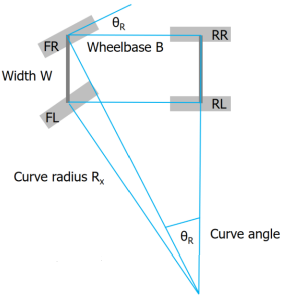
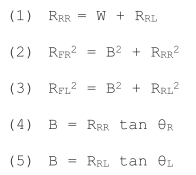
Tire Pressure Monitoring

DHBW-Stuttgart IT-Automotive Vorlesung: Grafische Programmierung

Von: 4851688, 6496528

Einführung und Überblick

Die Projektarbeit befasst sich mit der Thematik der Reifendruck-Überwachung. Detaillierte Anforderungen werden im folgenden Kapitel erläutert.

Die folgenden Gleichungen stellen eine Linkskurve dar.



Das Verhältnis zweier Geschwindigkeiten ist nach (6) proportional zum Verhältnis der entsprechenden Kurvenradien.

Ein Druckverlust im Reifen führt zu einer höheren Radgeschwindigkeit auf Grund des geringeren Umfangs.

Anforderungen und Aufgabenstellung

Tabelle

|  |  |
| --- | --- |
| **ID** | **Check** |
| 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 in 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. |

Tabelle

|  |  |
| --- | --- |
| 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 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. |

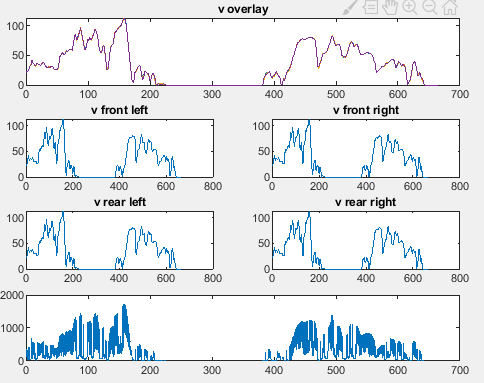
Realisierung und Dokumentation

## D1 - 3 Punkt Schätzung

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Task** | | **Tools** | | **opt** | **prob** | | **pess** | **mean** | **std. Deviantion** | **std. Error** | **(std. Error)²** | **act** |
| D1 | 3-Point-Estimate | |  | | 12 | 15 | | 20 | 15,3333333 | 1,333333333 | 0,402015126 | 0,161616162 | 15 |
| D2 | Analyse curve.mat | | MATLAB | | 20 | 30 | | 50 | 31,6666667 | 5 | 1,507556723 | 25 | 65 |
| D3 | driving model (driving distance for wheel) | | SIMULINK | | 30 | 45 | | 60 | 45 | 5 | 1,507556723 | 25 | 80 |
| D4 | Tire pressure monitoring | | SIMULINK | | 20 | 30 | | 45 | 30,8333333 | 4,166666667 | 1,256297269 | 17,36111111 | 120 |
| D5 | Random number generator | | MATLAB | | 15 | 30 | | 60 | 32,5 | 7,5 | 2,261335084 | 56,25 | 45 |
| D6 | Test modell with random number generator | | SIMULINK | | 20 | 35 | | 50 | 35 | 5 | 1,507556723 | 25 | 50 |
| D7 | Transfer pressure monitoring to ASCET | | ASCET | | 30 | 60 | | 90 | 60 | 10 | 3,015113446 | 100 | 120 |
| D8 | Unit Test for Pressure Monitoring | | ASCET | | 20 | 35 | | 60 | 36,6666667 | 6,666666667 | 2,010075631 | 44,44444444 | 60 |
| D9 | Implement warning function | | ASCET | | 10 | 20 | | 30 | 20 | 3,333333333 | 1,005037815 | 11,11111111 | 20 |
| D10 | Unit Tests for random number generatorr | | ASCET | | 20 | 40 | | 50 | 38,3333333 | 5 | 1,507556723 | 25 | 45 |
| D11 | System Test for system | | ASCET | | 30 | 60 | | 90 | 60 | 10 | 3,015113446 | 100 | 50 |
|  |  | |  | |  |  | |  |  |  |  | Time | 670 |
| **E(project)** | | 405,3333333 | |  | | | | | | | | | |
| **SE(project)** | | 20,72023848 | |
| **E(project, best)** | | 363,8928564 | | 6,06488094 | | |  | | | | | | | |
| **E(project, worst)** | | 446,7738103 | | 7,44623017 | | |
| Abgabe | | 26. März | |  | | | | | | | | | |

Die starken Abweichungen von den Schätzwerten lassen sich durch den Mangel an Erfahrung mit ASCET erklären. Viele Sachverhalte mussten durch Ausprobieren und Durchlesen der Hilfe erschlossen werden.

