual Time
D1	5	10	15		10	1,666666667	0,502518908		10
D2	30	45	60		45	5	1,507556723		65
D3	20	30	40		30	3,333333333	1,005037815		40
D4	15	25	35		25	3,333333333	1,005037815		45
D5	15	20	25		20	1,666666667	0,502518908		15
D6	20	30	40		30	3,333333333	1,005037815		15
D7	75	90	105		90	5	1,507556723		100
D8	60	75	90		75	5	1,507556723		30
D9	15	20	25		20	1,666666667	0,502518908		25
D10	20	25	30		25	1,666666667	0,502518908		25
D11	45	60	75		60	5	1,507556723		70
E(project)					430				
SE(project)							3,623715377		
Project Time (-)							422,7525692		
Project Time (+)							437,2474308		
Avg Project Time (in h)							7,17		
Actual Project Time (in h)									7,33

Abbildung 1 – Projektzeitschätzung

Aufgabe D2

Um die beispielhaften Fahrdaten für das Reifendruckmodell in die aktuelle Matlab-Umgebung laden zu können, muss der Befehl `load('curve.mat')` benutzt werden. Der Kurvenradius lässt sich mittels der Funktion, welche die gegebenen Formeln aus dem ersten Kapitel dieser Dokumentation verwendet, in Abbildung 2 berechnen.

```
function RFL = calcCurveRadiuses(vx, vy, W, B)
    RRL = W./((vx./vy)-1);
    RFL = sqrt(B^2 + RRL.^2);
end
```

Abbildung 2 - Berechnung des Kurvenradius

Anschließend werden alle vorhandenen Daten visualisiert, woraus sich die Graphen der Abbildung 3 ergeben.

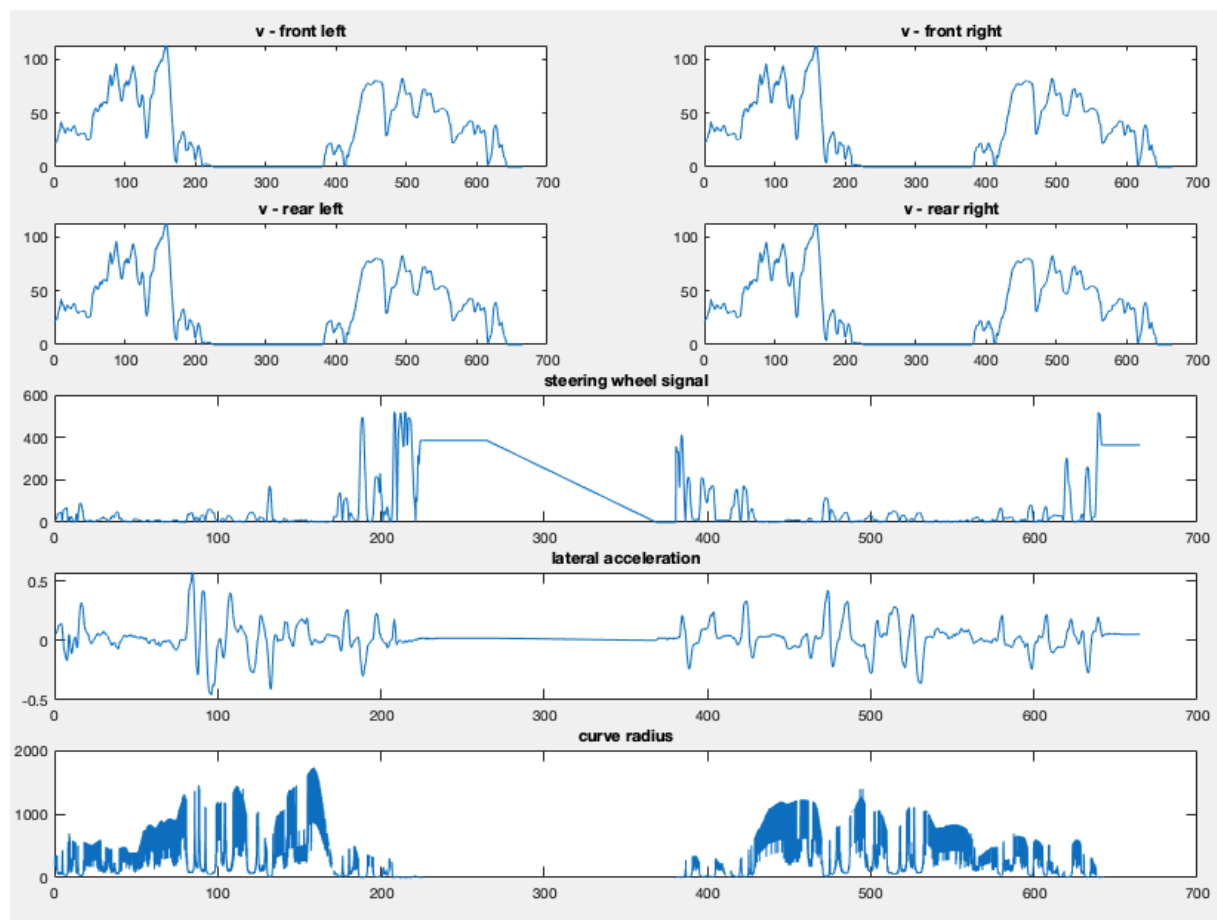


Abbildung 3 - Visualisierung der beispielhaften Fahrdaten

Die Visualisierungen des Kurvenradiuses sowie der lateralen Beschleunigung lassen deuten, dass es sich keinesfalls um eine gerade Strecke handeln kann.

Aufgaben D3 und D4

Das Simulink Modell, um die beispielhaften Fahrdaten zur Berechnung der Fahrdistanzen der einzelnen Räder sowie deren Abweichungen nach der Anforderung R2 zu berechnen, ist in der Abbildung 4 ersichtlich. Hierbei werden die jeweiligen Geschwindigkeiten zuerst von der Einheit km/h in die Einheit m/s umgerechnet und anschließend integriert, um die Fahrdistanz zu erhalten.

