



---

# STOP-ROLL-GO

---

Tom Schubert (4328112) \ Andrik Seeger (1319658) \ Thomas Tran (1060449)



12. APRIL 2022

## Inhaltsverzeichnis

Designentscheidungen .....	2
Anpassung an myCar.....	2
Proximity .....	2
Verhalten bei Gelb.....	3
Maximale Entschleunigung .....	3
Gliederung von Klassen.....	3
Messages.....	3
Namenskonvention.....	3
D1: Aufwandsschätzung aller Aufgaben .....	3
D2: Machbarkeit und Limitierungen .....	3
D3: Erstellung des Ampelmoduls .....	6
Unittests .....	9
D4: Automatisierter Fahrer und Ampelblitz.....	12
Ampelblitz .....	12
Funktionalität .....	12
Automatisierter Fahrer .....	12
Geschwindigkeitsregelung .....	12
Kontrollrechnung der Fahrzeug Entschleunigung .....	13
Entscheidungsfindung.....	14
Unittests .....	16
D5: Unitests .....	19
D6: Experimentierumgebung .....	19
D7: Gelbzeiten .....	24
D8: Konstante Grünphasen.....	25
D9: Verbesserung mittels V2X oder Verwendung von Ampeltimings / -positionen.....	27
D12: Auswirkung von Reaktion und anderen Verzögerungen.....	30
D13: Reflexion.....	33
Planung:.....	33
Modellierung: .....	34
Anforderungen: .....	34
Funktionen: .....	34
Lösungsansatz: .....	34
Tests: .....	35
Persönliches Empfinden Grafische Programmierung mit ASCET: .....	35
Quellen .....	36

## Designentscheidungen

### Anpassung an myCar

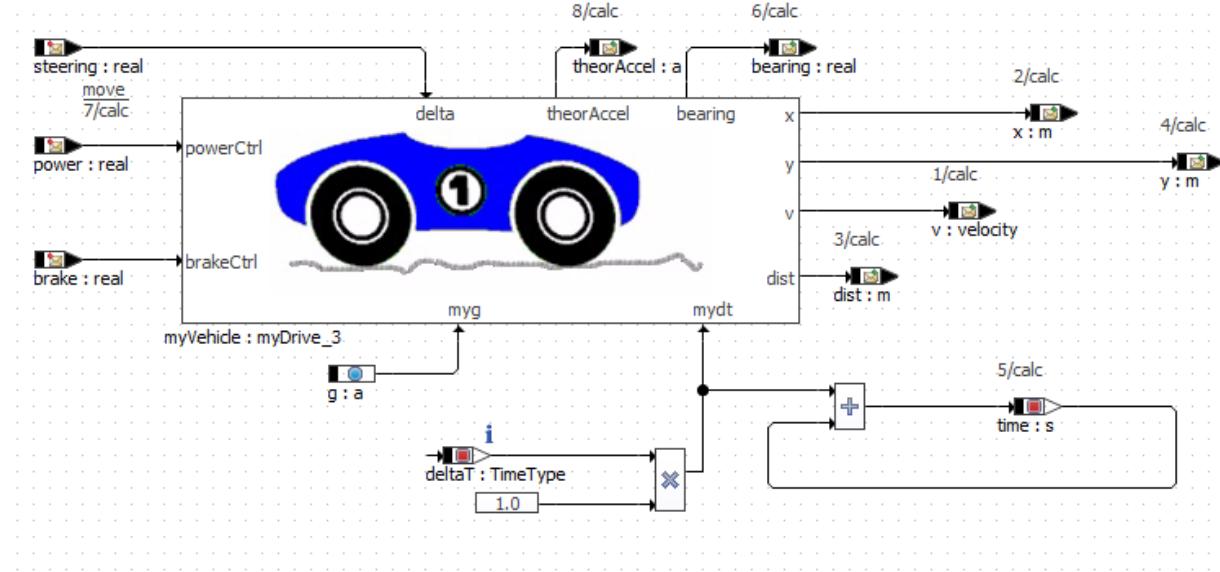


Abbildung 1 myCar.bd

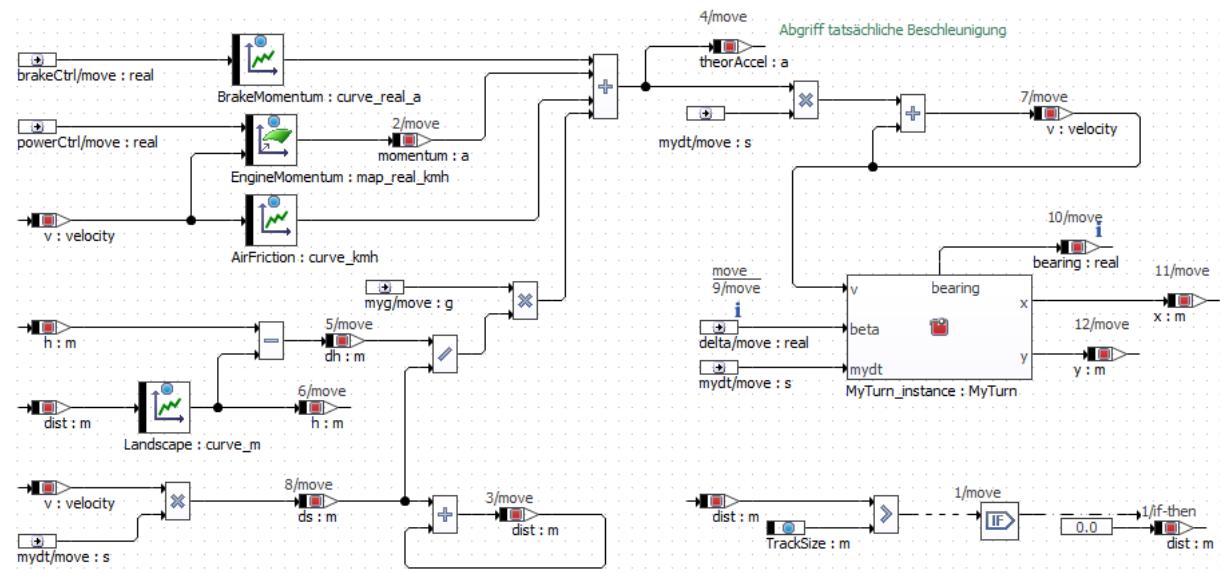


Abbildung 2 myDrive.bd

1. myCar wurde um  $\delta t$  ergänzt, sodass Klassen, wie Stopwatch oder Timer, aus der SystemLib genutzt werden können (siehe Abbildung 1).
2. myCar gibt jetzt auch dist als Message aus, sodass keine weitere Modulo 1000 Rechnung mit x mehr nötig ist (siehe Abbildung 1).
3. myCar gibt auch die aktuelle Beschleunigung a raus, um die maximale Entschleunigung zu überprüfen (siehe Abbildung 2).

## Proximity

Proximity ist die Distanz vom Auto zur nächsten Ampel und ist immer positiv (inklusive null). Nach der dritten Ampel wird die restliche Streckendistanz und die Distanz zur Ampel eins

addiert. Ist Proximity null, so ist das Auto an der Haltelinie. Proximity funktioniert nur, wenn die Ampel  $\leq 100$  m zum Auto ist. Sonst ist der Wert 99999.0 und nicht sichtbar für den Fahrer.

## **Verhalten bei Gelb**

Falls es möglich ist, innerhalb der restlichen Strecke zur Ampel rechtzeitig zum Stehen zu kommen, wird gebremst. Sonst wird mit 50 km/h mit Gelb über die Ampel gefahren. Grund dafür sind Gelbblitzer, welche blitzen, wenn ein Auto über Gelb fährt, obwohl es hätte bremsen können.

## **Maximale Entschleunigung**

Es wurden drei Kontrollen für die Einhaltung der maximalen Entschleunigung implementiert, um eine erhöhte Sicherheit zu erreichen.

## **Gliederung von Klassen**

Die erstellten Klassen wurden zuerst nach den Anforderungen D2 und D3 aufgeteilt. Und darauf nach Funktion unterteilt. So gibt es eine Klasse für das Flashlight, für das CruiseSpeedControl, für den automatisierten Fahrer(DrivingDecision) und eine Klasse für das Überprüfen der maximalen Entschleunigung. Die Ampelklasse wurde wiederum in zwei Teile unterteilt. Die TrafficLight-Klasse regelt die Ampelphasen und die vom Fahrer wahrgenommene Distanz zur Ampel. Die TrafficLightPosition-Klasse implementiert die Positionen der Ampeln und die wahre Distanz zum Fahrzeug. Die Ampelklasse wurde aufgeteilt, um schnell die Anzahl und Position der Ampeln ändern zu können, ohne die generelle Ampelklasse zu ändern. Genauso kann man die Timings der Ampeln ändern, ohne die Position zu ändern. Allgemein wurde auf funktionale Trennung geachtet. Die Fahrerkasse ruft die CruiseSpeedControl Klasse und die Ampelklasse die Ampelpositionsklasse auf. Die Klassen wurden in zwei Packages aufgeteilt, eins für alle Klassen, welche mit den Ampeln zusammenhängen und eine, welche mit dem automatisierten Fahrer im Zusammenhang sind. Es existiert noch ein Package UnitTests für alle Unitests der Implementierungsklassen. Es wurden zum Testen, falls notwendig aufgrund von Messages oder deltaT, extra Testklassen, erkennbar bei Namensendung mit [...]TestFunction.[bd / esdl], erstellt. Diese Testklassen befinden sich auch entweder im Fahrer oder Ampel Package.

## **Messages**

Es wurden zusätzliche Ampelmessages eingeführt, welche unter dem Package resources in der Datei TrafficLightMessages.esdl zu finden sind. Es handeln sich dabei um Signale, welche vom Fahrer wahrgenommen werden können. Die Signale entsprechen den Lichtern einer Ampel, dem Flashlight und zusätzlich die theoretische Beschleunigung des Autos. Also insgesamt fünf Signale (rot, gelb, grün und Flashlight) vom Typ boolean und die Beschleunigung in  $\frac{m}{s^2}$

## **Namenskonvention**

Es werden ausdrucksstarke Namen verwendet. Die angewendete Konvention ist CamelCase, wobei Funktions- und Variablennamen klein beginnen und Klassennamen groß. Die Konvention wurde auf alle selbsterstellten Klassen angewendet.

## **D1: Aufwandsschätzung aller Aufgaben**

Siehe Excel-Sheet.

## **D2: Machbarkeit und Limitierungen**

Zur Demonstration der Limitierung der Aufgaben werden folgende Anforderungen theoretisch auf ihre Machbarkeit überprüft:

1. Kann in jedem Fall garantiert werden, dass das Fahrzeug nicht über eine rote Ampel fahren muss?

**Annahmen:**

- Maximale Entschleunigung des Fahrzeugs:  $2,5 \frac{m}{s^2}$
- Maximale Bewegungsgeschwindigkeit  $50 \frac{km}{h}$

Vorgehen:

Max-Cruise speed:

$$50 \frac{km}{h} = 13,88889 \frac{m}{s}$$

Bremszeit bei  $v=13,889 \frac{m}{s}$ :

$$t = \frac{v}{a} = \frac{13,889 \frac{m}{s}}{2,5 \frac{m}{s^2}} = 5,556s$$

Bremsweg bis Stillstand:

$$s = 0,5 * a * t^2 = 0,5 * 2,5 \frac{m}{s^2} * 5,556s^2 = 38,58 m$$

Somit lässt sich festhalten, dass bei einem Abstand  $> 38,58$  m zur Ampel der Fahrer in jedem Fall noch in der Lage ist, das Fahrzeug zum Stehen zu bringen.

Wenn der Abstand zur Ampel jedoch  $< 38,58$  m beträgt, muss der automatisierte Fahrer auf jeden Fall weiterfahren. Im Grenzwert gilt somit, dass in jedem Fall, das Auto mindestens 38,58m vor der Umschaltung auf Rot zurücklegen muss. Für die Machbarkeit dieser Aussage werden folgende Annahmen getroffen:

- Das Fahrzeug fährt auf gerader Fläche
- Der automatisierte Fahrer beschleunigt immer mit einer Gaspedalstellung von 100 %.
- Der Abstand zwischen den Ampeln beträgt minimal 200 m.
- Die Beschleunigung des Fahrzeugs berechnet sich aus dem geschwindigkeitsabhängigen Motormoment (vgl. Abbildung 3) und dem geschwindigkeitsabhängigen Luftwiderstand. Wie der Tabelle in Abbildung 3 zu entnehmen, beträgt bei einer Gaspedalstellung von 100% und einer Geschwindigkeit von bereits  $60 \frac{km}{h}$  die Beschleunigung des Fahrzeugs  $3,8 \frac{m}{s^2}$ . Zieht man davon noch den Luftwiderstand bei  $60 \frac{km}{h}$ , also  $0,2 \frac{m}{s^2}$  ab, ergibt sich pessimistisch gerechnet eine minimale Beschleunigung von  $3,6 \frac{m}{s^2}$ . Mit dieser Beschleunigung kann das Fahrzeug innerhalb  $0,5 * \left(\frac{13,89^2}{3,6}\right) = 26,8$  m auf die maximale Geschwindigkeit von  $50 \frac{km}{h}$  beschleunigen. Bei einem Ampelabstand von 200 m kann somit garantiert davon ausgegangen werden, dass das Fahrzeug sich mit  $50 \frac{km}{h}$  auf die Ampel zubewegt.

- Der automatisierte Fahrer hat eine Reaktionszeit von 0s und die Kommunikation zwischen Ampel und Fahrzeug ist vernachlässigbar.