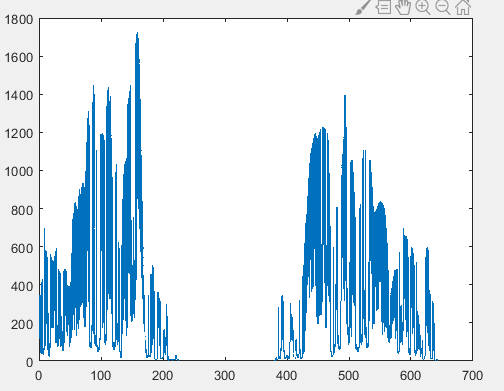
## D2 – Analysieren der curve.mat-Datei



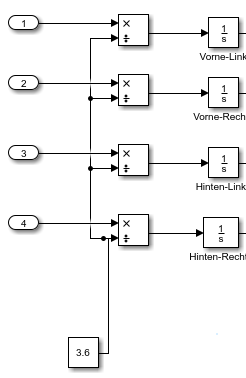
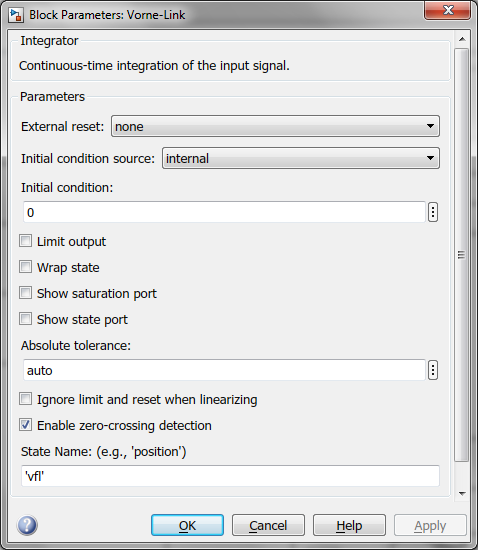
Der Plot gibt die Geschwindigkeiten der einzelnen Räder aus. Unten im Bild ist der Verlauf des Lenkradwinkels abgebildet. An den Geschwindigkeitsdiagrammen ist erkennbar, dass zwischen ca. 250 und 350 Einheiten auf der X-Achse die Messung pausiert wurde, da keine Geschwindigkeiten und kein Lenkradwinkel vorhanden sind.

Die Berechnung der Kurven Radien sind im Folgenden Plot dargestellt.

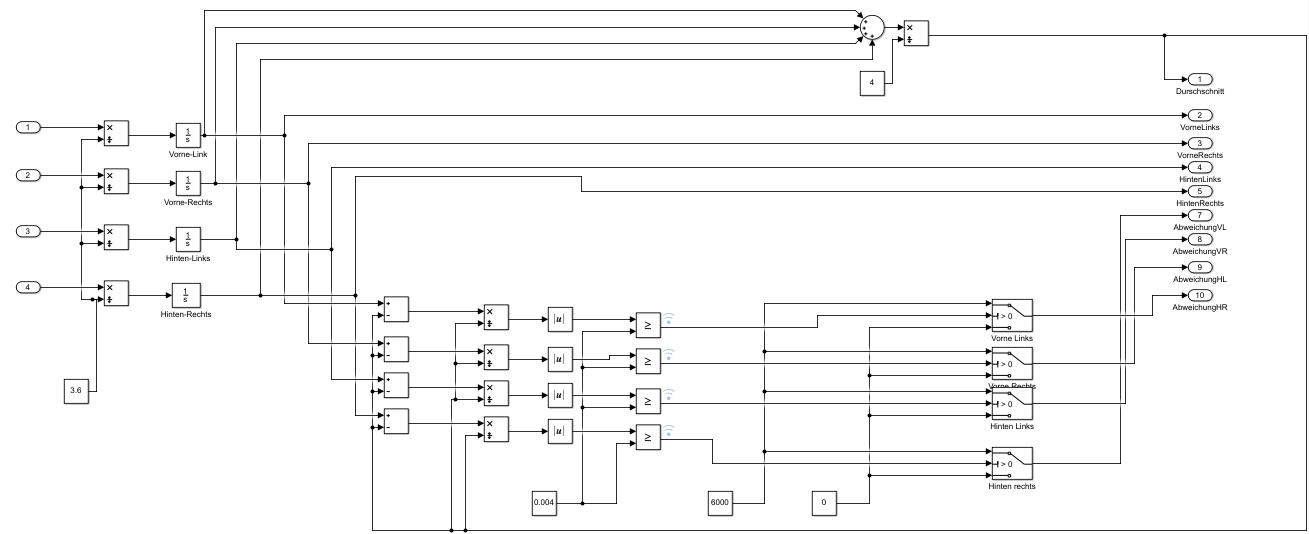
Einen einzelnen Kurvenradius kann man aus den Formeln (1-6) folgendermaßen berechnen:



## D3 – Simulink Modell für Radstrecken

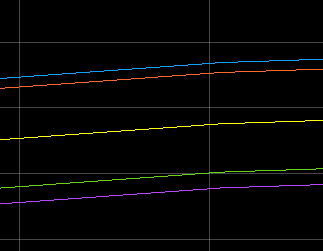
In der linken Abbildung sieht man den Teil des Modells, der die zurückgelegte Strecke pro Rad berechnet. Die einzelnen Radgeschwindigkeiten (Eingang 1-4) werden in m/s umgerechnet und danach integriert. In der rechten Abbildung sind die Einstellungen der genutzen Integratoren dargestellt. Die „Initial condition“ = 0 wurde gewählt, da wir von einer Strecken von 0m zum Beginn der Messung ausgehen.

## D4 – Simulink Modell für den Reifendruck-Monitor



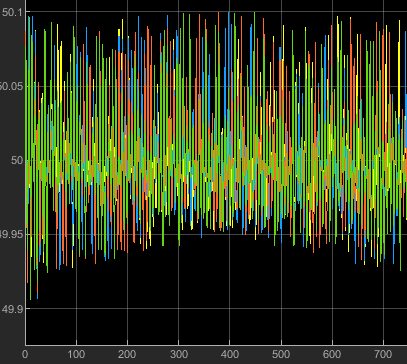
In der Abbildung auf der vorherigen Seite ist anfangs (unten) die Umrechnung der Geschwindigkeiten der einzelnen Räder in zurückgelegte Strecken erkennbar. Im oberen Teil wird der Durchschnitt der Strecken berechnet, der nachher als Vergleichswert für die einzelnen Räder dient. Unten wird die prozentuale Abweichung der einzelnen Radstrecken zu der Durchschnittsstrecke berechnet. Falls eine Abweichung (größer 0.4%) Auftritt wird ein vertikales Signal in der Simulation ausgegeben.

Die Kurve im Plot stellt die Durchschnittsstrecke und die einzelnen Radstrecken dar. Die Ausschläge am Anfang sind wenig aussagekräftig, da die Messung mit einer Kurve beginnt und der Durchschnitt diese Abweichungen zu diesem Zeitpunkt noch nicht ausgleichen kann.

Dieser Plot stellt eine Nahaufnahme der Streckenkurve dar. Hierbei ist erkenntlich, dass die Räder unterschiedliche Strecken gefahren sind. Gelb stellt den Durchschnitt der Radstrecken dar und liegt deshalb in der Mitte.

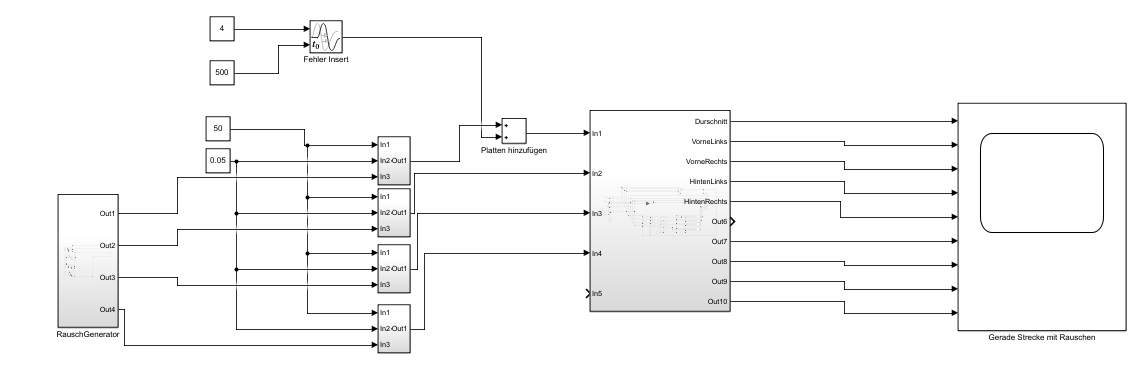
## D5 – Simulink Modell für Zufallszahlen Generator