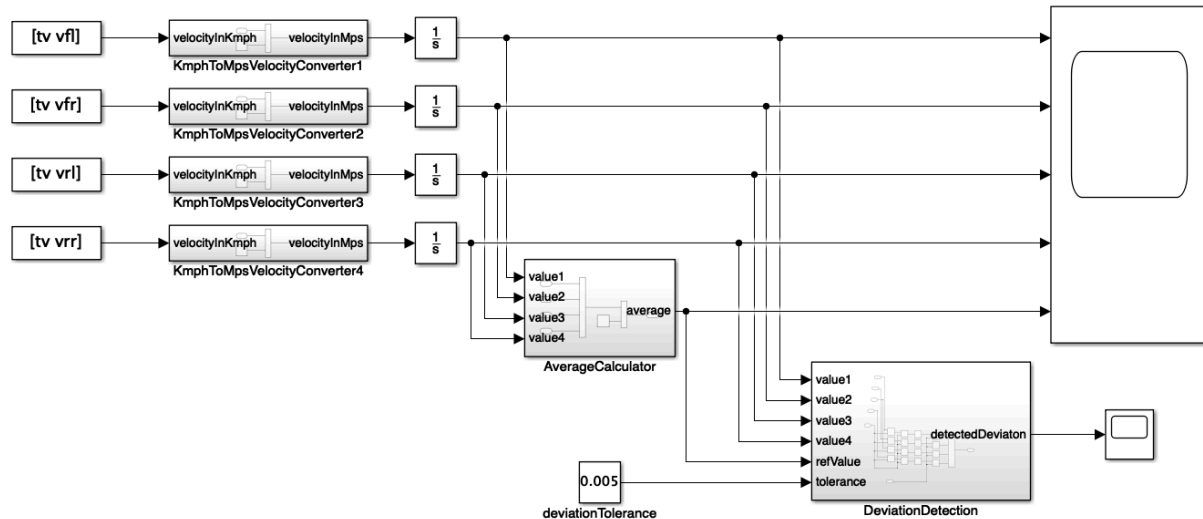


Abbildung 4 - Simulink Modell zur Analyse der Fahrdistanzen und derer Abweichungen

Damit etwaige Abweichungen erkannt werden können, muss der Durchschnitt der Fahrdistanzen berechnet werden, um diesen als Referenzwert zu nehmen.

Zur Erkennung der Abweichungen der Fahrdistanzen wird das in der Abbildung 5 ersichtliche Subsystem „DeviationDetection“ verwendet.

Die Toleranzschwelle beträgt nach der Anforderung R2 0,5%.

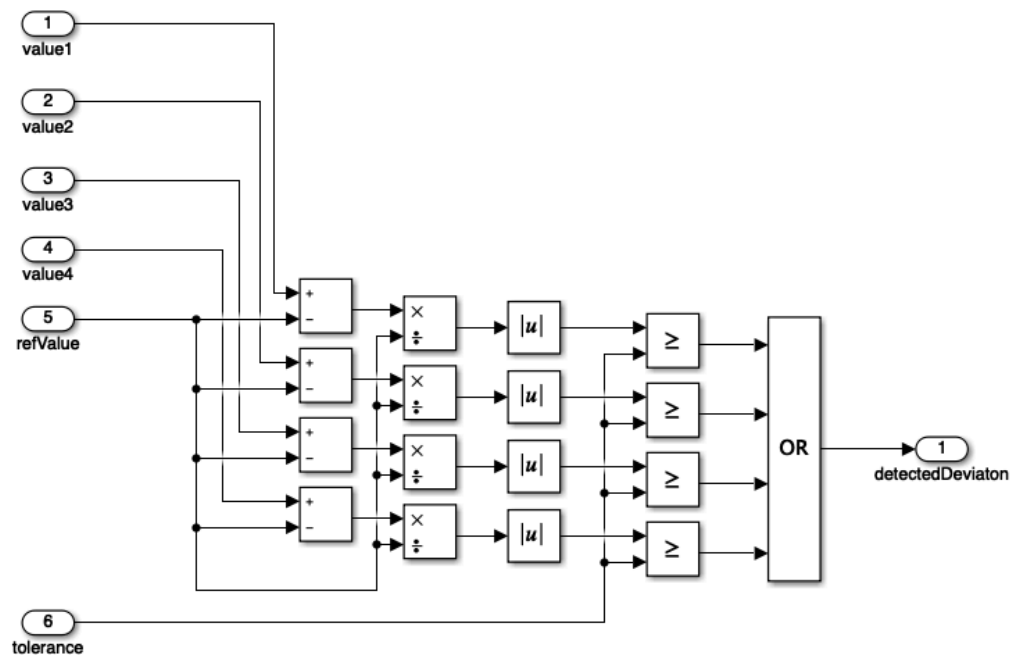


Abbildung 5 - Simulink Subsystem zur Erkennung von Abweichungen

Führt man dieses Modell aus, so ergibt sich im unteren Oszilloskop das binäre Signal aus der Abbildung 6. Eine binäre 1 bedeutet eine Abweichung trotz der Toleranzschwelle. Hierbei wurde eine Simulationszeit von 1000s gewählt, da dies alle Datensätze der beispielhaften Fahrdaten umfasst.

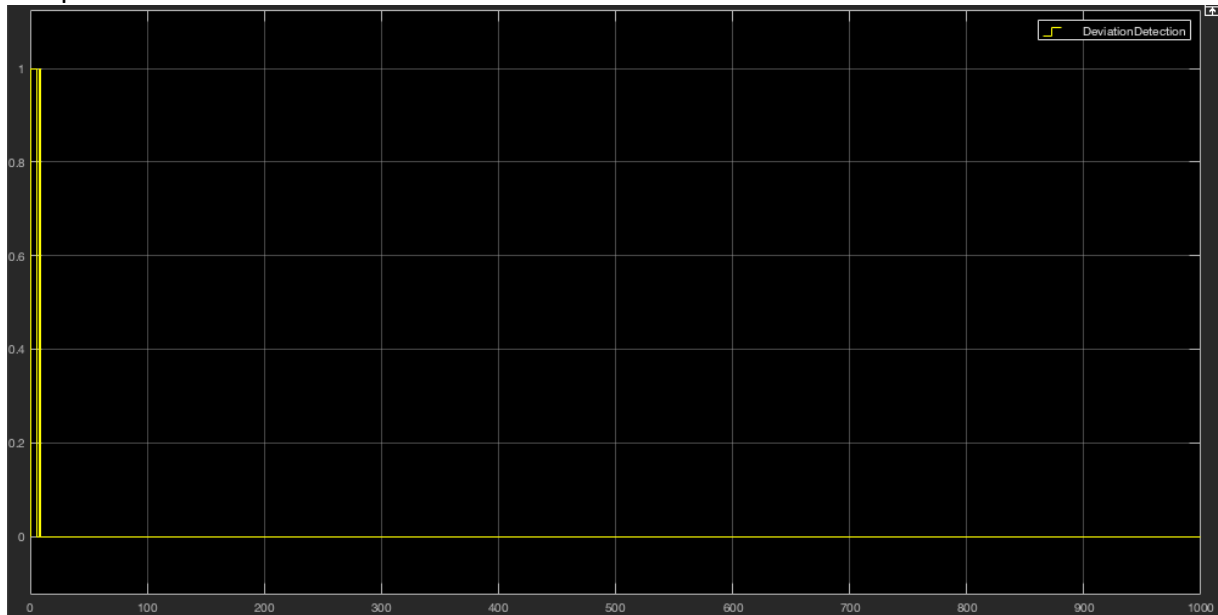


Abbildung 6 - Das Oszilloskop zeigt etwaige Abweichungen der Fahrdistanzen an

Es lässt sich erkennen, dass es zum Anfang der Analyse teilweise Ausschläge gibt, da sich das Modell erst „kalibrieren“ muss. Dies liegt daran, dass die integrierten Fahrdistanzen zu Beginn wenig Aussagekraft besitzen.