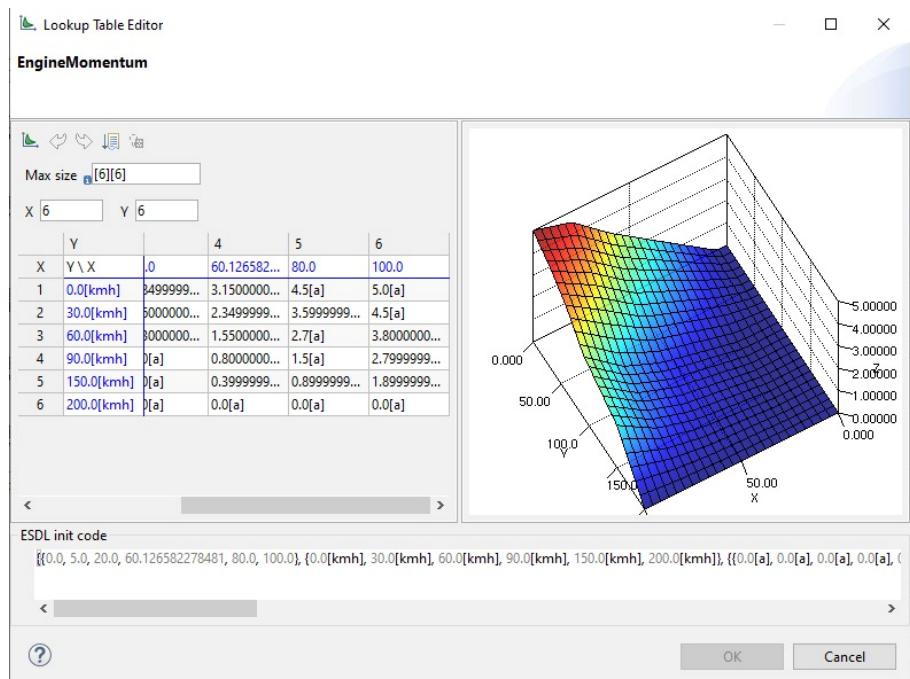


Abbildung 3 Kennfeld vom Motormoment

Betrachtet man die eben aufgezählten Annahmen kann das Fahrzeug den Bremsweg von 38,58 m mit einer Geschwindigkeit von 50  $\frac{km}{h}$  in einer minimalen Zeit von:

$$\frac{\text{Bremsweg}}{\text{Cruise speed}} = \frac{38,58 \text{ m}}{13,889 \frac{\text{m}}{\text{s}^2}} = 2,778 \text{ s}$$

Überwinden.

Wichtig: Es soll verhindert werden, dass das Fahrzeug über Rot fährt:

Der Grenzfall, dass die Ampel auf Gelb umschaltet und der Fahrer weder anhalten noch weiterfahren kann, ohne über Rot zu fahren, muss somit verhindert werden. Dafür gilt Folgendes:

R1 definiert eine Gelbzeit von 3s und R2 definiert eine maximale Sichtweite auf die Ampel von 100 m.

Da die 100 m Sichtweite > 38,58 m Bremsweg sind kann der Fahrer unabhängig vom Ampelzustand reagieren.

Die Dauer den Bremsweg bei 50  $\frac{km}{h}$  zu überwinden beträgt 2,778s, was kleiner als die Ampelgelbzeit von 3s ist. Somit kann der Fahrer in allen drei folgenden Fällen reagieren:

1. Ampel wird gelb und der Abstand zur Ampel beträgt  $proximity \leq 38,58 \text{ m} \rightarrow$  Fahrzeug fährt weiter
2. Ampel wird gelb und der Abstand zur Ampel beträgt  $100 \text{ m} \geq proximity \geq 38,58 \text{ m} \rightarrow$  Fahrzeug hält an
3. Ampel wird gelb und der Abstand zur Ampel beträgt  $proximity > 100 \text{ m} \rightarrow$  Fahrzeug reagiert nicht

## D3: Erstellung des Ampelmoduls

Für die Implementierung der Ampelfunktionalität wird ein separates Modul „TrafficLight“ erstellt.

Das Block Diagramm dieses Moduls ist in den Abbildungen 4 bis 8 abgebildet

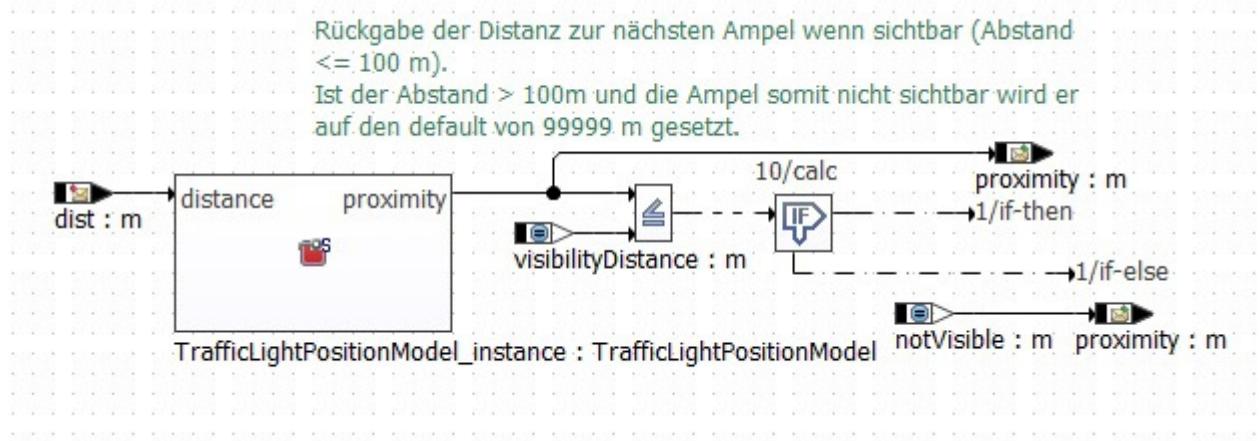


Abbildung 4 Proximity in TrafficLight.bd

Hierbei wird über eine Message die Position auf dem Rundtrack ( $0 \leq dist \leq 1000$ ) in das Modul gesendet. Die Position des Fahrzeugs dient als Eingabeparameter für die Klasse „TrafficLightPositionModel“. Diese gibt als Ausgabeparameter den Abstand zur nächsten Ampel zurück.

Falls die nächste Ampel nicht in Sichtweite steht ( $prox > 100$  m), wird als Proximity der Wert 99999 gesendet, ansonsten der reale Abstand zur Ampel. Für visibilityDistance wird hierbei eine Konstante verwendet, weil die Sichtweite in der Aufgabenstellung als konstant angenommen werden soll.

Die Implementierung der TrafficLight-Klasse übernimmt folgende Aufgaben:

Beim Aufruf der Klasse werden zuerst alle internen Ampellichtzustände auf den Wert False zurückgesetzt. Somit wird verhindert, dass der zuvor berechnete Zustand ein unvorhersehbares Verhalten erzeugt.(vgl. Abbildung 5)

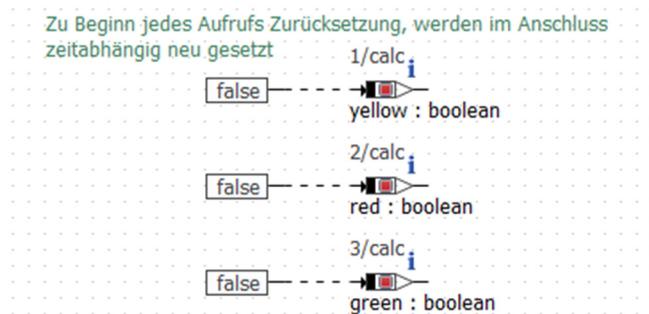


Abbildung 5 Zurücksetzung der Ampellichtzustände

In den nächsten Schritten wird basierend auf der Zeit der aktuelle Ampelzustand errechnet, siehe Abbildung 6. Hierfür wird eine StopWatch-Klasse aus dem Systemlib Package importiert und beim ersten Aufruf instanziert und auf 0s gestellt. Bei jedem Aufruf erhöht sich der Output der StopWatch um den Wert dt (vergangene Zeit seit letztem Aufruf). Solange die Zeit auf der StopWatch unter 3 Sekunden liegt, wird nur die private Variable „yellow“ auf True gesetzt.

Sobald die Zeit über den Wert von 3s springt, wird der nächste Ampelzustand (Rot) angezeigt. Hierfür wird die private Variable red auf True gesetzt. Mit dieser Logik werden alle Ampelzustände implementiert. Da beim Aufruf immer zuerst alle internen Zustände zurückgesetzt werden, müssen bei der Implementierung der Zustände in der if-else Verschachtelung nur Ampelzustände gesetzt und nicht zurückgesetzt werden.

Erreicht die verstrichene Zeit t den Wert von 60s und ist somit größer als die konstante greenTime wird die Stoppuhr zurückgesetzt.

Für die Implementierung der Ampelzeiten werden die Konstanten yellowTime, redTime, redYellowTime, GreenTime verwendet, um die Anforderung R1 exakt zu erfüllen und keine zufälligen Veränderungen zu riskieren. Die Konstanten sind vom Typ real, weil sie somit direkt mit dem Ausgangswert der Stopp-Watch vergleichbar sind. Die Konstanten halten die folgenden Werte:

yellowTime: 3

redTime: 42

redYellowTime: 44

GreenTime: 60

Der interne Zustand der Ampeln wird in den drei Variablen yellow, red und green gehalten.

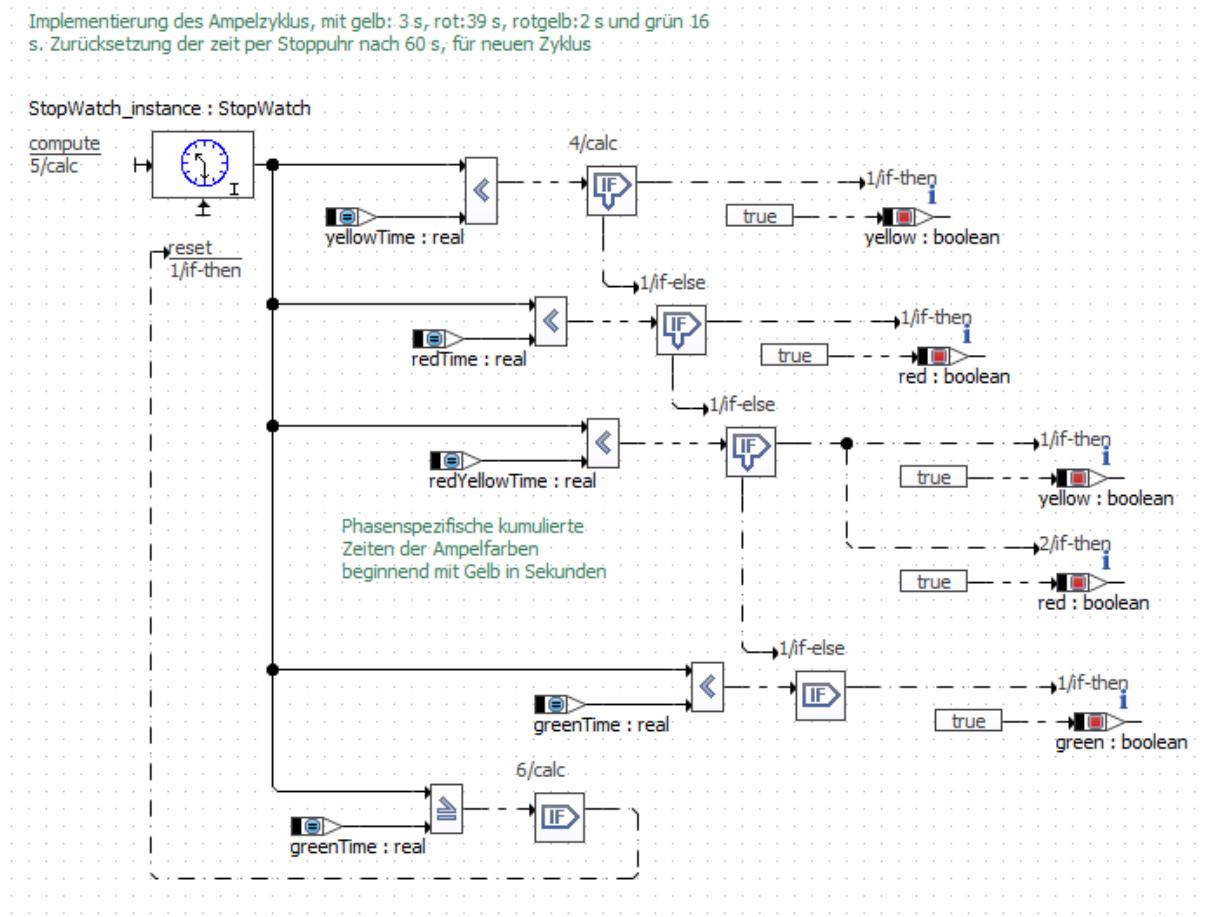


Abbildung 6 Implementierung des Ampelzyklus

In den letzten Schritten der TrafficLight-Funktionalität wird der momentane interne Ampelzustand über Messages nach außen gesendet (Abbildung 7).

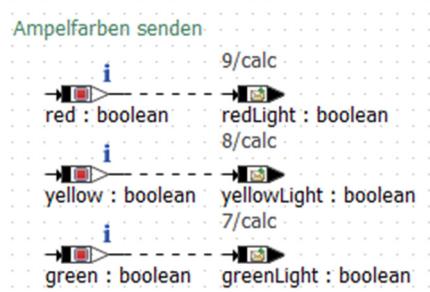


Abbildung 7 Ampelfarben senden

Abbildung 8 zeigt die Implementierung der TrafficLightPositionModel-Klasse, die bereits weiter oben erwähnt wurde.