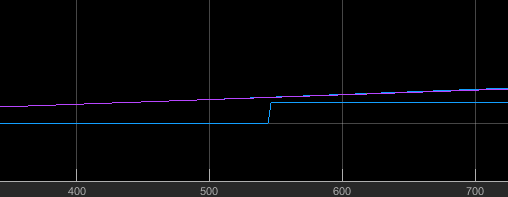
In der Darstellung ist der Simulink Aufbau des Zufallszahlen Generators zu sehen. Die Argumente 1,2,3 und 4 Stellen in selbiger Reihenfolge die Parameter A, C, M und den Startwert für X nach der Formel aus D5 dar. Nach dem Berechnen der Zufallszahl wird diese mit einem Sinus-Signal multipliziert, um auch negatives Rauschen zu erhalten. Vor dem addieren auf die generierten Radgeschwindigkeiten wird das Rauschen noch skaliert.



Hier ist das Rauschen, das auf die Radgeschwindigkeiten addiert wird, noch vor der Skalierung zu sehen.

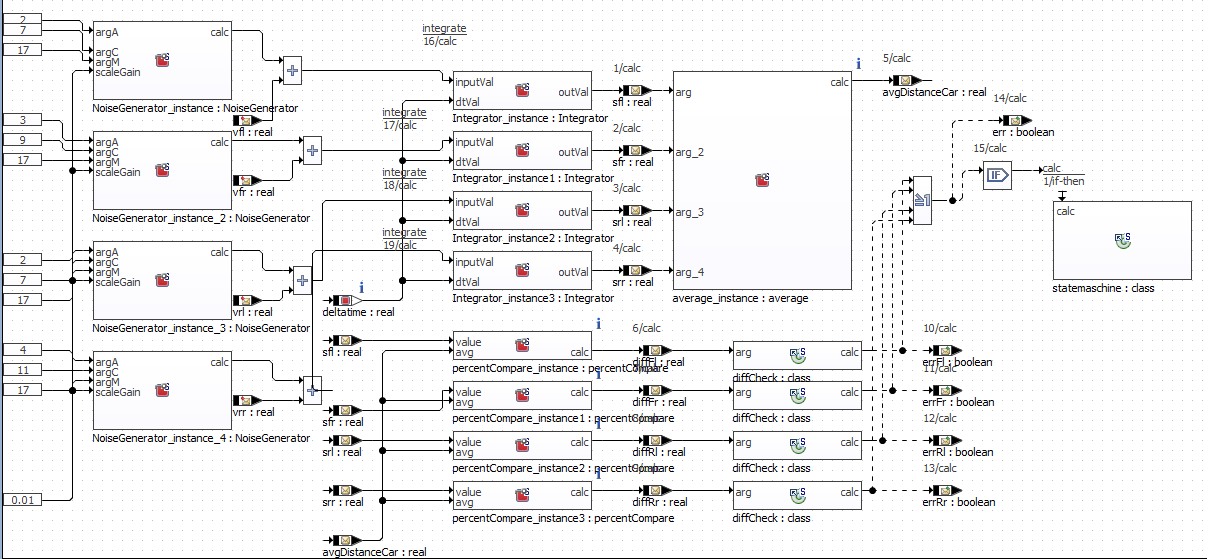
## D6 – Simulink Modell Test



Dargestellt ist der System Test mit einer Geschwindigkeit von 50 auf die das vorher gezeigt Rauschen skaliert addiert wird. Oben im Bild ist ein Störsignal zu erkennen, welches nach 500 Zeiteinheiten auf die Geschwindigkeit des ersten Rads addiert wird. Zu erwarten ist daher ein leicht verzögertes Fehlersignal nach 500 Zeiteinheiten (wie im unteren Plot am blauen Signal erkennbar). Die Verzögerung tritt hierbei auf, da die Geschwindigkeit integriert wird und daher zunächst geglättet wird.

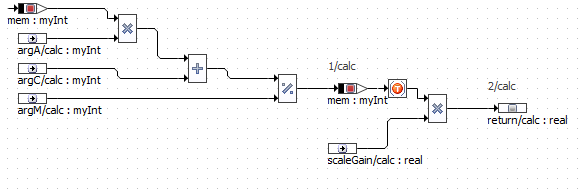
## D6 – Transfer des Reifendruckmonitors in ASCET

Auf der nächsten Seite ist das ASCET Modell des Reifendruckmonitors zu sehen. Im Folgenden werden nun die einzelnen Komponenten näher erläutert. Grob erklärt befindet sich links im Bild der Teil mit den Rausch- und Signal-Generatoren. Darauffolgend befinden sich die Integratoren zur Umrechnung bzw. Glättung der Radgeschwindigkeiten. Rechts davon wird der Durchschnitt der Strecken berechnet. Darunter befindet sich die Berechnung der Abweichung und die Auswertung der einzelnen Reifenstrecken. Ganz rechts im Bild ist die „Statemachine“, die im Fehlerfall ein „SOS“ Signal erzeugt.