Aufgaben D5 und D6

Der implementierte, linear kongruente Zufallszahlengenerator ist in der Abbildung 7 zu sehen.

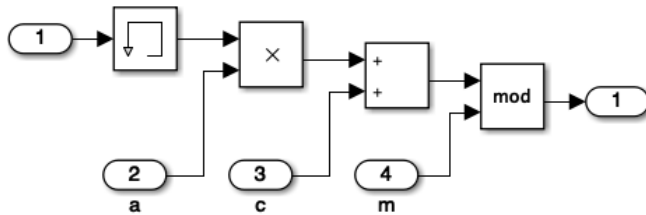


Abbildung 7 - Zufallszahlengenerator als Simulink Modell

Als Parameterwerte wurden $a=15$, $c=26$ und $m=28$ für Räder mit korrektem Reifendruck und $a=15$, $c=27$ und $m=27$ für Räder mit abweichendem Reifendruck gewählt.

Analog zu den Aufgaben D3 und D4 wurde das in der Abbildung 8 ersichtliche Simulink Modell implementiert, um den Reifendruckmonitor durch Zufallszahlen zu testen.

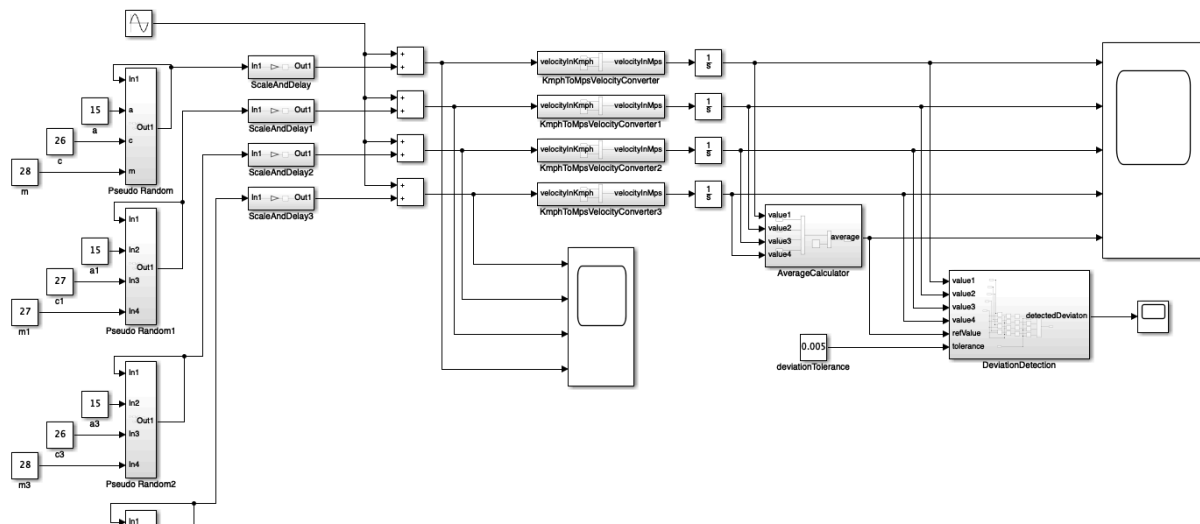


Abbildung 8 - Simulink Modell zum Testen des Reifendruckmonitors mittels Zufallszahlen

Hierbei wird das Rauschen durch die Zufallsgeneratoren kleiner skaliert und um 300s verzögert. Anschließend wird das Rauschen auf eine Sinuskurve mit der Amplitude 20 addiert, um schlussendlich die Geschwindigkeit zu simulieren.

Die Geschwindigkeiten der einzelnen Räder über den Simulationszeitraum lassen sich in der Abbildung 9 einsehen. Hier wird auch deutlich, dass das Rauschen erst ab knapp über 300s inkludiert wird, was der Berechnung der Fahrdistanzen durch Integration zuschulden kommt.

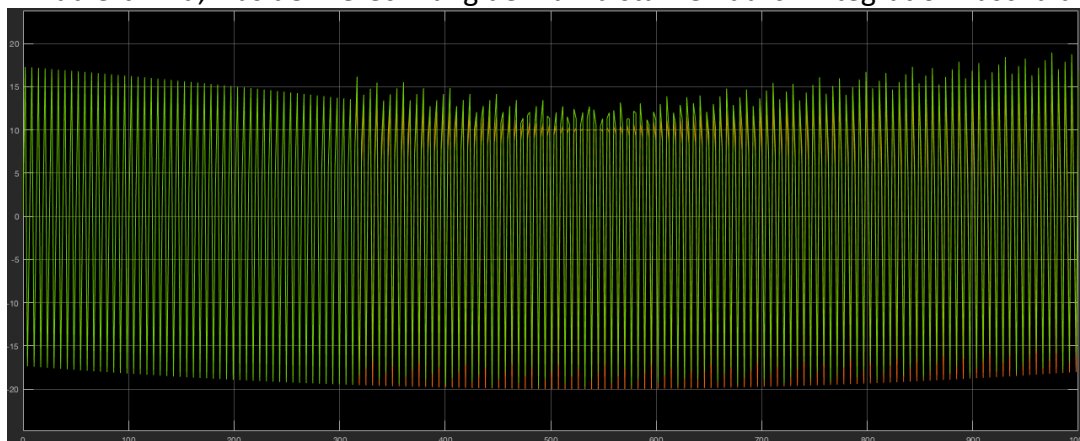


Abbildung 9 - Die Radgeschwindigkeiten inklusive Rauschen

Das um 300s verzögerte Rauschen wird anschließend im Oszilloskop, welches die erkannten Abweichungen darstellt, angezeigt, was in der Abbildung 10 zu sehen ist.