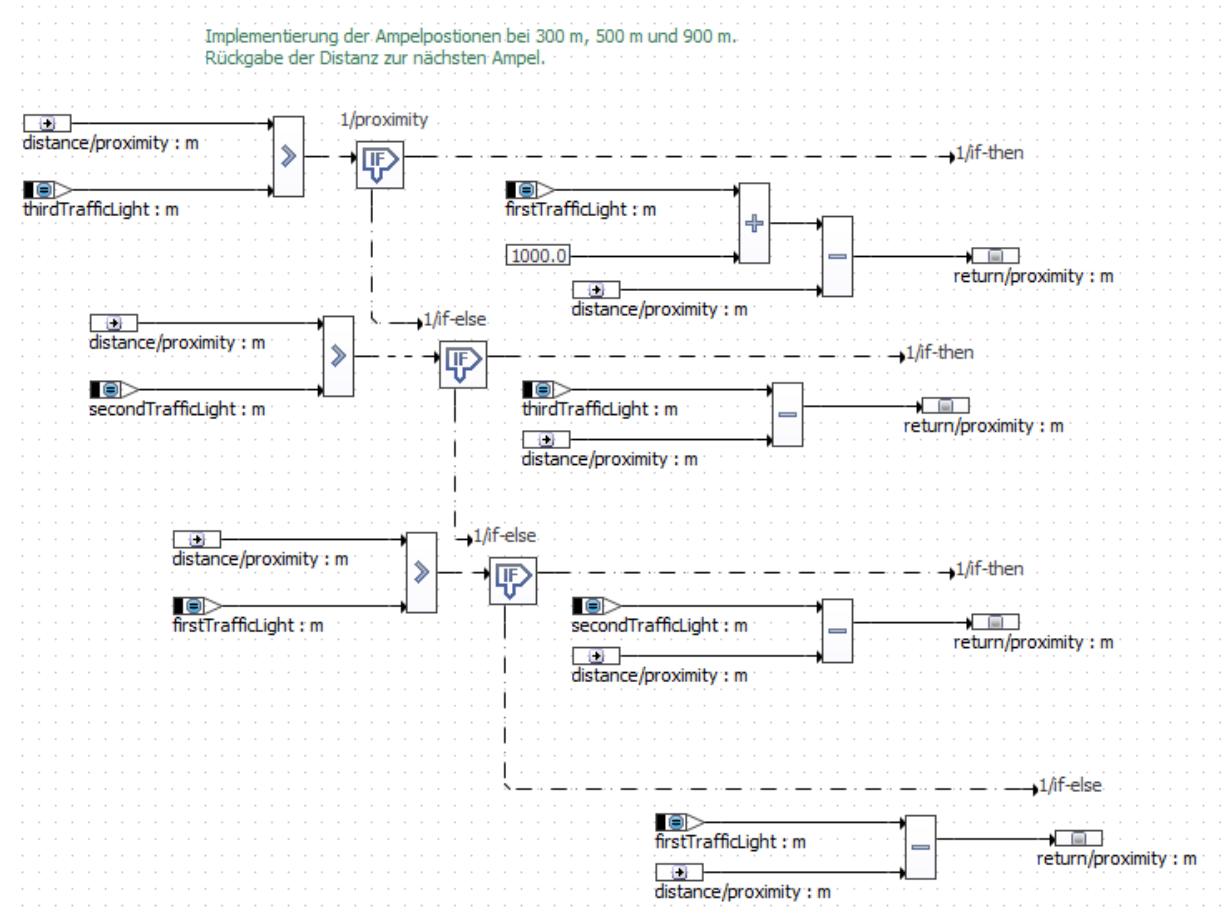


Abbildung 8 TrafficLightPosition.bd

Das Blockschaltbild zeigt die Implementierung der Abstandsberechnung zur nächsten Ampel. Die Positionen der Ampel (300m,500m,900m) sind mittels Konstanten fest im Blockschaltbild definiert. Über eine if else Abfrage wird die Position des Fahrzeugs mit der jeweiligen Position der Ampel verglichen. Ist die Position des Fahrzeugs hinter der zweiten Ampel kann davon

ausgegangen werden, dass sich das Fahrzeug mit einem Abstand x vor der dritten Ampel befindet. Dieser Abstand x wird dann berechnet und über eine Rückgabe an das „TrafficLight“-Modul zurückgegeben.

Befindet sich das Fahrzeug vor der zweiten Ampel, wird überprüft ob es sich auch vor der ersten Ampel befindet. Basierend auf dem Resultat der Entscheidung wird entweder der Abstand zu Ampel 2 oder der Abstand zu Ampel 3 berechnet und zurückgegeben. Da alle drei Ampeln den gleichen Ampelzyklus durchlaufen reicht lediglich die Rückgabe des Abstands zur nächsten Ampel und es ist irrelevant vor welcher Ampel sich das Fahrzeug befindet.

## Unitests

Die Implementierung der Unit-Tests ist in den Abbildungen 9 bis 11 gezeigt. Aufgrund der vorhandenen Messages und dem Gebrauch von DeltaT in der Stopwatch, musste eine extra Testklasse implementiert werden, welche die Messages durch Argumente und DeltaT durch konstante Werte ersetzt. Hier wird je nach Ampelanzeige ein Zustand zurückgegeben, welcher an den Sprungzeiten überprüft wird. Es wurde eine Coverage von 100 % erreicht und alle Zustände wurden getestet (Abbildung 9).

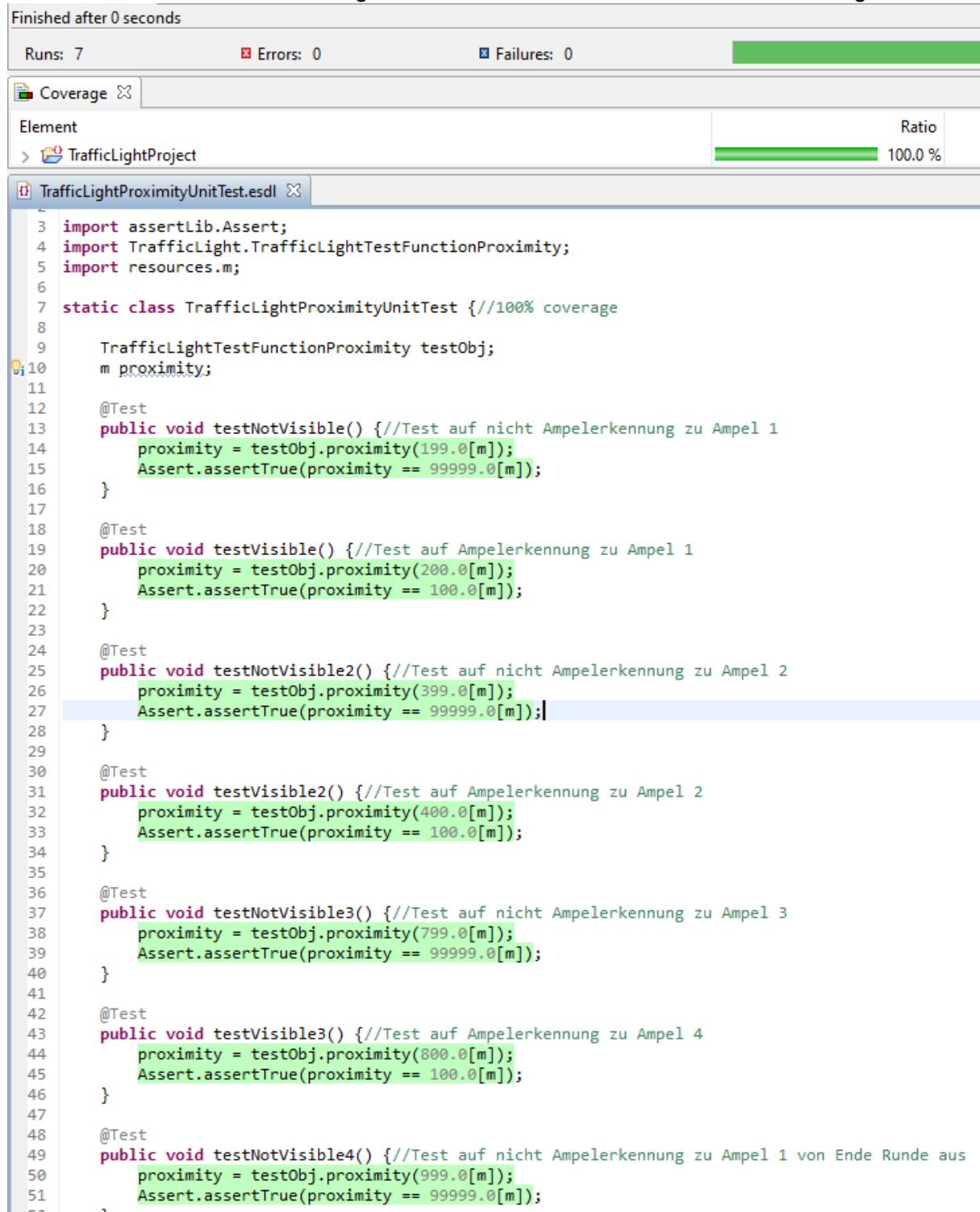
```

1 package UnitTests;
2
3 import assertLib.Assert;
4 import TrafficLight.TrafficLightTestFunction;
5
6 static class TrafficLightUnitTest { //100% coverage
7
8     TrafficLightTestFunction testObj;
9     integer stateLight;
10    //Ampelzeiten je Zyklus
11    const real yellowTime = 3.0;
12    const real redTime = 42.0;
13    const real redYellowTime = 44.0;
14    const real greenTime = 60.0;
15
16    @Test
17    public void testForYellow() { //Test auf gelbe Ampel bei 0s
18        stateLight = testObj.calc(0.0);
19        Assert.assertTrue(stateLight == 1);
20    }
21
22    @Test
23    public void testForRed() { //Test auf rote Ampel bei 3s
24        stateLight = testObj.calc(3.0);
25        Assert.assertTrue(stateLight == 2);
26    }
27
28    @Test
29    public void testForYellowRed() { //Test auf gelbrote Ampel bei 42.0s
30        stateLight = testObj.calc(42.0);
31        Assert.assertTrue(stateLight == 3);
32    }
33
34    @Test
35    public void testForGreen() { //Test auf grüne Ampel bei 44s
36        stateLight = testObj.calc(44.0);
37        Assert.assertTrue(stateLight == 4);
38    }
39
40    @Test
41    public void testForReset() { //Test für Reset bei 60s
42        stateLight = testObj.calc(60.0);
43        Assert.assertTrue(stateLight == 5);
44}

```

Abbildung 9 TrafficLightUnitTest.esdl

Auch im folgenden Unitest (Abbildung 10) musste, aufgrund der Messages, eine extra Testklasse implementiert werden. Bei dem Test wird die Proximity zur Ampel getestet, um den Test vom normalen Ampelbetrieb zu trennen wurde ein extra Test angelegt. Es wird für jede Ampel auf Erkennung unter 100 m getestet und auf nicht Erkennung bei einem Abstand von > 100 m. Es wird auch extra dafür getestet, wenn die Strecke wieder von vorne beginnt.



The screenshot shows a Java IDE interface with a coverage analysis for a file named `TrafficLightProximityUnitTest.esdl`. The coverage report indicates 100.0% coverage across 7 runs, with 0 errors and 0 failures. The code itself contains several test methods for proximity detection at different distances from traffic lights.

```

3 import assertLib.Assert;
4 import TrafficLight.TrafficLightTestFunctionProximity;
5 import resources.m;
6
7 static class TrafficLightProximityUnitTest { //100% coverage
8
9     TrafficLightTestFunctionProximity testObj;
10    m proximity;
11
12    @Test
13    public void testNotVisible() {//Test auf nicht Ampelerkennung zu Ampel 1
14        proximity = testObj.proximity(199.0[m]);
15        Assert.assertTrue(proximity == 99999.0[m]);
16    }
17
18    @Test
19    public void testVisible() {//Test auf Ampelerkennung zu Ampel 1
20        proximity = testObj.proximity(200.0[m]);
21        Assert.assertTrue(proximity == 100.0[m]);
22    }
23
24    @Test
25    public void testNotVisible2() {//Test auf nicht Ampelerkennung zu Ampel 2
26        proximity = testObj.proximity(399.0[m]);
27        Assert.assertTrue(proximity == 99999.0[m]);
28    }
29
30    @Test
31    public void testVisible2() {//Test auf Ampelerkennung zu Ampel 2
32        proximity = testObj.proximity(400.0[m]);
33        Assert.assertTrue(proximity == 100.0[m]);
34    }
35
36    @Test
37    public void testNotVisible3() {//Test auf nicht Ampelerkennung zu Ampel 3
38        proximity = testObj.proximity(799.0[m]);
39        Assert.assertTrue(proximity == 99999.0[m]);
40    }
41
42    @Test
43    public void testVisible3() {//Test auf Ampelerkennung zu Ampel 4
44        proximity = testObj.proximity(800.0[m]);
45        Assert.assertTrue(proximity == 100.0[m]);
46    }
47
48    @Test
49    public void testNotVisible4() {//Test auf nicht Ampelerkennung zu Ampel 1 von Ende Runde aus
50        proximity = testObj.proximity(999.0[m]);
51        Assert.assertTrue(proximity == 99999.0[m]);
52    }

```

Abbildung 10 `TrafficLightProximityUnitTest.esdl`

The screenshot shows a software interface for unit testing. At the top, there's a header bar with the text "C/C++ Unit" and a progress bar indicating "Runs: 4", "Errors: 0", and "Failures: 0". Below this is a "Coverage" tab with a tree view showing a project named "TrafficLightProject" which has a 100.0% coverage ratio. The main area displays the source code for "TrafficLightPositionUnitTest.esdl". The code is annotated with line numbers and contains several test methods using annotations like @Test and public void. The code is as follows:

```

1 package UnitTests;
2
3 import assertLib.Assert;
4 import TrafficLight.TrafficLightPositionModel;
5 import resources.m;
6
7 static class TrafficLightPositionUnitTest { //100% coverage
8
9     TrafficLightPositionModel testObj;
10    m proximity;
11
12    @Test//Test auf korrekte Distanz zu Ampel 3
13    public void testForLight3() {
14        proximity = testObj.proximity(800.0[m]);
15        Assert.assertTrue(proximity==100.0[m]);
16    }
17
18    @Test//Test auf korrekte Distanz zu Ampel 2
19    public void testForLight2() {
20        proximity = testObj.proximity(450.0[m]);
21        Assert.assertTrue(proximity==50.0[m]);
22    }
23
24    @Test//Test auf korrekte Distanz zu Ampel 1
25    public void testForLight1() {
26        proximity = testObj.proximity(297.0[m]);
27        Assert.assertTrue(proximity==3.0[m]);
28    }
29
30    @Test//Test auf korrekte Distanz zu Ampel 1 nach Ampel 3 bevor die Strecke zurückgesetzt wird
31    public void testForLight1from3() {
32        proximity = testObj.proximity(999.0[m]);
33        Assert.assertTrue(proximity==301.0[m]);
34    }
35 }

```

Abbildung 11 TrafficLightPositionUnitTest.esdl

Für das TrafficLightPositionModel war keine extra Testklasse notwendig, da sie keine Messages besitzt. Es wurden die Distanzen zu jeder Ampel überprüft. Auch bevor die Strecke zurückgesetzt wird, siehe Abbildung 11.