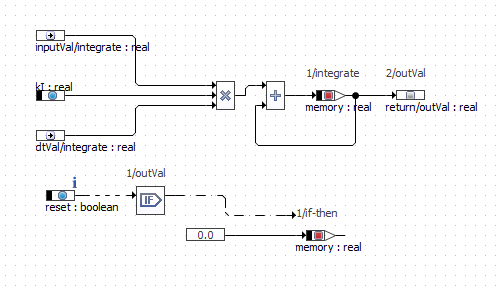


### Komponente: Rausch-Generator

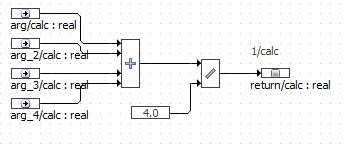
Zu sehen ist der Aufbau des Rausch-Generators. Links zu sehen sind die drei Argumente der Formel aus D5.

Die Variable „mem“ stellt das Ergebnis des vorhergegangenen Iterationsschrittes bereit. Die gewählten Argumente sind in der Gesamtansicht auf der vorherigen Seite ganz links zu sehen.

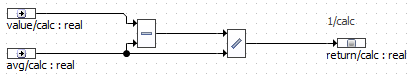
### Komponente: Integrator

Dieser Integrator wurde der ASCET Hilfe nach aufgebaut. Die Reset Funktion setzt nach Anforderung R4 den Speicher des Integrators zurück.

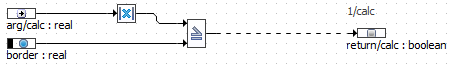
### Komponente: Durchschnitt

Die Durchschnittsberechnung funktionier über die vier Eingangsargumente indem diese addiert werden und danach durch vier geteilt wird. Zurückgegeben wird der Durchschnitt

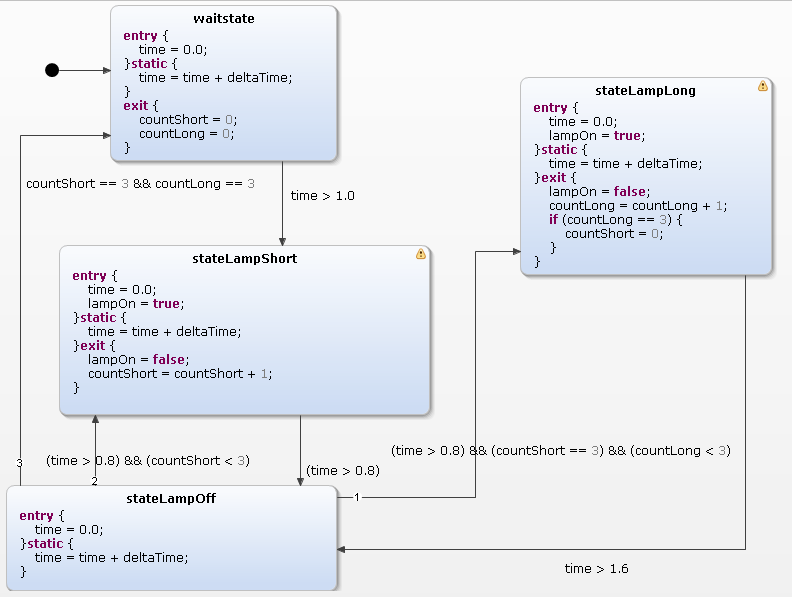
### Komponente: Berechnung der Abweichung vom Durchschnitt

Diese Komponente berechnet die Abweichung der einzelnen Radstrecken in Prozent vom Durchschnitt nach der Formel: abweichung = (wert – durchschnitt) / durchschnitt

### Komponente: Auswertung

Hier wird der Absolutwert der Abweichung in Prozent mit der definierten Grenze von 0,5% verglichen.

### Komponente: „SOS“-Ausgabe

Hier sieht man die „Statemachine“ zur Ausgabe des „SOS“-Signals. Der erste Zustand erzeugt ein Delay von einer Sekunde, da sonst die Ansteuerung der LED in der Experimentierumgebung nicht wie gewünscht funktioniert. „stateLampShort“ lässt die LED für ein kurzes Signal aufleuchten und springt dann in „stateLampOff“. Die LED leuchtet drei Mal kurz (0.8 Sekunden) mit jeweils 0.8 Sekunden Pause dazwischen. Das gleiche passiert mit dem langen Signal (1.6 Sekunden) und wieder drei Mal mit dem kurzen Signal.