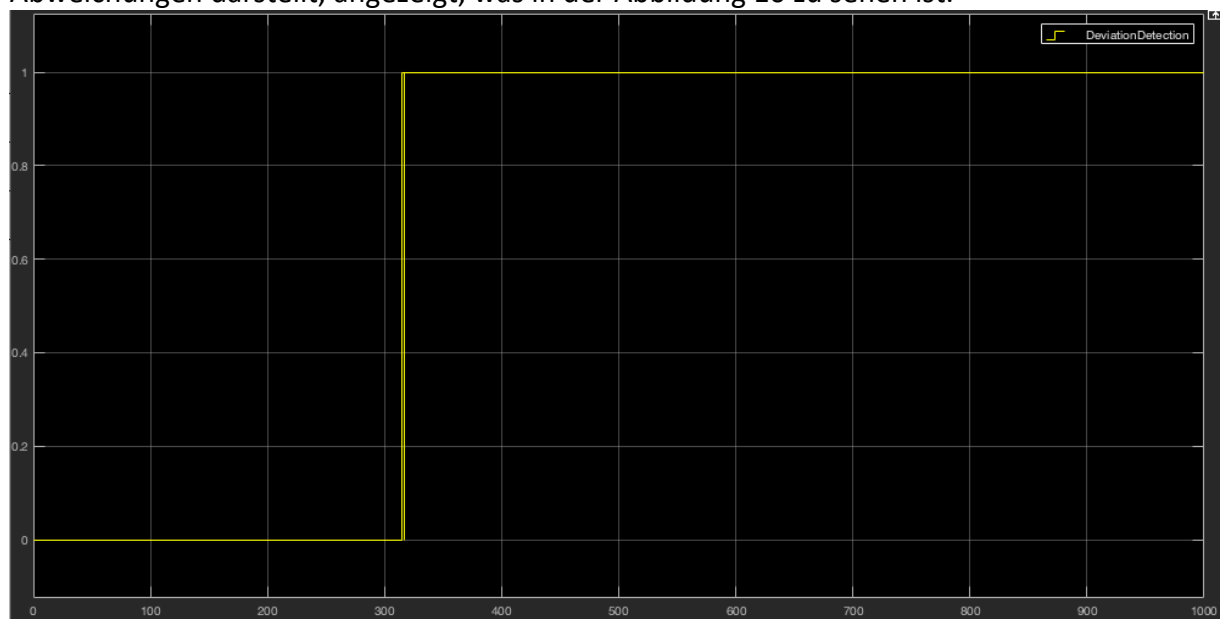


Abbildung 10 - Erkennung der durch Rauschen erzeugten Abweichungen

Aufgaben D7, D9, D10, D11 und D12

Um das gesamte System in Ascet zu konvertieren, wurden wieder die gleichen Modelle mit vergleichbaren Komponenten implementiert. Die Implementierungen sind mit den jeweiligen Simulink Äquivalenten vergleichbar.

Das Modell zur Berechnung der Durchschnittswerte wurde nach Abbildung 11 entworfen.

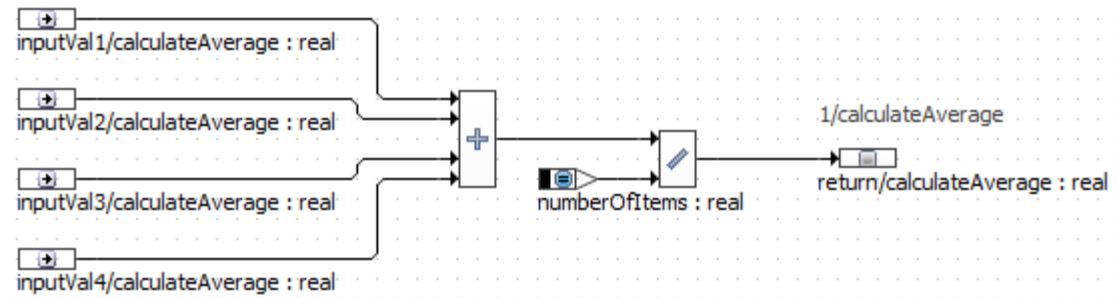


Abbildung 11 - Ascet Modell zur Berechnung der Durchschnittswerte

Da Ascet nicht über eine Komponente zur elementaren Integration von Zahlenwerten verfügt, wurde diese zusätzlich nach Abbildung 12 implementiert.

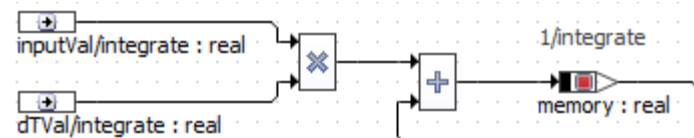


Abbildung 12 - Ascet Modell zur Integration

Um Zufallszahlen generieren zu können, wurde das zugehörige Modell, wie in der Abbildung 13 ersichtlich ist, implementiert.

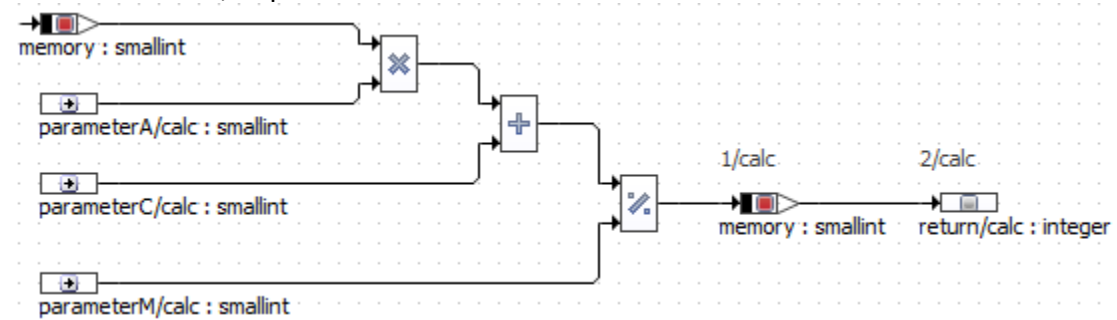


Abbildung 13 - Ascet Modell zur Zufallszahlengenerierung

Auch das Modell zur Abweichungserkennung findet ebenfalls Einzug in Ascet, wie in Abbildung 14 zu erkennen ist.