## D4: Automatisierter Fahrer und Ampelblitz

### Ampelblitz

#### Funktionalität

Ein Ampelblitz wird ausgelöst, wenn die proximity steigt, also eine Ampel überfahren wurde und die Ampel rot war. Dies muss im Sequencing hinter der Kalkulation von proximity abgefragt werden. Da die Default proximity 99999.0 ist, gibt es keine Rising Edge bei erstem Start. Weil die Ampel nur unter 100 m sichtbar ist, ist steigt proximity nur bei Ampeldurchfahrt (proximity von 0.0 auf 99999.0) an (Rising Edge == true;). Es wird ein Timer verwendet damit das Flashlight nur kurz leuchtet. Nach einer Sekunde wird das Flashlight auf false zurückgesetzt. Da das Auto circa 50 km/h fährt und die nächste Ampel mindestens 200 m entfernt ist, erreicht das Auto nie innerhalb einer Sekunde die nächste Ampel (Abbildung 12).

Die Flashlight wurde als Message implementiert, damit sie nach außen verfügbar ist und zukünftige Module es nutzen. Es gehört zur Umgebung und ist sichtbar für den Fahrer.

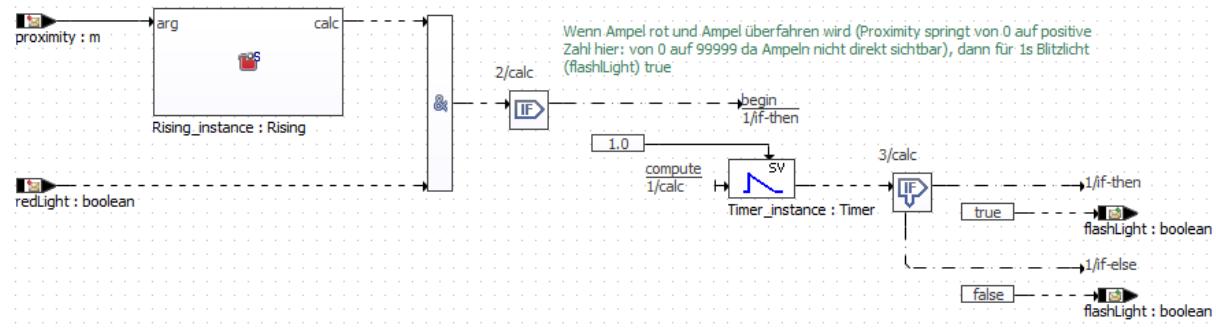


Abbildung 12 Flashlight.bd

### Automatisierter Fahrer

#### Geschwindigkeitsregelung

Der Klassenname ist „CruiseSpeedControl“. Es regelt auf die konstante Geschwindigkeit von 50 km/h.

Ist die Geschwindigkeit kleiner als 50 km/h, so wird 100% Gas gegeben. Wenn aber die Geschwindigkeit größer als 50 km/h ist, so wird mit einem Wert X gebremst. Für Wert X muss gelten:

Durch die Bremspedalstellung X darf keine stärkere Entschleunigung des Fahrzeugs als  $-2,5 \frac{m}{s^2}$  erzeugt werden

#### Ziel:

Das Fahrzeug soll mit annähernd  $-2,5 \frac{m}{s^2}$  entschleunigen. Dafür spielen folgende Faktoren eine Rolle:

1. Geschwindigkeitsabhängiger Luftwiderstand
2. Bremspedalstellung

Für die Umsetzung wird die BrakePedal LookupTable aus dem Car-Modell verwendet und invertiert.

Die Konstante maxDecel gibt den Zielwert von  $2,5 \frac{m}{s^2}$  Entschleunigung an.

Die Lookuptable AirFriction ist eine exakte Kopie aus dem Car-Modell.

Durch Verrechnung von maxDecel und dem geschwindigkeitsabhängigen Luftwiderstand, ergibt sich das nötige Moment zum genauen Entschleunigen mit  $-2,5 \frac{m}{s^2}$ . Der Invertierte BrakePedal Lookuptable nimmt gewünschte Entschleunigung und gibt eine Bremspedalstellung aus. Der invertierte Lookuptable wurde in CharTableTypes definiert. Die Abbildung ist von a auf real.

Das Ergebnis ist eine bremsadaptive Geschwindigkeitsregelung siehe Abbildung 13.

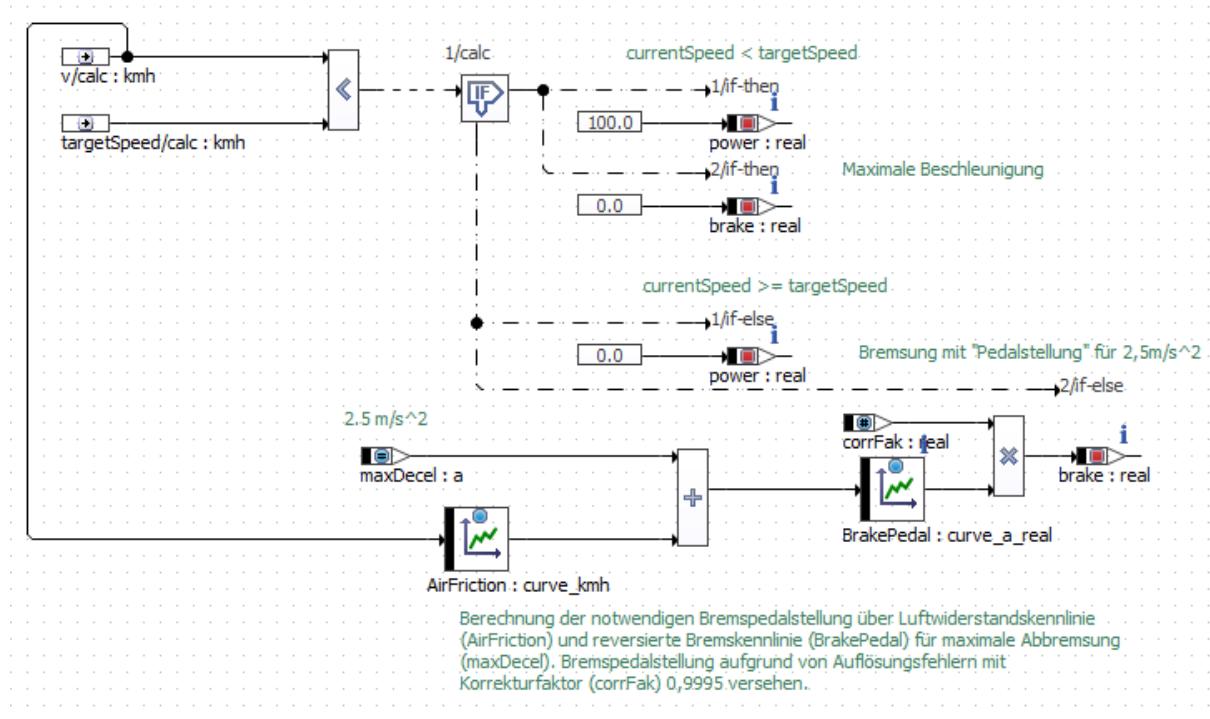


Abbildung 13 CruiseSpeedControl.bd

## Kontrollrechnung der Fahrzeug Entschleunigung

Um die Richtigkeit des Requirements 5 (R5) zu gewährleisten, wird in einem separaten Modul „DeaccelerationControl“ die Entschleunigung des Fahrzeugs gemessen. Hierfür werden drei verschiedene Methoden verwendet:

1. Die Ableitung der Geschwindigkeit nach der Zeit.
2. Theoretische Nachberechnung der Entschleunigung mittels redundanter Implementierung der Adaptiven Bremsregelung
3. Abgriff der Entschleunigung im Fahrzeugmodell über eine Message

Die drei Methoden bringen jeweils verschiedene Vor und Nachteile mit sich:

Methode 1 ist aufgrund der Rundung der Geschwindigkeiten und der Ungenauigkeit bei der Division nicht wirklich präzise.

Methode 2 liefert theoretisch immer den gleichen Entschleunigungswert wie bereits die adaptive Geschwindigkeitsregelung. Aufgrund der identischen Implementierung handelt es

sich hierbei eher weniger um eine Prüfung der Entschleunigung und vielmehr um eine Prüfung der korrekten Rechenausführung.

Methode 3 ist die präziseste Lösung, erfordert jedoch die Veränderung des Fahrzeugmodells, beziehungsweise die Erweiterung um eine Message

Abbildung 14 zeigt die Blockschaltbild-Implementierung der drei Methoden. Alle drei berechneten Kontrollwerte werden in der Experimentierumgebung angezeigt.

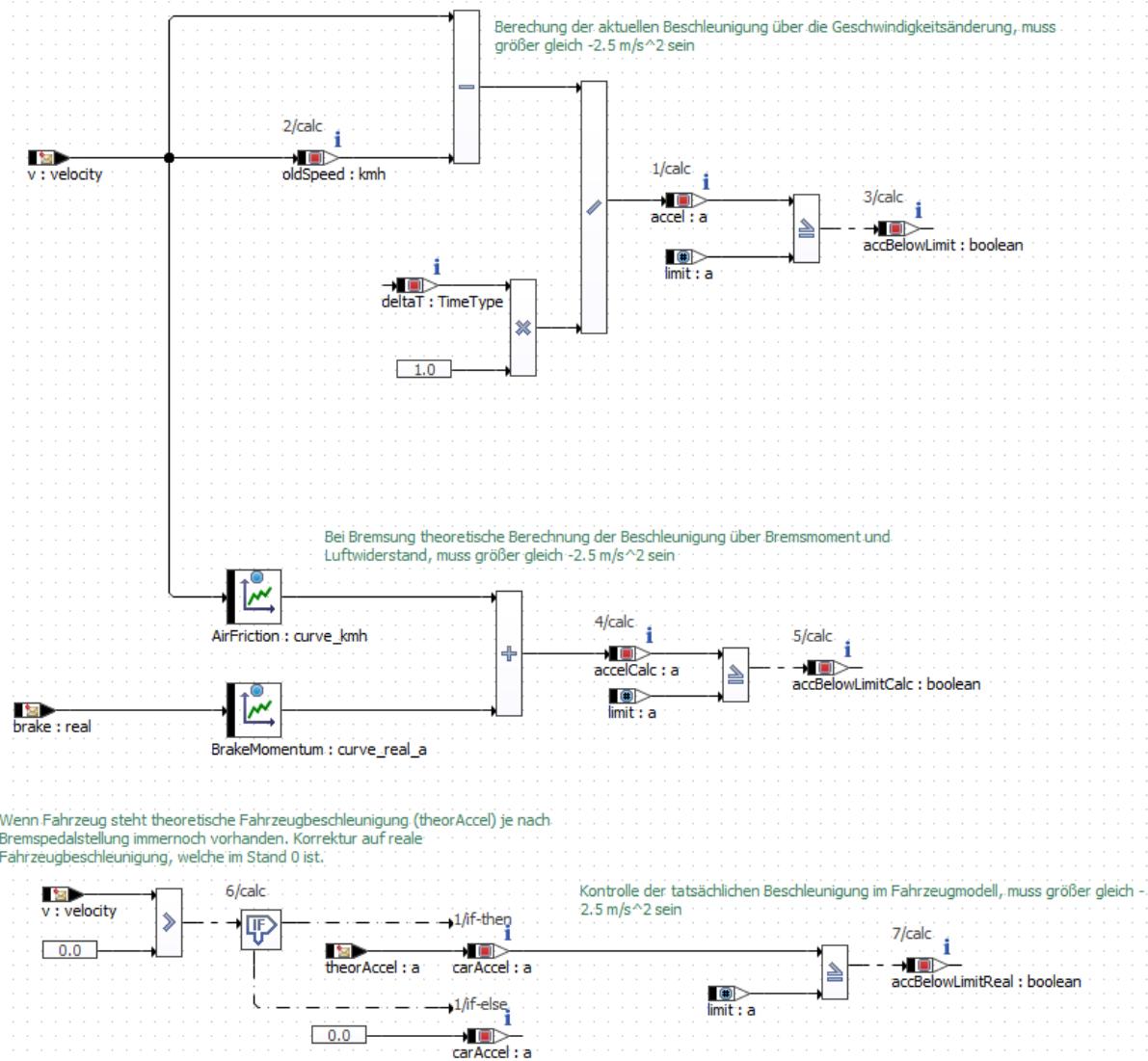


Abbildung 14 DecelerationControl.bd

## Entscheidungsfindung

Eine weitere wichtige Aufgabe des automatisierten Fahrers ist das Treffen von Verkehrssentscheidungen. Bezogen auf die Aufgabenstellung handelt es sich hierbei lediglich um die Evaluation der Ampelsignale. Hierfür wird bei jedem Aufruf des Moduls „DrivingDecision“ zuerst der aktuelle theoretische Bremsweg des Fahrzeugs basierend auf der Geschwindigkeit berechnet.

Für die Berechnung des Anhaltewegs wird eine konstante Entschleunigung von  $2,455 \text{ m/s}^2$  angenommen.

Basierend auf dem aktuellen Anhalteweg wird im nächsten Schritt eine Fahrentscheidung getroffen.

Hierfür wird der Abstand zur Ampel, der Anhalteweg und das Lichtsignal der Ampel berücksichtigt.

Folgende drei Entscheidungskriterien dienen zur Evaluation:

1. Es wird überprüft, ob der Abstand zur Ampel kleiner ist als der Anhalteweg plus einen konstanten Puffer( $\delta_{Brake}$ ) von einem Meter:  $prox \leq anhalteweg + \delta_{Brake}$
2. Es wird überprüft, ob es überhaupt noch möglich ist anzuhalten, also ob folgendes gilt:  $prox > anhalteweg$
3. Es wird überprüft ob es überhaupt nötig ist anzuhalten. Es soll angehalten werden, wenn die Ampel entweder ein rotes oder ein gelbes Lichtsignal anzeigt. Insofern möglich, ist das Anhalten bei einer gelben Ampel, nach dem deutschen Bußgeldkatalog, Pflicht. Ein Verstoß kann mit einem Bußgeld bis zu zehn Euro bestraft werden (vgl. 1).

Diese drei Bedingungen werden mit einem Logikbaustein verundet. Insofern die Ampel Rot und/oder Gelb zeigt und das Fahrzeug in der Lage ist anzuhalten und sich in der Nähe der Ampel befindet wird eine adaptive Bremsung mit ungefähr  $-2,5m/s^2$  eingeleitet.