Ist das „SOS“-Signal beendet geht die Statemachine wieder in den „waitState“ über und beginnt von vorne. Sobald kein Fehler mehr anliegt terminiert die Statemachine sofort. Ist die LED zu diesem Zeitpunkt an, bleibt sie auch an. Deshalb bauten wir eine Funktion ein, die die LED ausschaltet, wenn kein Druckunterschied anliegt.

D8 – Unit Tests

### Komponente: Durchschnitt

**static** **class** AverageTest {

average avg;

@Test

**public** **void** testAverageNormal() {

Assert.assertTrue(avg.calc(1.0,2.0,3.0,4.0)==2.5);

}

@Test

**public** **void** testAverageSpecial() {

Assert.assertTrue(avg.calc(1.0,1.0,1.0,1.0)==1.0);

}

@Test

**public** **void** testAverageNegative() {

Assert.assertTrue(avg.calc(-1.0,2.0,-3.0,4.0)==0.5);

}

### }

### Komponente: Auswertung

**static** **class** diffCheckTest {

@Test

**public** **void** testOFDiffCheckUpper() {

Assert.assertTrue(diffCheck.calc(0.005));

}

@Test

**public** **void** testOFDiffCheckInBoundPositive() {

Assert.assertFalse(diffCheck.calc(0.0005));

}

@Test

**public** **void** testOFDiffCheckInBoundNegative() {

Assert.assertFalse(diffCheck.calc(-0.0005));

}

@Test

**public** **void** testOFDiffCheckLower() {

Assert.assertTrue(diffCheck.calc(-0.005));

}

}

### Komponente: Integrator

**static** **class** IntegratorTest {

Integrator integ1;

Integrator integ2;

Integrator integ3;

Integrator integ4;

@Test

**public** **void** testIntegratorNormal() {

integ1.integrate(50.0, 10.0);

Assert.assertTrue(integ1.outVal() == 500.0);

}

@Test

**public** **void** testIntegratorNegative() {

integ2.integrate(-50.0, 10.0);

Assert.assertTrue(integ2.outVal() == -500.0);

}

@Test

**public** **void** testIntegratorZero() {

integ3.integrate(0.0, 10.0);

Assert.assertTrue(integ3.outVal() == 0.0);

}

@Test

**public** **void** testIntegratorFlow() {

integ4.integrate(2.0, 10.0);

integ4.integrate(5.0, 10.0);

integ4.integrate(1.0, 10.0);

integ4.integrate(3.0, 10.0);

Assert.assertTrue(integ4.outVal() == 110.0);

}

}

### Komponente: Rauschgenerator

**static** **class** noiseGeneratorTest {

NoiseGenerator ng1;

NoiseGenerator ng2;

NoiseGenerator ng3;

NoiseGenerator ng4;

@Test

**public** **void** testOfNoiseNormal() {

**real** x = ng1.calc(2, 7, 17, 0.01);

Assert.assertTrue((x < 0.1) || (x > 0.08));

}

@Test

**public** **void** testNoiseNegativeA() {

**real** x = ng2.calc(-2, 7, 17, 0.01);

Assert.assertTrue((x < 0.06) || (x > 0.04));

}

@Test

**public** **void** testNoiseNegativeC() {

**real** x = ng2.calc(2, -6, 17, 0.01);

Assert.assertTrue((x < 0.05) || (x > 0.03));

}

@Test

**public** **void** testNoiseZeroA() {

**real** x = ng3.calc(0, 7, 17, 0.01);

Assert.assertTrue((x < 0.08) || (x > 0.06));

}

@Test

**public** **void** testNoiseZeroC() {

**real** x = ng3.calc(2, 0, 17, 0.01);

Assert.assertTrue((x < 0.03) || (x > 0.01));

}

@Test

**public** **void** testNoiseFlow() {

**real** x;

ng4.calc(2, 7, 17, 0.01);

ng4.calc(2, 7, 17, 0.01);

ng4.calc(2, 7, 17, 0.01);

x = ng4.calc(2, 7, 17, 0.01);

Assert.assertTrue(x == 0.02);

}

}

### Komponente: Abweichung

**static** **class** percentCompareTest {

percentCompare pc;

@Test

**public** **void** testPerCompNormal() {

**real** x = pc.calc(9.5, 10.0);

Assert.assertTrue(x == -0.05);

}

@Test

**public** **void** testPerCompNegativeVal() {

**real** x = pc.calc(-5.1, 10.0);

Assert.assertTrue(x == -1.51);