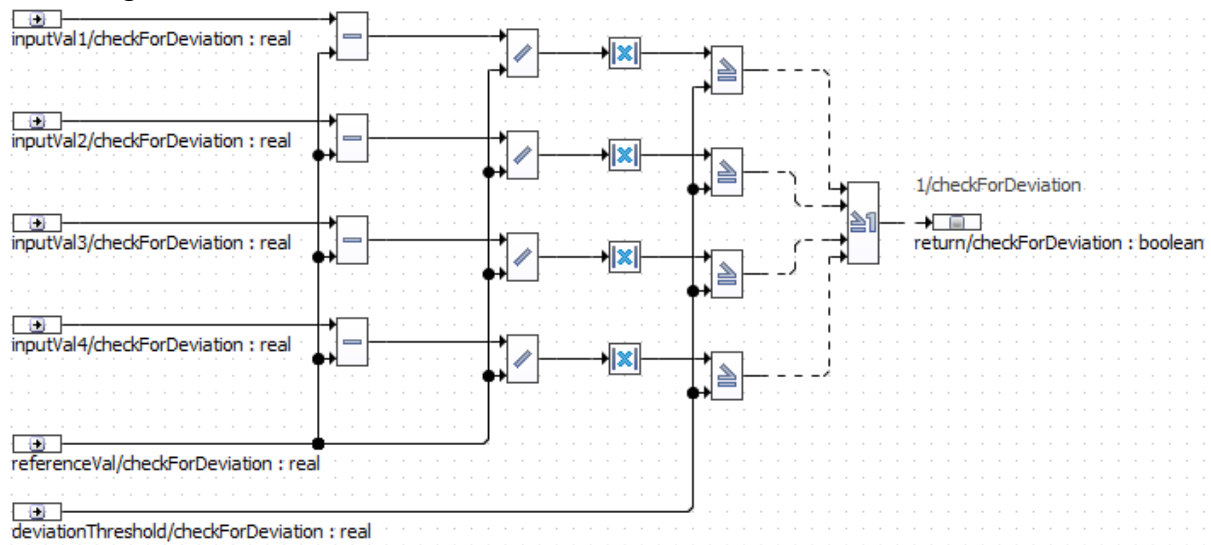


Abbildung 14 - Ascet Modell zur Abweichungserkennung

Um Nachrichten zwischen den Komponenten austauschen zu können, müssen diese wie in Abbildung 15 implizit deklariert werden.

```
data interface velocityMessage {
    real vFrontLeft = 0.0;
    real vFrontRight = 0.0;
    real vRearLeft = 0.0;
    real vRearRight = 0.0;
}

data interface distanceMessage {
    real distanceFrontLeft = 0.0;
    real distanceFrontRight = 0.0;
    real distanceRearLeft = 0.0;
    real distanceRearRight = 0.0;
    real averageDistance = 0.0;
}

data interface deviationMessage {
    boolean deviationDetected = false;
    boolean recalibrate = false;
}
```

Abbildung 15 - Deklaration von Nachrichten in Ascet

Die Warnlampe mit dem SOS-Signal, wie in der Aufgabe D9 gefordert ist, wurde durch eine Ascet Statemachine nach Abbildung 16 entworfen. Diesem Modell muss die `deltaT` Variable als Parameter übergeben werden, damit die zeitliche Abfolge gewährleistet ist. Solange am Statemachine Trigger ein boolean-Wert „true“ anliegt, wird die Statemachine in einer Endlosschleife weitergeführt.

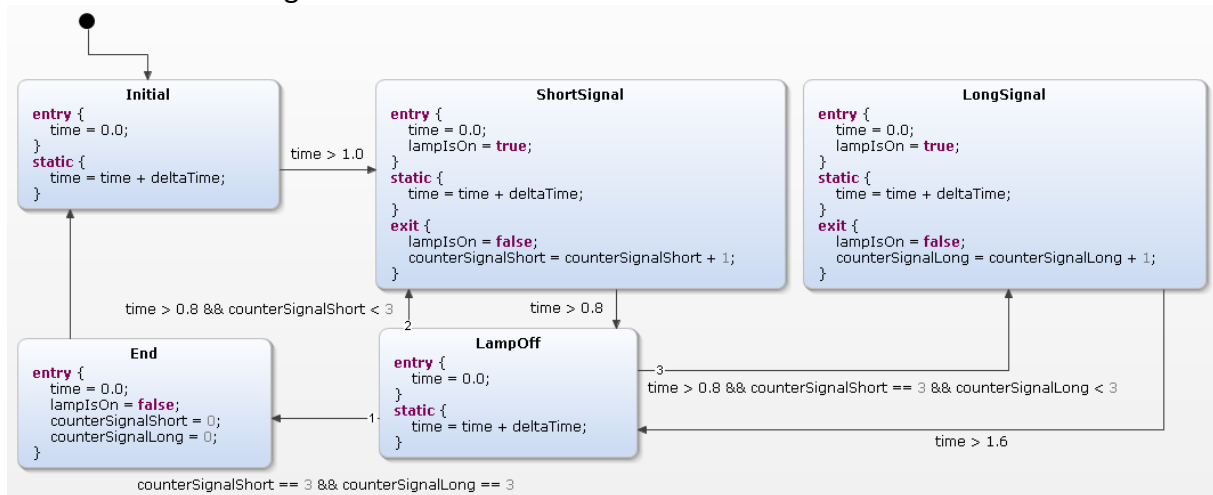


Abbildung 16 - Ascet Statemachine für die SOS Lampe

In der Abbildung 17 ist die Überführung des Reifendruckmonitors von Simulink zu Ascet ersichtlich. Hierbei wurde eine Möglichkeit zur Rekalibrierung des Systems implementiert, indem die internen Speicher der Integratormodelle über eine eigens dafür implementierte Funktion zurückgesetzt werden. Ein weiterer Unterschied zum Simulink System besteht in dem Einbringen der Warnlampe. Die zur Warnlampe zugehörige Statemachine bricht instantan ihre Weiterführung der Endlosschleife ab, sofern kein boolean-Wert „true“ mehr am Trigger anliegt. Dies führt potentiell dazu, dass die SOS-Lampe aktiv bleibt, obwohl die Statemachine gestoppt wurde. Daher wurde ein manuelles Zurücksetzen des Lampenstatus implementiert, was durch eine Funktion von außen ausgeführt werden kann.