Kombiniert man die Ungleichungen der Entscheidungskriterien eins und zwei ergibt sich folgende Gleichung:

$$anhalteweg < prox \leq anhalteweg + \delta_{Brake}$$

Hierraus lässt sich ableiten, dass die Konstante  $\delta_{Brake}$  lediglich eine Fensterbreite definiert. Befindet sich der Abstand des Fahrzeugs zur Ampel in diesem Fenster und die Ampel zeigt ein entsprechendes Signal, wird eine Bremsung eingeleitet. Die Konstante  $\delta_{Brake}$  definiert somit auch den maximalen Abstand zur Haltelinie bei  $prox=0$ .

Entscheidet sich der automatisierte Fahrer zum Zeitpunkt  $t$  in einem Funktionsaufruf  $x$  zu bremsen wird in der nachfolgenden Berechnung des Car-Modells auch eine Bremsung in der Geschwindigkeitsberechnung des Fahrzeugs berücksichtigt. Die geringere Geschwindigkeit des Fahrzeugs resultiert im Funktionsaufruf  $x+1$  in einer Verminderung des theoretischen Anhaltewegs. Somit verschiebt sich das Fenster mit der konstanten Breite  $\delta_{Brake}$ , in jedem Funktionsaufruf weiter in Richtung 0. Die Reduktion der Geschwindigkeit hat dabei einen quadratischen Einfluss auf den Anhalteweg und die Verschiebung des Fensters.

$$anhalteweg = \frac{v^2}{2a}$$

Durch die konstante Geschwindigkeitsreduktion des Fahrzeugs sinkt auch die proximity nicht mehr mit einem konstanten Wert, sondern in quadratischer Abhängigkeit von  $v$ :

$$prox(t) = pos_{ampel} - \int (v(t))dt = pos_{ampel} - \left(\frac{v(t)^2}{2}\right)$$

Abbildung 15 zeigt die Implementierung der Theorie in einem Blockschaltbild:

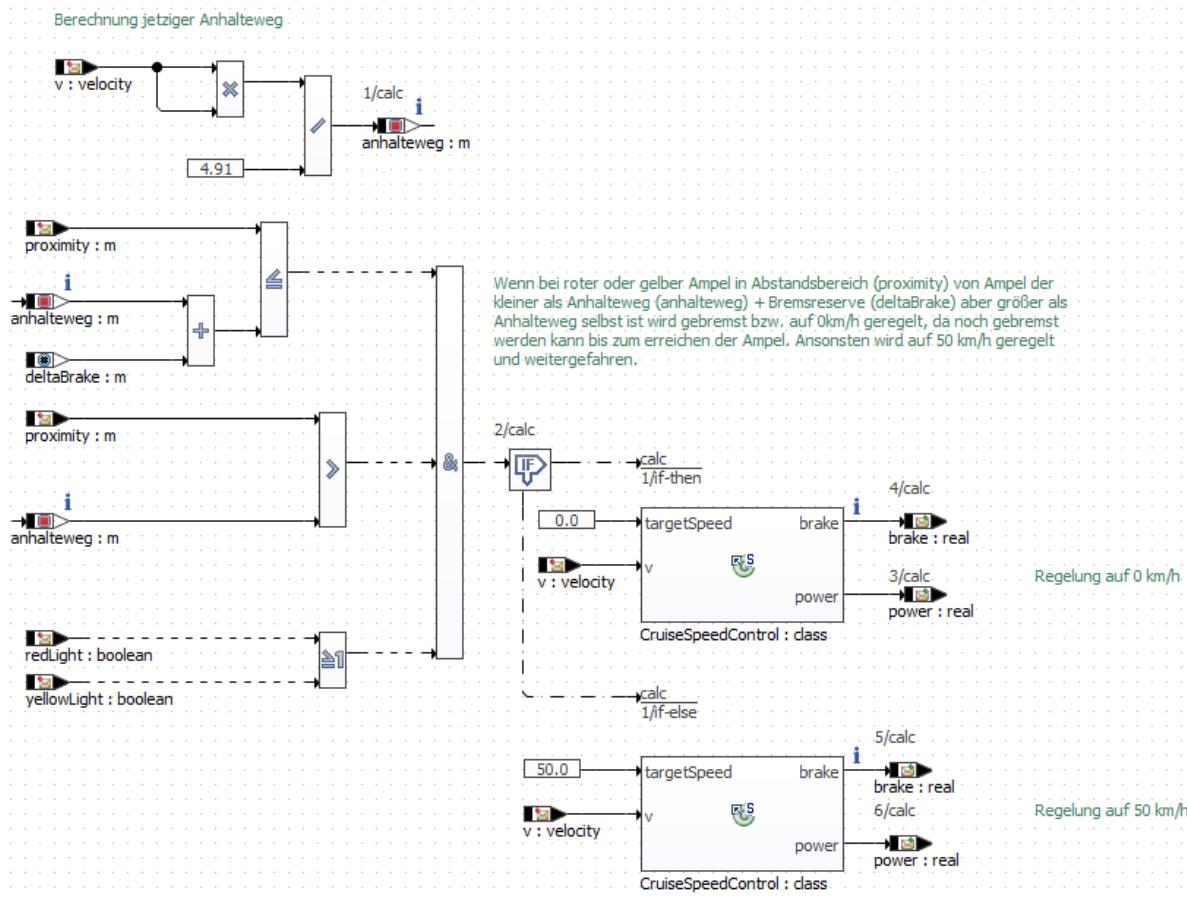


Abbildung 15 DrivingDecision.bd

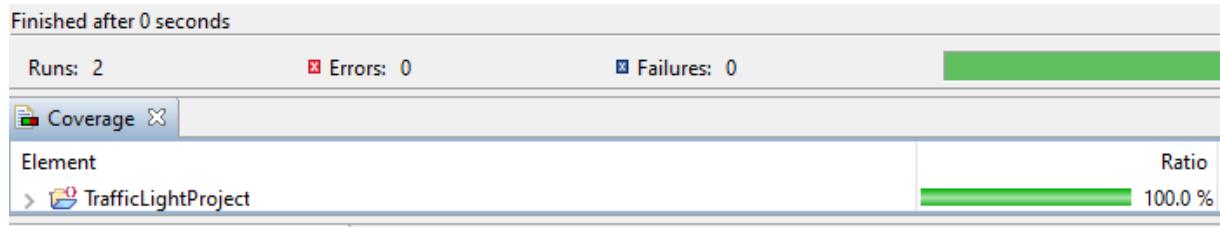
## Unitests

Die Implementierung der Unit-Tests ist in den Abbildungen 16 bis 18 gezeigt.

Für die Klassen CruiseSpeedControl, DrivingDecision, DeaccelerationControl und Flashlight mussten extra Testklassen aufgrund von Messages implementiert werden.

Bei CruiseSpeedControlUnitTest wurde das Beschleunigen und Bremsen, je nachdem was die jetzige Geschwindigkeit und Zielgeschwindigkeit ist, getestet. Es wurde eine Coverage von 100 % erreicht. Dabei musste das Bremspedal sich in einem engen Bereich befinden, um die maximale Abbrenzung nicht zu überschreiten und trotzdem so stark wie möglich zu bremsen.

Im DrivingDecisionUnitTest (Abbildung 16) wurden die zwei Modi „Grün“ und „Rot“ getestet. Der erste Test, testet wenn die Ampel grün ist und die jetzige Geschwindigkeit kleiner der Sollgeschwindigkeit ist und deswegen über CruiseSpeedControl, das Gaspedal betätigt wird. Im zweiten Testfall wird das Bremsen bei roter Ampel getestet. Über CruiseSpeedControl wird die Bremse betätigt. Beim Test für



The screenshot shows a coverage analysis interface. At the top, it says "Finished after 0 seconds". Below that, there are three status indicators: "Runs: 2", "Errors: 0", and "Failures: 0", each accompanied by a colored bar (green for runs, red for errors, blue for failures). The main area is titled "Coverage" and contains a table. The table has two columns: "Element" and "Ratio". Under "Element", there is a single entry: "TrafficLightProject" with a green icon. To its right, under "Ratio", is "100.0 %". Below this table, the code for "DrivingDecisionUnitTest.esdl" is displayed in a code editor. The code is annotated with numbers 1 through 22 on the left side.

```

1 package UnitTests;
2
3 import assertLib.Assert;
4 import Driver.DrivingDecisionTestFunction;
5 import resources.kmh;
6 import resources.m;
7
8 static class DrivingDecisionUnitTest { //100% coverage
9
10     DrivingDecisionTestFunction testObj;
11     real value;
12
13     @Test//Test das bei einer grünen Ampel Gas gegeben wird
14     public void testGreen() {
15         value = testObj.calc(40.0[kmh], false, false, 10.0[m]);
16         Assert.assertTrue(value == 100.0); // power pedal 100
17     }
18     @Test//Test das bei einer roten Ampel mit passender Distanz maximal gebremst wird
19     public void testRed() {
20         value = testObj.calc(50.0[kmh], true, false, 39.5[m]);
21         Assert.assertTrue(value <= 67.0);
22         Assert.assertTrue(value >= 66.0); // brake pedal ~66.66
23     }

```

Abbildung 16 DrivingDecisionUnitTest.esdl

In DecelerationControlUnitTest (Abbildung 17) wird überprüft, ob die Formel für die Berechnung der Deacceleration korrekt ist. In den Testfällen wurden Werte der Bremse eingegeben mit dem Rückgabewert der Deacceleration, welche nicht kleiner als  $-2.5 \frac{m}{s^2}$  sein darf. Hier kann man erkennen das das Auto maximal abremst. Zudem wird noch der Fall abgetestet, dass bei Stillstand die Beschleunigung null ist.

The screenshot shows a software interface with the following details:

- Runs: 3**, **Errors: 0**, **Failures: 0**
- Coverage** section: Element TrafficLightProject, Ratio 100.0 %, Covered Statements 21, Total Statements 21
- DecelerationControlUnitTest.esdl** file content:

```

1 package UnitTests;
2
3 import assertLib.Assert;
4 import Driver.DecelerationControlTestFunction;
5 import resources.kmh;
6 import resources.a;
7
8 static class DecelerationControlUnitTest { //100% coverage
9
10     DecelerationControlTestFunction testObj;
11     integer underLimit;
12     kmh speed;
13     @Test//Test dass bei einer Bremspedalstellung von 66 der Grenzwert eingehalten wird
14     public void testUnderLimit() {
15         testObj.calc(49.9999[kmh], 66.0, 1.0[a]);
16         underLimit = testObj.calc(50.0[kmh], 66.0, 1.0[a]) ;
17         Assert.assertTrue(underLimit==0); // brake < 2.5 m/s^-2
18     }
19
20     @Test//Test das bei einer Bremspedalstellung von 67 der Grenzwert nicht mehr eingehalten wird
21     public void testOverLimit() {
22         underLimit = testObj.calc( 50.0[kmh], 67.0, 1.0[a]);
23         Assert.assertTrue(underLimit==1); // brake > 2.5 m/s^-2
24
25     } //--> CruiseSpeedControlUnitTest nutzt maximal Bremsung
26
27     @Test//Test, dass die reale Fahrzeugbeschleunigung im Stillstand immer null ist
28     public void testStanding() {
29         underLimit = testObj.calc(0.0[kmh], 66.0, 1.0[a]);
30         Assert.assertTrue(underLimit==2); // brake > 2.5 m/s^-2
31
32     }
33
34 }
35 }
```

Abbildung 17 DecelerationControlUnitTest.esdl

Der Flashlighttest (Abbildung 18) testet folgende drei Fälle:

1. Überfahren einer roten Ampel. Hier sollte das Flashlight angehen
2. Überfahren einer grünen Ampel. Hier sollte das Flashlight nicht angehen
3. Stoppen vor einer roten Ampel. Hier sollte das Flashlight nicht angehen

Finished after 0 seconds

Runs: 3 Errors: 0 Failures: 0

Coverage

Element	Ratio
TrafficLightProject	100.0 %

FlashlightUnitTest.esdl

```

1 package UnitTests;
2
3 import assertLib.Assert;
4 import TrafficLight.FlashlightTestFunction;
5 import resources.m;
6
7 static class FlashlightUnitTest { //100% coverage
8
9     FlashlightTestFunction testObj;
10    boolean redlight;
11    boolean flash;
12
13    @Test//Test auf Flashlight bei überfahren einer roten Ampel
14    public void testFlashDrive() {
15        redLight = true;
16        testObj.calc(0.0[m], redLight);
17        flash = testObj.calc(99999.0[m], redLight);
18        Assert.assertTrue(flash);
19    }
20
21    @Test//Test auf kein Flashlight bei überfahren einer grünen Ampel
22    public void testNoFlashDrive() {
23        redLight = false;
24        testObj.calc(40.0[m], redLight);
25        flash = testObj.calc(99999.0[m], redLight);
26        Assert.assertFalse(flash);
27    }
28
29    @Test//Test auf kein Flashlight bei nicht überfahren einer roten Ampel
30    public void testNoFlashStand() {
31        redLight = true;
32        testObj.calc(0.0[m], redLight);
33        flash = testObj.calc(0.0[m], redLight);
34        Assert.assertFalse(flash);
35    }

```

Abbildung 18 FlashlightUnitTest.esdl

## D5: Unitests

Für alle selbsterstellten Klassen wurden Unitests erstellt, siehe D3 und D4. **Alle Unitests haben eine Coverage von 100 %** und wurden erfolgreich abgeschlossen.

## D6: Experimentierumgebung

Das Experiment Environment wurde wie in Abbildung 19 dargestellt aufgebaut. Großen Wert wurde dabei auf die Elementarten, Einfärbung, Größe und Gesamtdarstellung gelegt, um ein intuitives Verständnis für das Modell zu schaffen. Die folgende Aufgliederung bezieht sich auf die Gruppierungsnamen:

### Total values since start:

Oben links befinden sich die Gesamtvariablen, welche ab Simulationsstart nicht zurückgesetzt werden und den Gesamtfortschritt der Simulation anzeigen. Oben befindet sich dabei *x*, die Gesamtstrecke in m und unten *time*, die Gesamtzeit in s.