}

@Test

**public** **void** testPerCompNegativeAvg() {

**real** x = pc.calc(9.5, -10.0);

Assert.assertTrue(x == -1.95);

}

@Test

**public** **void** testPerCompZeroVal() {

**real** x = pc.calc(0.0, 10.0);

Assert.assertTrue(x == -1.0);

}

@Test

**public** **void** testNoiseZeroAvg() {

**real** x = pc.calc(9.4, 0.0);

Assert.assertTrue(x == 0.0);

}

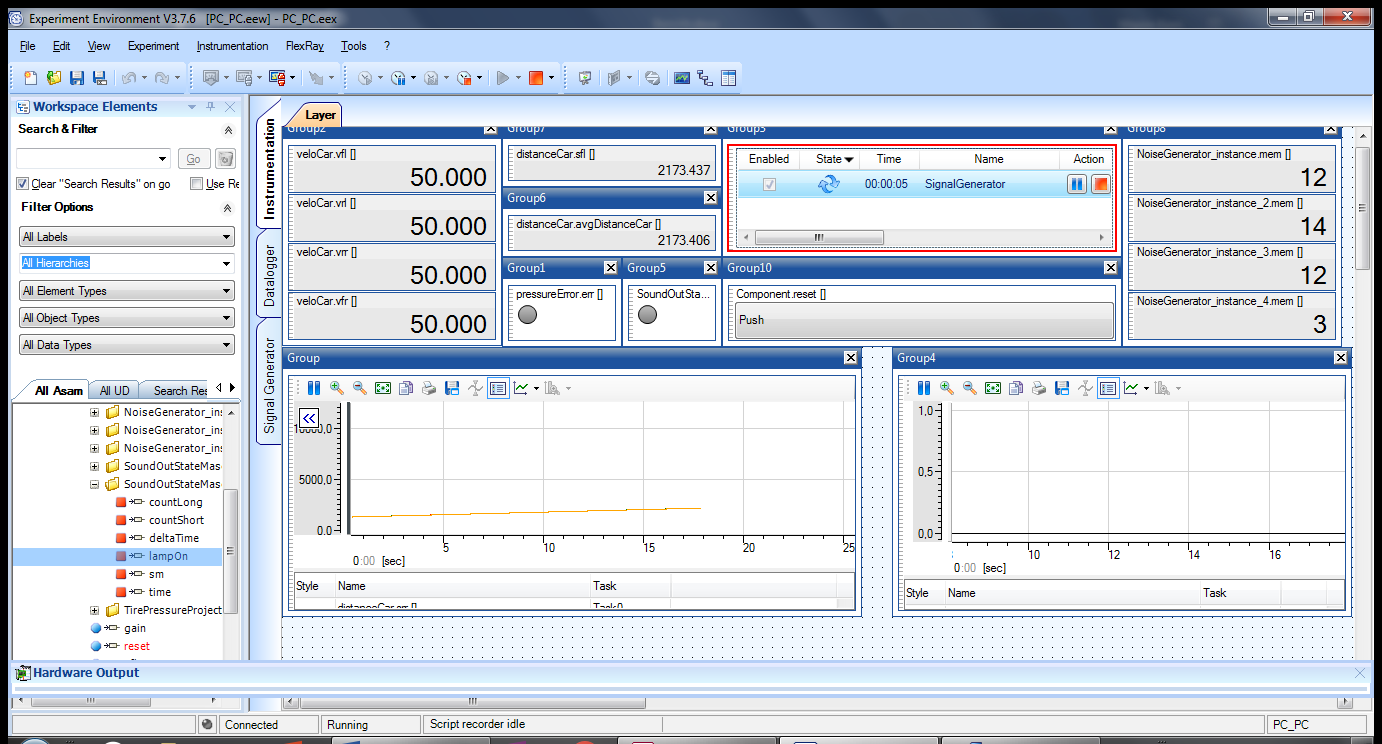
}

D9 – Warnfunktion

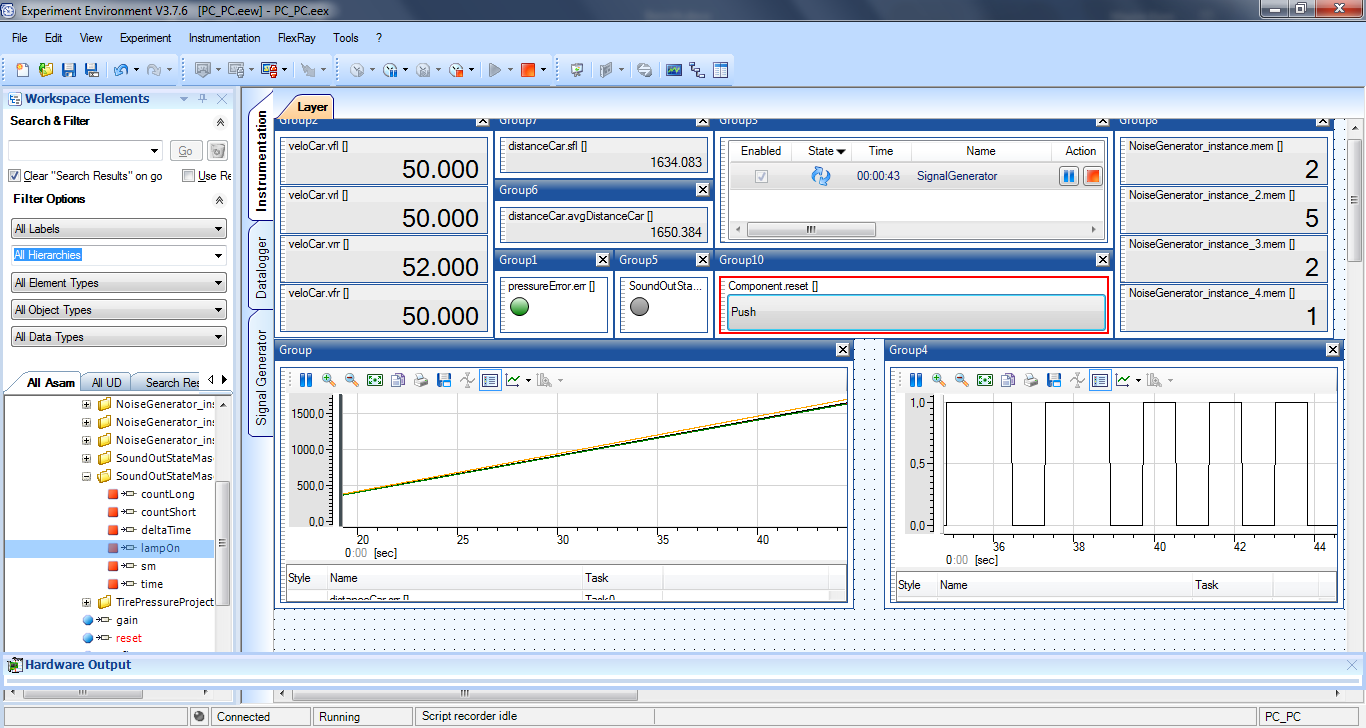
siehe D7 – ASCET-Modell, Komponente: “SOS“-Ausgabe

D10 – Unit Tests Zufallszahlengenerator

Siehe D8 – Unit Tests, Komponente: Rauschgenerator

D11 – Systemtest

In der Abbildung sieht man die Experimentierumgebung zum Projekt. Links oben sind die Geschwindigkeiten der einzelnen vier Räderdargestellt. Daneben stehen beispielhaft die zurückgelegte Strecke eines Rades und die durchschnittliche Strecke aller vier Räder. Dies ist auch im Diagramm links unten dargestellt. In der Mitte sind zwei LEDs zum Anzeigen von Fehlern. Die linke LED zeigt einen Fehler an, die rechte sendet das „SOS“-Signal, dies ist auch im Diagramm rechts unten dargestellt. Rechts oben werden die vier Rauschsignale visualisiert und daneben der Signalgenerator, der die Geschwindigkeiten erzeugt und ein Button, um alles zurückzusetzen.



In diesem Screenshot sieht man die Experimentierumgebung im Fehlerfall. Das Rad vorne rechts fährt nun schneller als die anderen (🡪 Druckverlust). Im Diagramm links unten erkennt man, wie sich die Linien trennen, die Strecke des fehlerbehafteten Reifens, die Strecke der übrigen Reifen und der Durchschnitt. Die LED zum Anzeigen eines Fehlers leuchtet und die „SOS“-LED blinkt (siehe Diagramm rechts unten).

D12 – Reset-Funktion

Um ein Reset umzusetzen, muss man den Speicher der Integratoren auf null zurücksetzen (siehe D7 – Komponente: Integrator). Der Durchschnitt wird zu jedem Zeitschritt neu berechnet und wird dadurch automatisch zurückgesetzt. Die Statemachine zur Ausgabe des „SOS“-Signals terminiert selbstständig, wenn kein Fehler mehr anliegt.

Um das Monitoring neu beginnen zu können, muss man nach dem Reset weit genug fahren (einige Meter) um das Rauschen der Messung zu kompensieren.