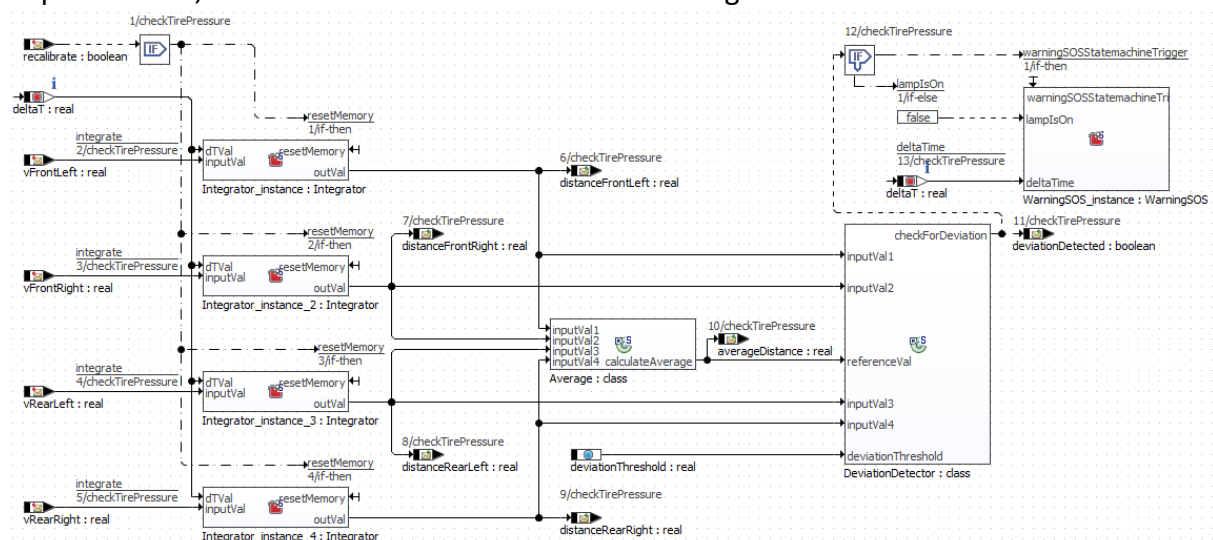


Abbildung 17 - Ascet Modell des Reifendruckmonitors

Das Systemtestmodell, gefordert durch die Aufgabe D11, wurde nach Abbildung 18 implementiert. Hierbei wird der Zufallszahlengenerator verwendet, dessen erzeugte Zahlen skaliert werden, um diese als Bias-Wert für das Basissignal zu nutzen. Das sich daraus ergebende Signal wird anschließend als Nachricht versendet. Des Weiteren befindet sich in diesem Modell die Möglichkeit zur Rekalibrierung, da in der Experiment Environment keine Nachricht direkt versendet werden kann und dies über eine Characteristic, als Umweg zu einer Nachricht, geschehen muss.

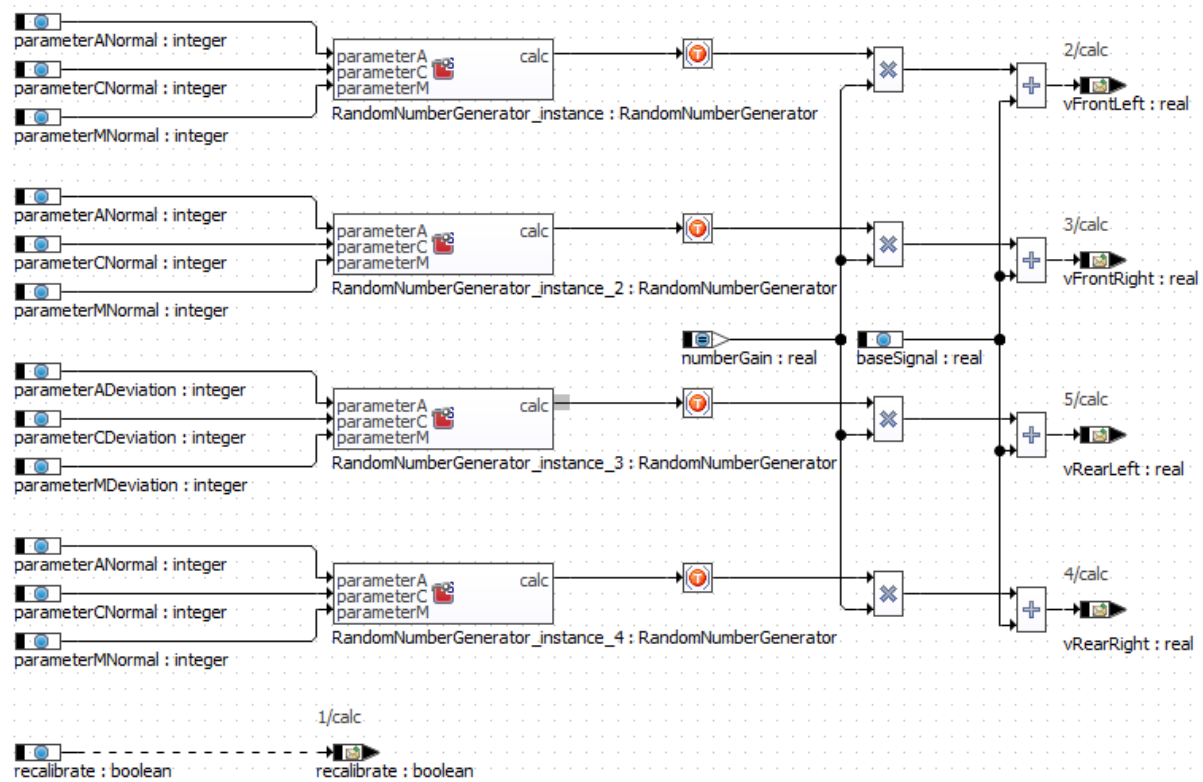


Abbildung 18 - Ascet Modell des Systemtests

Um das Systemtest Modell und das Reifendruckmonitor Modell in der Experiment Environment testen zu können, ist eine zugehörige App (Systemtest App) notwendig, die die Thread-Funktionen der beiden Modelle periodisch ausführt. Dies ist in der Abbildung 19 zu sehen.

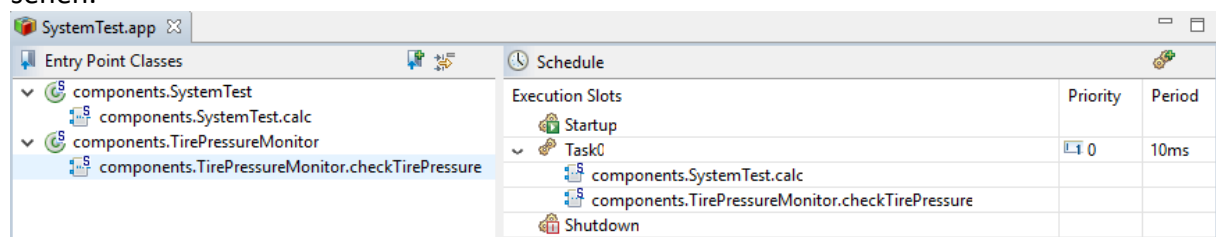


Abbildung 19 - Ascet App zur Ausführung des Systemtests und des Reifendruckmonitors

Nach Ausführung der Systemtest App lässt sich die Experiment Environment durch Drag&Drop wie in der Abbildung 20 konfigurieren. Auf der linken Seite befinden sich die Daten zum Rauschen, dem Basissignal, der Radgeschwindigkeiten und der Raddistanzen. In der Mitte befindet sich eine Anzeige der Abweichungstoleranz, der Abweichungsdetektion als Textfeld und als Oszilloskop und eine Möglichkeit zur Rekalibrierung per Button. Auf der rechten Seite befinden sich Daten zur Statemachine der Warnlampe und die Warnlampe an sich.