### Vehicle movement:

Unter den Gesamtvariablen befinden sich die aktuellen Bewegungsdaten. Dazu gehört die Strecke (*dist*) in m innerhalb der aktuellen Runde von 1000 m, welche orange dargestellt ist, sowie die aktuelle Geschwindigkeit (*v*) in km/h, blau dargestellt. Diese Farben korrespondieren mit den Linienfarben im mittleren Hauptplot zur Fahrzeugbewegung.

#### **Vehicle movement plot:**

In selber Reihenfolge und mit denselben Einheiten wie in der vorherigen Gruppierung (Rundenstrecke oben, Geschwindigkeit unten) werden im Hauptplot die Variablen geplottet. Dieser Plot stellt die Hauptübersicht dar, was aktuell im Modell geschieht.

#### **Driver:**

Hier, im Fahrer, werden die aktuellen Fahrparameter der Steuerung angezeigt. Oben befindet sich der Bremswert in %, unten die Gaspedalstellung in %.

#### **Proximity next TrafficLight:**

Dieses Feld gibt den aktuell berechneten Abstand zur nächsten Ampel in m an, sofern diese sichtbar ist. Ansonsten wird der Wert 99999 angezeigt.

#### **Approx. braking distance:**

Im Feld wird der abgeschätzte Anhalteweg auf Basis der aktuellen Geschwindigkeit mit der maximal zulässigen Verzögerung von  $2,5 \frac{m}{s^2}$  angezeigt.

#### **TrafficLight:**

Angezeigt werden die aktuellen Ampelfarben in Abhängigkeit der Zeit. Die dargestellten LEDs leuchten in jeweiliger Ampelfarbe.

#### **Flasher:**

Flasher ist der Ampelblitzer bei Überfahrt während der Rotphase. Der Blitz leuchtet für 1s blau bei Aktivierung.

#### **Acceleration gauge:**

Das Rundinstrument zeigt die aktuelle Beschleunigung in  $\frac{m}{s^2}$  an.

#### **Acceleration:**

Das Feld zeigt die aktuelle, tatsächliche Fahrzeugbeschleunigung in  $\frac{m}{s^2}$  an.

#### **Max. deceleration exceeded:**

Die beiden LEDs leuchten orange, sofern das Verzögerungslimit eingehalten wird. Die obere LED ist hierbei die interne Nachrechnung, die untere LED ist die im Fahrzeug abgegriffene Beschleunigung.

#### **Max. deceleration exceeded plot:**

Darstellung eines Plots der in Max. deceleration exceeded dargestellten Variablen in selber Reihenfolge. Sofern das Verzögerungslimit eingehalten wird, befinden sich beide Linien auf 1, ansonsten ist eine Linie / ein Peak auf 0 zu sehen.

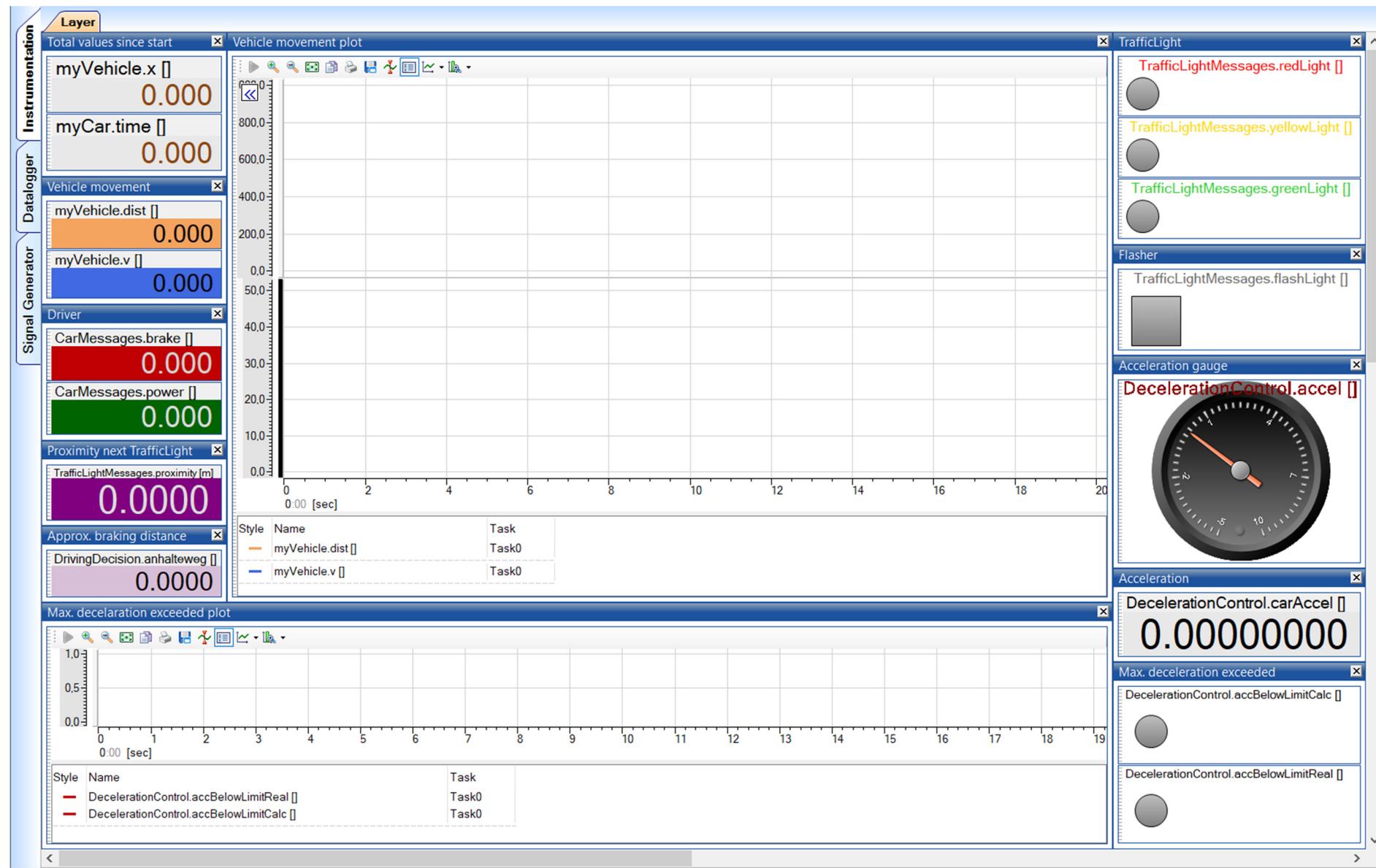


Abbildung 19 Experimentierumgebung

In Abbildung 20 ist das erste Anfahren und der aktuelle Abbremsvorgang für die erste Ampel nach Umschaltung auf Rot dargestellt.

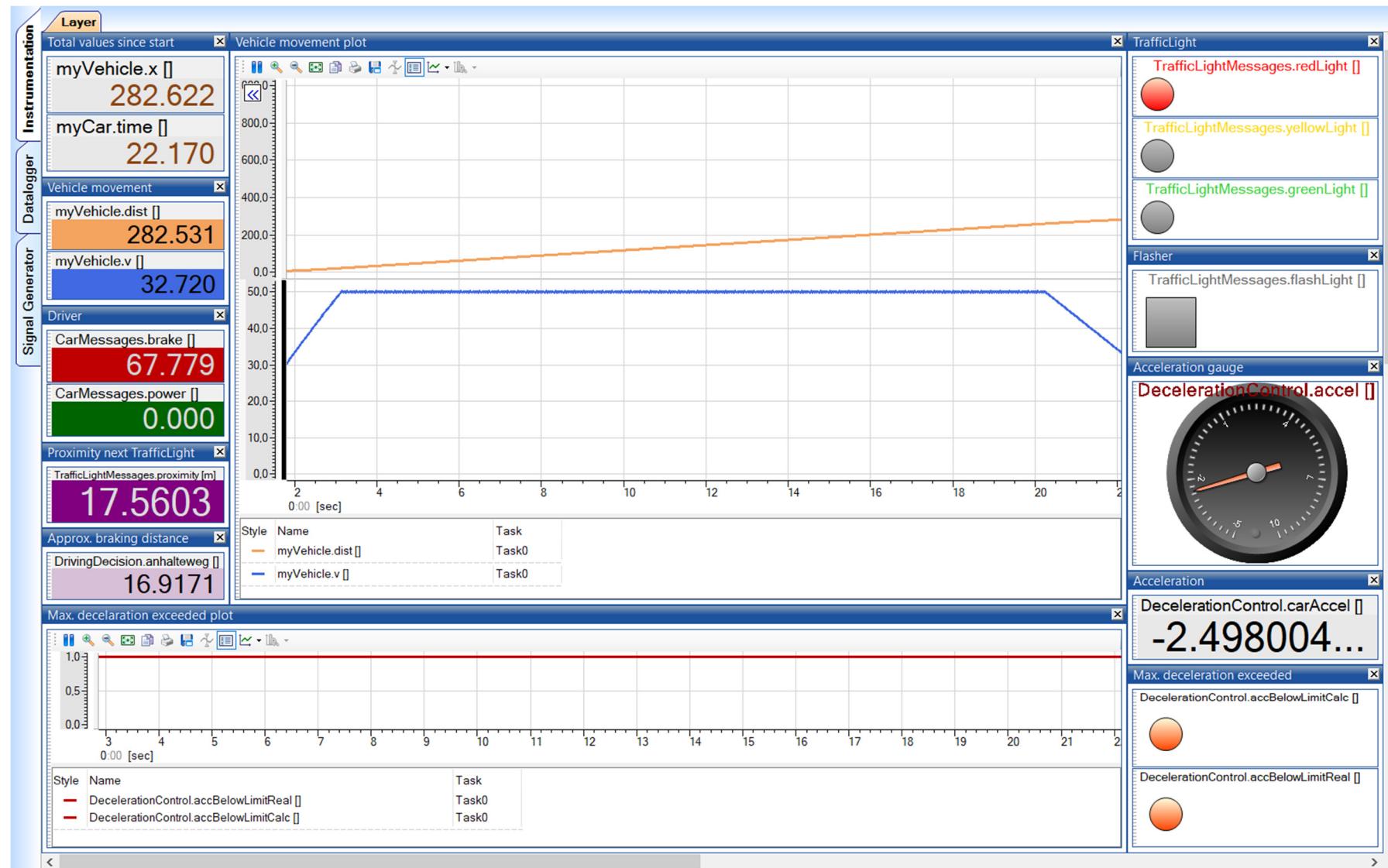


Abbildung 20 Abbremsvorgang

In Abbildung 21 ist das Abbremsen an der dritten Ampel und der aktuelle Anfahrvorgang nach Umschaltung auf grün dargestellt.

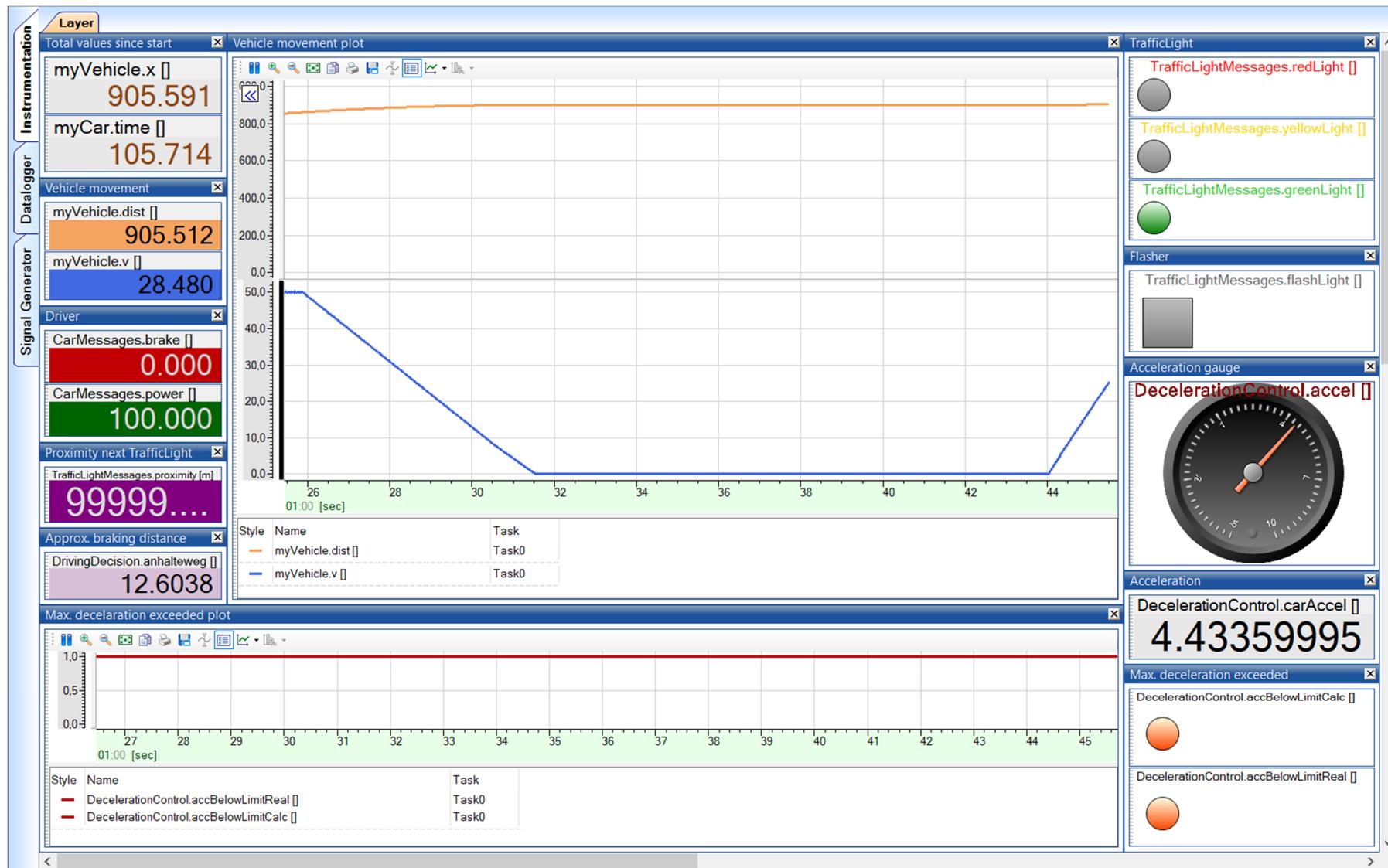


Abbildung 21 Abbremsen und Anfahren

## D7: Gelbzeiten

Die Geschwindigkeit des Fahrzeugs bei Bremsung lässt sich mit  $v_{Start} = a * t$  beschreiben.

Hierbei ist  $v_{Start}$  die Startgeschwindigkeit des Fahrzeugs vor Einleitung der Bremsung,  $a$  die Verzögerungsrate bei Bremsung und  $t$  die Dauer ab Start der Bremsung.

Über die Integration dieser Geschwindigkeit, kann die zurückgelegte Strecke ermittelt werden. Es gilt:

$$s = \int (v_{Start} - a * t) = v_{Start} * t - 0.5 * a * t^2 = (v_{Start} - 0.5 * a * t) * t$$

Der minimale Bremsweg des Fahrzeugs lässt sich mit der maximalen Verzögerungsrate  $a_{max}$  und der Zeit für den Bremsvorgang bis zum Stillstand:  $t_{Bremsung} = \frac{v_{Start}}{a_{max}}$  berechnen.

Setzt man  $a = a_{max}$  und  $t = t_{Bremsung}$  in die Berechnung der zurückgelegten Strecke ein, folgt:

$$\begin{aligned} s_{Bremsung} &= (v_{Start} - 0.5 * a_{max} * t_{Bremsung}) * t_{Bremsung} \\ &= \left( v_{Start} - 0.5 * a_{max} * \frac{v_{Start}}{a_{max}} \right) * \frac{v_{Start}}{a_{max}} = 0.5 * v_{Start} * \frac{v_{Start}}{a_{max}} \\ &= 0.5 * \frac{v_{Start}^2}{a_{max}} \end{aligned}$$

Nach der Umschaltung von Grün auf Gelb muss entweder in der verbleibenden Strecke bis zur Ampel angehalten oder in der Gelbzeit  $t_{Gelb}$  die Ampel mit konstanter Geschwindigkeit überfahren werden können.

Für die rechtzeitige Bremsung muss mindestens die Reststrecke ( $s_{Rest}$ )  $s_{Bremsung}$  bis zur Ampel vorhanden sein, es gilt:  $s_{Rest} \geq s_{Bremsung}$ .

Ist hingegen  $s_{Rest} < s_{Bremsung}$ , also im Grenzwert  $s_{Rest} = s_{Bremsung}$  muss die Ampel mit der konstanten Geschwindigkeit  $v_{Start}$  in der Zeit  $t_{Überfahren} \leq t_{Gelb}$  überfahren werden können.

Es gilt:  $t_{Überfahren} = \frac{s_{Rest}}{v_{Start}}$  und demnach im Grenzwert:

$$t_{Überfahren} = \frac{s_{Bremsung}}{v_{Start}} = 0.5 * \frac{\frac{v_{Start}^2}{a_{max}}}{v_{Start}} = 0.5 * \frac{v_{Start}}{a_{max}}$$

Als Anforderung an die Gelbzeit lässt sich schlussfolgern:  $t_{Gelb} \geq 0.5 * \frac{v_{Start}}{a_{max}}$ .

Nur so kann die bei Umschaltung von Grün auf Gelb die Möglichkeit der rechtzeitigen Bremsung oder des Überfahrens mit konstanter Geschwindigkeit sichergestellt werden.

Für die konstante Maximalverzögerung von  $2,5 \frac{m}{s^2}$  ist die notwendige Gelbzeit über dem Geschwindigkeitslimit / der gefahrenen Geschwindigkeit in Abbildung 22 dargestellt.

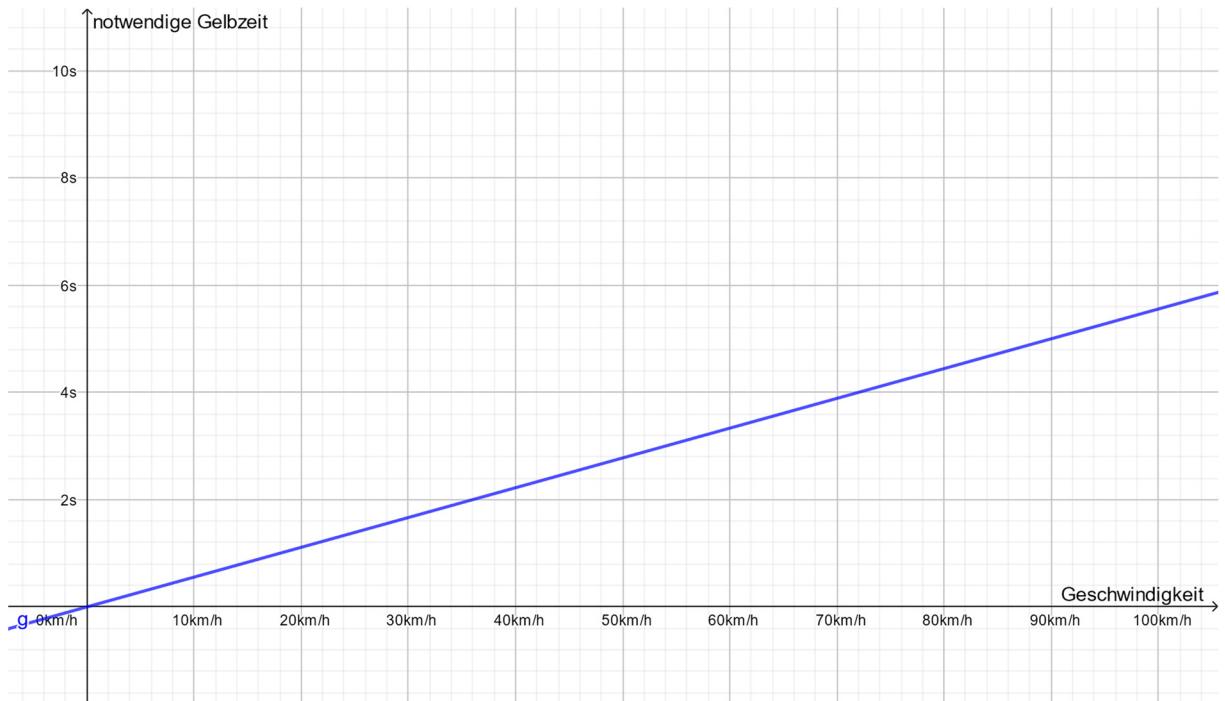


Abbildung 22 Notwendige Gelbzeit

## D8: Konstante Grünphasen

### Aufgabe:

How should the relative timings of the traffic lights be chosen to realize a constant (**green phase**) go while cruising at speeds of 45 to 50 km/h?

Für die Beantwortung der Frage D8 müssen in erster Linie einige Definitionen festgehalten werden:

1. Beim Start der Simulation ( $t=0$ ) befindet sich jedes Fahrzeug  $F_x$  bereits auf seiner Ausgangsgeschwindigkeit  $v$ , der Beschleunigungsprozess von  $0 \text{ km/h}$  auf die Ausgangsgeschwindigkeit  $v$  wird somit vernachlässigt.
  2. Unter einem „constant green phase go“ wird verstanden, dass zu jedem Zeitpunkt  $t$ , an dem sich das Fahrzeug  $V$  an den Positionen  $P_1 = 300 \text{ m}$ ,  $P_2 = 500 \text{ m}$  oder  $P_3 = 900 \text{ m}$  befindet, die jeweilige Ampel sich im Ampelzustand „Grün“ befindet.
  3. Unter einem Ampelzyklus wird der einmalige Ablauf der Ampelzustände (Gelb, Rot, Rot/Gelb, Grün) mit der Dauer  $T_A$  verstanden.
- 3.1 Jeder Ampelzustand ( $[Gelb,0]$ ,  $[Rot,1]$ ,  $[Rot&Gelb,2]$ ,  $[Grün,3]$ ) hat eine definierte Dauer  $T_z$
- 3.2 Es gilt:  $\sum_{z=0}^3 (T_z) = T_A$ , die Summe aller Ampelzustandszeiten  $T_z$  entspricht der Ampelzykluszeit  $T_A$
- 3.3 Der Startwert  $m$  eines Ampelzyklus, wobei gilt  $m < T_A$ , definiert zum Zeitpunkt  $t = 0$  die Verschiebung des Ampelzyklus gegenüber dem gewöhnlichen Ampelzyklus, definiert in Annahme 3.0.

**Antwort 1:****Annahmen:**

- Das angegebene Geschwindigkeitsintervall  $I = [45,50]$  (in  $\frac{km}{h}$ ) definiert lediglich eine Reihe an Möglichkeiten, aus welchen ein Geschwindigkeitswert  $v$  gewählt werden kann, womit sich das Fahrzeug F dann konstant fortbewegt.
- Unter dem Ausdruck „relative timings“ wird verstanden, dass jede Ampel X den in Requirement 1 (R1) definierten Ampelzyklus durchlaufen muss und lediglich ein unterschiedlicher Startwert  $m$  für die jeweiligen Ampel definiert werden kann.

**Lösung:**

Unter den getroffenen Annahmen gibt es keine Möglichkeit einen „constant green phase go“ zu implementieren.

**Begründung**

Beweis durch Widerspruch: Ein Fahrzeug F, dass sich mit einer konstanten Geschwindigkeit von  $v = 50 \frac{km}{h}$  bewegt benötigt 72 s für genau eine Runde auf dem Rundtrack. Befindet sich das Fahrzeug F an der Position  $P1 = 300m$  (Ampel1) zum Zeitpunkt  $t = x$ , wird sich das Fahrzeug F auch wieder an der Position  $P1 = 300m$  (Ampel1) zum Zeitpunkt  $t = x + 72$  befinden. Der Ampelzyklus der Ampel 1 hat eine Zyklusdauer von  $T_z = 60s$ . Dies wiederum führt zu einer Verschiebung des Ampelzustands um 12s. Nach spätestens zwei Runden ergibt sich somit eine Verschiebung um 24s, was außerhalb des Grünphasentimings von 16s liegt (Referenz R1)

**Antwort 2:****Annahmen:**

- Das angegebene Geschwindigkeitsintervall  $I=[45,50]$  (in km/h) definiert lediglich eine Reihe an Möglichkeiten, aus welchen ein Geschwindigkeitswert  $v$  gewählt werden kann, womit sich das Fahrzeug F dann konstant fortbewegt.
- Unter dem Ausdruck „relative timings“ wird verstanden, dass jede Ampel X einen komplett variablen Ampelzyklus besitzen kann, mit von R1 variierenden  $T_z$  Zeiten.

**Lösung:**

Unter den getroffenen Annahmen gibt es eine Möglichkeit einen „constant green phase go“ zu implementieren. Alle Ampeln müssen dauerhaft auf grün geschalten sein.

**Begründung****Grenzwerte:**

Ein Fahrzeug F1, dass sich mit der konstanten Geschwindigkeit  $v=50$  km/h bewegt, holt nach einer Anzahl von X Runden ein langsameres Fahrzeug F2 mit einer Geschwindigkeit  $v = 45 \frac{km}{h}$  ein.

Erweitert man diese Beobachtung auf das gesamte Intervall, ergibt sich Folgendes:

Betrachtet man ein kontinuierlich verteiltes Spektrum an Fahrzeuggeschwindigkeiten im Intervall  $I = [45,50] \left( \text{in } \frac{\text{km}}{\text{h}} \right)$  und weist einer unendlichen Anzahl an Fahrzeugen jeweils eine unterschiedliche Geschwindigkeit aus dem Intervalls I zu. Dann gilt Folgendes:

Es existiert eine Runde y für die gilt:  $y > x$  und  $y > 0$ , in welcher sich auf jeder Position P des Rundtracks ein Fahrzeug mit einer konstanten Geschwindigkeit v befindet.

Dies wiederum führt dazu, dass in der Runde y eine Konstellation an Ampelüberfahrten entsteht, wobei zu jedem möglichen Zeitpunkt t innerhalb dieser Runde ein Fahrzeug k mit der konstanten Geschwindigkeit aus dem Intervall I über eine Ampel fährt. Um dabei einen „constant green phase go“ zu ermöglichen ist die einzige Möglichkeit alle Ampeln dauerhaft im Zustand „Grün“ zu halten.

### **Antwort 3:**

#### **Annahmen:**

- Ein Fahrzeug F kann sich mit variierender Geschwindigkeit im Intervall  $I = [45,50] \left( \text{in } \frac{\text{km}}{\text{h}} \right)$  bewegen, wobei eine maximale Geschwindigkeitsreduktion von  $2,5 \frac{\text{m}}{\text{s}^2}$  erlaubt ist.
- Unter dem Ausdruck „relative timings“ wird verstanden, dass jede Ampel X den in Requirement 1 (R1) definierten Ampelzyklus durchlaufen muss und lediglich ein unterschiedlicher Startwert m für die jeweiligen Ampel definiert werden kann.

#### **Lösung**

Unter den getroffenen Annahmen gibt es keine Möglichkeit einen „constant green phase go“ zu implementieren.

#### **Begründung**

Hierbei lässt sich die gleiche Begründung wie in Antwort 1 treffen. Bereits für ein Fahrzeug mit 50 km/h entsteht von Runde zu Runde eine Zeitdifferenz im Ampelzyklus. Ein Fahrzeug, welches sich mit einer konstanten Geschwindigkeit von 45 km/h bewegen würde, benötigt für eine Runde auf dem Rundtrack 80s. Somit kann davon ausgegangen werden, dass selbst bei einer variablen Geschwindigkeit die Umrundungsdauer immer zwischen 72s und 80s bewegt, insofern die Geschwindigkeit des Fahrzeugs innerhalb des Intervalls I liegt ( $I=[45,50](\text{in km/h})$ ). Somit entsteht nach mindestens zwei Runden eine Zyklusabweichung im Intervall  $K=[24,40](\text{in s})$ , was in jedem Fall zu einer Ampelüberfahrt außerhalb des Zustands „Grün“ führt.