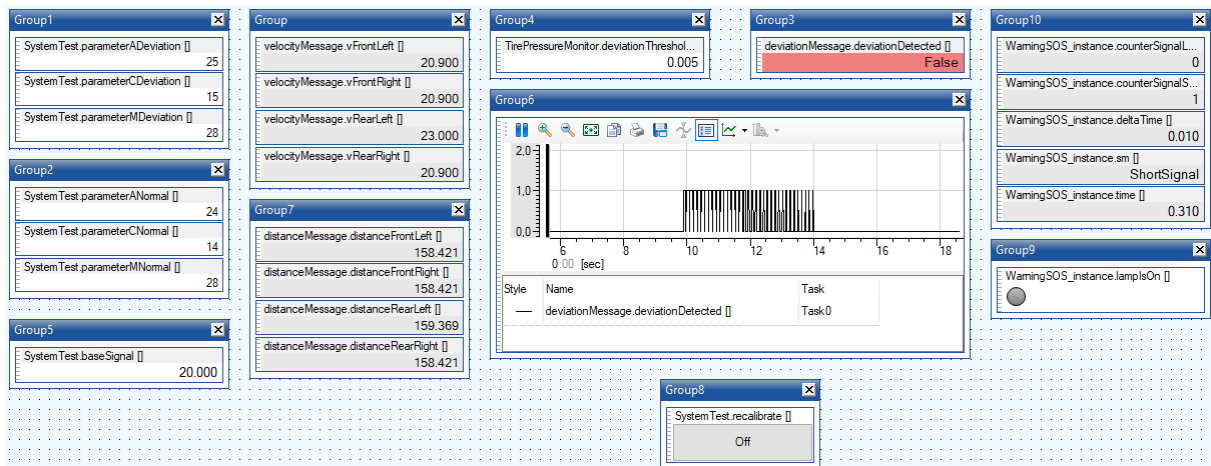


Abbildung 20 - Ascet Experiment Environment mit dem Systemtest und dem Reifendruckmonitor

Aufgabe D8

Um das Modell für die Durchschnittsberechnung testen zu können, wurden Unit Tests nach der Abbildung 21 entworfen und ausgeführt, was ein positives Ergebnis erbrachte.

```
static class AverageTest {
    @Test
    public void testAveragePositive() {
        real avgResult = Average.calculateAverage(1.0, 2.0, 3.0, 4.0);

        Assert.assertTrue(avgResult == 2.5);
    }

    @Test
    public void testAverageNegative() {
        real avgResult = Average.calculateAverage(-1.0, -2.0, -3.0, -4.0);

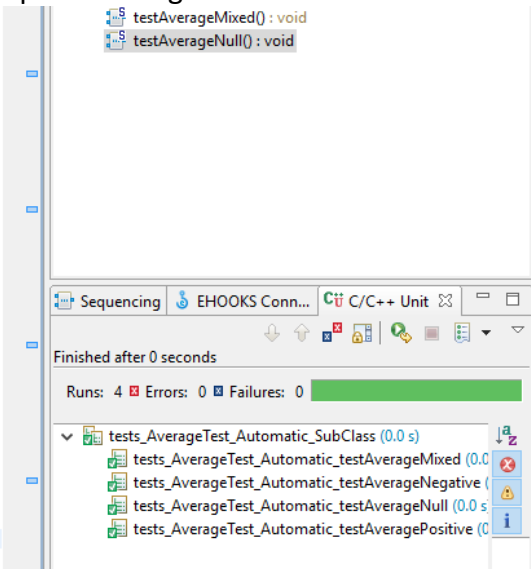
        Assert.assertTrue(avgResult == -2.5);
    }

    @Test
    public void testAverageMixed() {
        real avgResult = Average.calculateAverage(-1.0, 2.0, -3.0, 4.0);

        Assert.assertTrue(avgResult == 0.5);
    }

    @Test
    public void testAverageNull() {
        real avgResult = Average.calculateAverage(0.0, 0.0, 0.0, 0.0);

        Assert.assertTrue(avgResult == 0.0);
    }
}
```



The screenshot shows the C++ Unit Test runner interface. The top panel displays the test results for the `AverageTest` class. The tests `testAverageMixed()` and `testAverageNull()` are listed. The bottom panel shows the test results for the `tests_AverageTest_Automatic_SubClass` class. The tests `tests_AverageTest_Automatic_testAverageMixed`, `tests_AverageTest_Automatic_testAverageNegative`, `tests_AverageTest_Automatic_testAverageNull`, and `tests_AverageTest_Automatic_testAveragePositive` are listed. The overall status is "Finished after 0 seconds" with "Runs: 4", "Errors: 0", and "Failures: 0".

Abbildung 21 - Ascet Unit Tests für das Modell zur Durchschnittsberechnung

Das Modell zur Integration wurde durch Unit Tests nach Abbildung 22 getestet. Auch diese Tests ergaben ein positives Ergebnis.

```
static class IntegratorTest {
    Integrator integratorInstance;

    @Test
    public void testIntegrator() {
        real integrationResult = 0.0;

        integratorInstance.integrate(1.5, 5.0);

        integrationResult = integratorInstance.outVal();

        Assert.assertTrue(integrationResult == 7.5);

        integratorInstance.integrate(1.5, 5.0);

        integrationResult = integratorInstance.outVal();

        Assert.assertTrue(integrationResult == 15.0);
    }

    @Test
    public void testIntegratorReset() {
        real integrationResult = 0.0;

        integratorInstance.integrate(1.5, 5.0);

        integrationResult = integratorInstance.outVal();

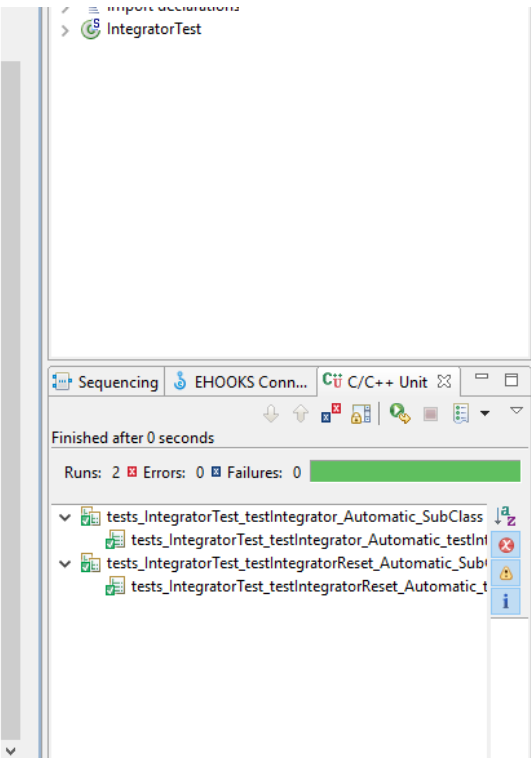
        Assert.assertTrue(integrationResult == 7.5);

        integratorInstance.resetMemory();

        integratorInstance.integrate(1.5, 5.0);

        integrationResult = integratorInstance.outVal();

        Assert.assertTrue(integrationResult == 7.5);
    }
}
```



The screenshot shows the C++ Unit Test runner interface. The top panel displays the test results for the `IntegratorTest` class. The tests `testIntegratorReset()` and `testIntegrator()` are listed. The bottom panel shows the test results for the `tests_IntegratorTest_testIntegrator_Automatic_SubClass` class. The tests `tests_IntegratorTest_testIntegratorReset_Automatic_SubClass` and `tests_IntegratorTest_testIntegratorReset_Automatic_SubClass` are listed. The overall status is "Finished after 0 seconds" with "Runs: 2", "Errors: 0", and "Failures: 0".

Abbildung 22 - Ascet Unit Tests für das Modell zur Integration

Die Unit Tests nach Abbildung 23 ergaben für das Modell zur Abweichungserkennung ein durchweg positives Ergebnis.

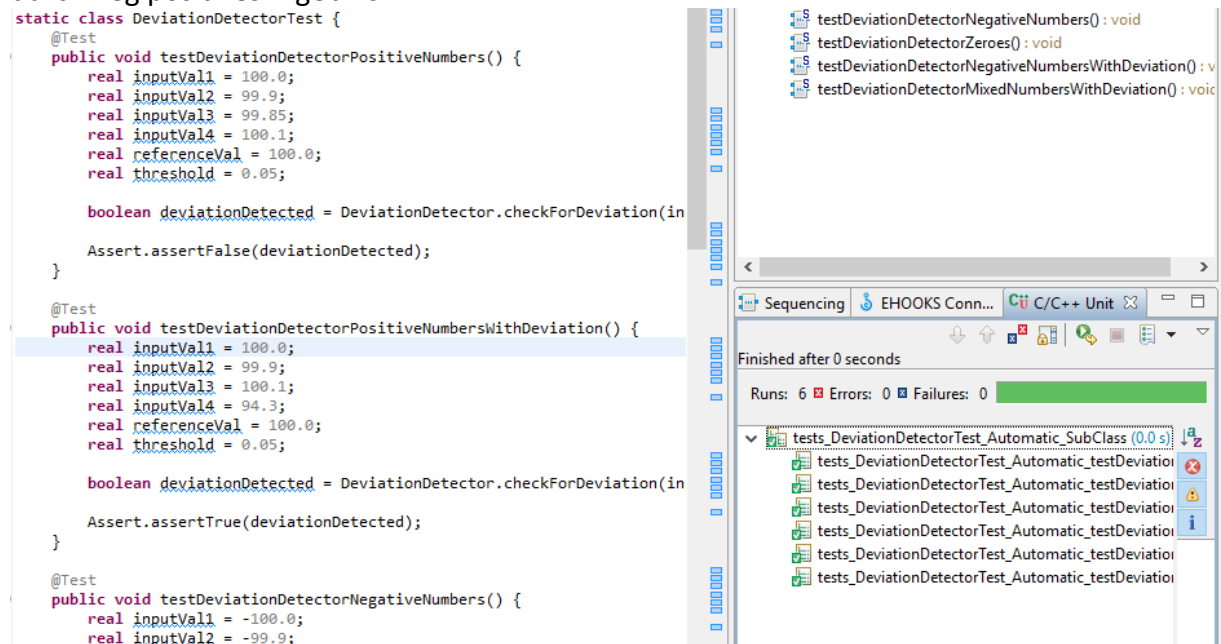


Abbildung 23 - Ascent Unit Tests für das Modell zur Abweichungserkennung

Um den Zufallsgenerator testen zu können, was in der Aufgabe D10 nochmals implizit gefordert wurde, wurden Unit Tests entworfen, wie in der Abbildung 24 ersichtlich ist.

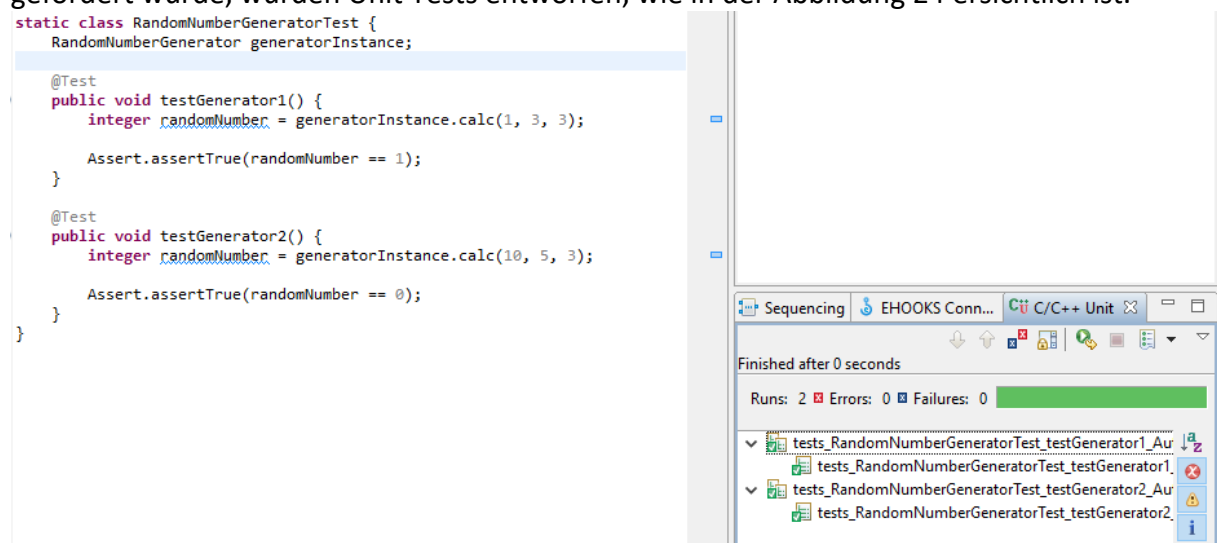


Abbildung 24 - Ascent Unit Tests für das Modell zum Zufallszahlengenerator