## **D9: Verbesserung mittels V2X oder Verwendung von Ampeltimings / -positionen**

#### **Aufgabe:**

How could the situation be improved with

- when V2X allows more information to be exchanged between car and traffic lights
- if R5 would be relaxed such that positions and timing could be used in motion planning?

a)

Mit V2X können alle denkbaren Informationen zwischen Ampel und Fahrzeug übertragen werden. Eine real bereits umgesetzte Möglichkeit ist die Übertragung der aktuellen Ampelphase so wie die (Rest) Zeiten der einzelnen Phasen der Ampel. Mit diesen Informationen kann der Verkehrsfluss verkontinuierlicht werden. Die offizielle Zielreichweite für V2X-Kommunikation beträgt 700 m. So kann bereits 700 m vor der Ampel anhand der aktuellen Geschwindigkeit und der Ampelposition die voraussichtliche Ampelphase bei Erreichen der Ampel berechnet werden. Auch kann die notwendige Anpassung der Geschwindigkeit im gesetzlichen Rahmen bestimmt werden, welche notwendig ist, um die Ampel bei Grünphase spätere Abbremsung zu erreichen. Die verbleibende Zeit bis zur nächsten Grünphase sowie die berechnete Geschwindigkeit sollte dem Fahrer als neue Empfehlung vorgeschlagen werden. Bei teil- und vollautonomen Fahrzeugen sollte die Geschwindigkeit kontinuierlich geregelt werden. So kann maximale Energieeffizienz durch Gleiten(Ausrollen), (Motor-)Bremse und Beschleunigung erzielt werden. Bei Annahme, dass mehrere Fahrzeuge vorhanden sind, welche nicht über diese Technologie verfügen bzw. diese nicht nutzen, sollte zusätzlich die Anzahl und Länge der haltenden Fahrzeuge übermittelt werden. Diese Erfassung kann über vorhandene Induktionsschleifen und/oder Ampelkameras bestimmt und somit mit in der Berechnung berücksichtigt werden. Dabei muss die voraussichtliche Beschleunigung der Fahrzeuge wie auch deren Abstand zueinander berücksichtigt werden, um die Verzögerungszeit, bis Erreichen der Ampel zu bestimmen und eine ideale Regelung zu gewährleisten (vgl. 2, 3)

.

b)

Eine Anpassung von R5 wäre ähnlich zur V2X. Für die Bekanntheit der Informationen sind zwei Versionen möglich:

1. Dem Fahrer sind die Ampelpositionen und die Dauer der Ampelphasen bekannt
2. Dem Fahrer sind die Ampelpositionen, die Dauer der Ampelphasen und die jeweilis anliegende Ampelphase bekannt

Erstere Version bietet zunächst wenig Verbesserungspotential, da immer noch starke Einschränkungen aufgrund der geringen Sichtweite von 100 m und der spontanen Umschaltung zwischen Ampelphasen bestehen.

Sobald der Fahrer allerdings der ersten in Sichtweite umschaltenden Ampel begegnet, kann er eine interne Mitrechnung der Ampelphasen starten und sich "auf Synchronisieren". Für den folgenden Verlauf der ersten Version besteht kein Unterschied mehr zur zweiten Version.

Zweitere Version beschreibt, das ideale V2X-System ohne die Einbeziehung anderer Fahrzeuge. So ist stets der Verlauf jeder Ampel bekannt. Die Erkennung zusätzlicher Fahrzeuge durch die Ampel ist hierbei nicht relevant, da nur ein Fahrzeug im Simulationsbeispiel vorhanden ist. Für den simulierten Fahrer können mit den vorhandenen Informationen verschiedene Strategien implementiert werden:

- Maximale Effizienz:

Der Fahrer kann intern fortlaufend eine Anzahl von x Runden simulieren, um die Strategie der geringsten Energienutzung zu bestimmen.

Dabei sollte der Regler, wenn möglich auf Bremsungen (Energieabträge) verzichten und sich einer konstanten Geschwindigkeit annähern, sodass stetig die Ampeln bei grüner Ampelphase erreicht werden.

Bei mehreren Möglichkeiten der Geschwindigkeitswahl sollte im Modell stets eine niedrigere Geschwindigkeit aufgrund des geringeren Luftwiderstands bevorzugt werden.

- Höchster Komfort:

Für die Fahrt mit höchstem Komfort wird Komfort als die Abwesenheit von Beschleunigung definiert. Im Modell besteht eine Differenzierung zur vorherigen Strategie, da die Motoreffizienz hier irrelevant ist und somit für die maximale Effizienz auch hohe Beschleunigungswerte denkbar sind. Zusätzlich soll für den Komfort die Anforderung einer möglichst hohen Geschwindigkeit bestehen.

Die beiden Faktoren zur Fahrerentscheidung können hierfür mittels einer Kostenfunktion individuell gewichtet und das globale Minimum bestimmt werden. Wie auch zuvor erfolgt die Berechnung durch Simulation folgender Runden.

- Geringste Zeit:

Für die schnellste Durchfahrt der Ampeln muss eine Zielposition bestimmt werden, um die ideale Strategie zu ermitteln. Hierbei kann es sinnvoll sein zunächst stärker zu beschleunigen, um zu einem späteren wieder bremsen zu müssen, um keine Grünphase zu verpassen. Auch sollte das Fahrzeug situationsbedingt in ausreichend nahem Ampelabstand nicht vollständig abgebremst, sondern im Rollen gehalten werden. Diese Funktion basiert auf der Information wann die Ampel umschaltet. Die Argumentation gegen das vollständige Abbremsen lokal betrachtet ist idealisiert (Auto hält an Haltelinie, ohne Luftwiderstand und konstanten Radmoment) wie folgt:

Beschleunigung von Haltelinie:

$$s_{Haltelinie} = \frac{1}{2} * a_{max} * t^2$$

Beschleunigung aus dem Rollen:

$$\begin{aligned} v_{Gesamt} &= v_{Rollen} + a_{max} * t \\ \rightarrow \int s_{Rollen} &= v_{Rollen} * t + \frac{1}{2} * a_{max} * t^2 \end{aligned}$$

Es muss zusätzlich die bis zur Haltelinie der zu überfahrenden Strecke abgezogen werden:

$$s_{Rollen} = v_{Rollen} * t + \frac{1}{2} * a_{max} * t^2 - s_{Abstand}$$

Um einen temporären Vorteil durch das Rollen zu erzielen, muss  $s_{Rollen} > s_{Haltelinie}$  in der gegebenen Zeitspanne  $t$  sein.

$$\begin{aligned} s_{Rollen} &> s_{Haltelinie} \\ \rightarrow v_{Rollen} * t + \frac{1}{2} * a_{max} * t^2 - s_{Abstand} &> \frac{1}{2} * a_{max} * t^2 \\ \rightarrow v_{Rollen} * t - s_{Abstand} &> 0 \\ \rightarrow v_{Rollen} * t &> s_{Abstand} \end{aligned}$$

Bei gegebener Rollgeschwindigkeit und dem Ampelabstand bei Umschaltung auf Grün (kann vorab durch Bekanntheit des Ampelverlaufs berechnet werden) hängt der Vorteil von der Dauer der Beschleunigung ab.

In diesem Projekt ist die Dauer durch das Erreichen des Tempolimits von 50 km/h beschränkt.

Ob eine möglichst hohe Durchschnittsgeschwindigkeit bis zur nächsten Ampel benötigt wird, hängt jedoch von den Ampelphasen und vom Ampelabstand ab.

Allgemein kann die Berechnung mit  $t_{Ampel\_zu\_Ampel} < t_{Grün} + t_{Gelb}$  durchgeführt werden.  $t_{Ampel\_zu\_Ampel}$  ist dabei vom Geschwindigkeitsverlauf, also der Fahr- / Rollstrategie des Fahrers abhängig.

## D12: Auswirkung von Reaktion und anderen Verzögerungen

Die durchschnittliche Reaktionszeit des Menschen beträgt 180 ms, sofern keine Blickzuwendung benötigt wird, also, dass zu erkennende Objekt im Hauptsichtfeld des Fahrers liegt.

Bei notwendiger Blickzuwendung erhöht sich diese Reaktionszeit auf durchschnittliche 350 ms.

Bei der Fahrtentscheidung bezüglich einer umschaltenden Ampel kann demnach von 180 ms ausgegangen werden, da Ampeln im Straßenverkehr bereits einige Zeit zuvor sichtbar sind und anzunehmen ist, dass der Fahrer sich einer möglichen Umschaltung und der Beachtung der Ampelfarbe bewusst ist. Diese Zeit erhöht sich der Praxis um die Fußumsetz- und Ansprechzeit des Systems bis zur Bremsung mit der gewünschten Verzögerung.

Durchschnittlich werden für die Summe beider Zeiten 600ms-800ms bei überraschender Vollbremsung angenommen. Da es sich bei Ampeln um eine vorhersehbare Bremsung mit moderater Verzögerung (hier:  $a_{max} = 2,5 \frac{m}{s^2}$ ) handelt und nicht um eine überraschende Notbremsung mit über  $10 \frac{m}{s^2}$ , wie es bspw. die eines VW-Golf ( $a_{max} = 10,6 \frac{m}{s^2}$ ), ist hier die Gesamtreaktionszeit hingegen als geringer anzunehmen (vgl. 4, 5).

Die Fußumsetzzeit bleibt bei Fußpositionierung auf dem Gaspedal konstant.

Für die Ansprechzeit wurde folgende Abschätzung gemacht:

$$F_{Auto} = m_{Auto} * a_{Auto} \rightarrow a_{Auto} \sim F_{Auto}$$

Dabei gilt aufgrund der Hebelgesetze von Fahrbahn zu Rad zu Bremse im Haftungsbereich:

$$F_{Auto} \sim F_{Bremskraft}$$

Nach der Gleitreibungskraft an der Bremsscheibe / der Bremstrommel gilt:

$$F_{Bremskraft} = \mu_{Gleitreibung} * F_{Bremszylinder}$$

somit

$$F_{Bremskraft} \sim F_{Bremszylinder}$$

Der notwendige Bremsdruck ist nach

$$F_{Bremszylinder} = p_{Bremsleitung} * A_{Zylinderstirnfläche(n)}$$

bei konstanter Fläche:

$$F_{Bremszylinder} \sim p_{Bremsleitung}$$

Schlussfolgernd gilt:

$$a_{Auto} \sim p_{Bremsleitung}$$

Die Systemverzögerung entsteht mehrheitlich durch Variabilität im Volumen des Bremsystems, bspw. durch Ausdehnung von Bauteilen wie Bremsschläuchen. Die Gegenkraft der Bauteil kann aufgrund der sehr geringen Flächenänderung nach  $F = p * A$  als idealisiert als lineare Ursprungsgerade angenommen werden. Demnach gilt vereinfacht  $p_{Bremsleitung} \sim t_{Bremsansprechen}$ . Kombiniert ergibt sich die Abschätzung:  $a_{Auto} \sim t_{Bremsansprechen}$

Nach eigener Messung wird der konstante Anteil der Fußumsetzzeit auf 290ms abgeschätzt. Im Mittel der Gesamtzeit von 700ms werden demnach für die Ansprechzeit 310ms bei Vollbremsung benötigt. Die Verzögerung des VW-Golfs wird als durchschnittliche Verzögerung für die mittlere Gesamtzeit angenommen.

Nach der zuvor erörterten Proportionalität ergibt sich:

$$10,6 \frac{m}{s^2} \text{ entspricht } 310 \text{ ms}$$

$$\rightarrow 2,5 \frac{m}{s^2} \text{ entspricht ca. } 73 \text{ ms}$$

Die anzunehmende Gesamtreaktionszeit ergibt sich aus der Summe aller Verzögerungs-/Reaktionszeiten:

$$\begin{aligned} t_{Gesamtreaktion} &= t_{Reaktion} + t_{Fußumsetzung} + t_{Bremsansprechen} \\ &= 180ms + 290ms + 73ms = 543ms \end{aligned}$$

Diese zusätzliche Zeit von 543ms für eine moderate Verzögerung von  $2,5 \frac{m}{s^2}$  muss mit in die Berechnung der Gelbzeit nach D7 eingehen. In die Rechnung des Anhaltewegs muss die zurückgelegte Strecke innerhalb der Reaktionsphase auf den Bremsweg summiert werden.

Es gilt:

$$\begin{aligned} s_{Anhalten} &= s_{Bremsung} + v_{Start} * t_{Gesamtreaktion} \\ &= \frac{1}{2} * \frac{v_{Start}^2}{a_{max}} + v_{Start} * t_{Gesamtreaktion} \text{ mit } a_{max} = 2,5 \frac{m}{s^2} \text{ und } t_{Gesamtreaktion} \\ &= 543ms \\ \rightarrow s_{Anhalten} &= \frac{v_{Start}^2}{5 \frac{m}{s^2}} + 0,543s * v_{Start} \end{aligned}$$

Weiterhin gelten dieselben Regeln wie bei D7 wobei  $s_{Anhalten}$ , anstatt von  $s_{Bremsung}$  zu verwenden ist:

Für rechtzeitige Bremsung bis zur Ampel muss demnach für die Reststrecke zur Ampel  $s_{Rest} \geq s_{Anhalten}$  gelten.

Ist hingegen  $s_{Rest} < s_{Anhalten}$ , also im Grenzwert  $s_{Rest} = s_{Anhalten}$ , muss die Ampel mit der konstanten Geschwindigkeit  $v_{Start}$  in der Zeit  $t_{Überfahren} \leq t_{Gelb}$  überfahren werden können.

Es gilt für

$$2,5 \frac{m}{s^2} : t_{\text{Überfahren}} = \frac{s_{\text{Rest}}}{v_{\text{Start}}}$$

und demnach im Grenzwert:

$$t_{\text{Überfahren}} = \frac{s_{\text{Anhalten}}}{v_{\text{Start}}} = \frac{\frac{v_{\text{Start}}^2}{5 \frac{m}{s^2}} + 0,543s * v_{\text{Start}}}{v_{\text{Start}}} = \frac{v_{\text{Start}}}{5 \frac{m}{s^2}} + 0,543s$$

Als Anforderung an die Gelbzeit lässt sich bei  $2,5 \frac{m}{s^2}$  Verzögerung schlussfolgern:

$$t_{\text{Gelb}} >= \frac{v_{\text{Start}}}{5 \frac{m}{s^2}} + 0,543s$$

Nur so kann die bei Umschaltung von Grün auf Gelb die Möglichkeit der rechtzeitigen Bremsung oder des Überfahrens mit konstanter Geschwindigkeit sichergestellt werden.

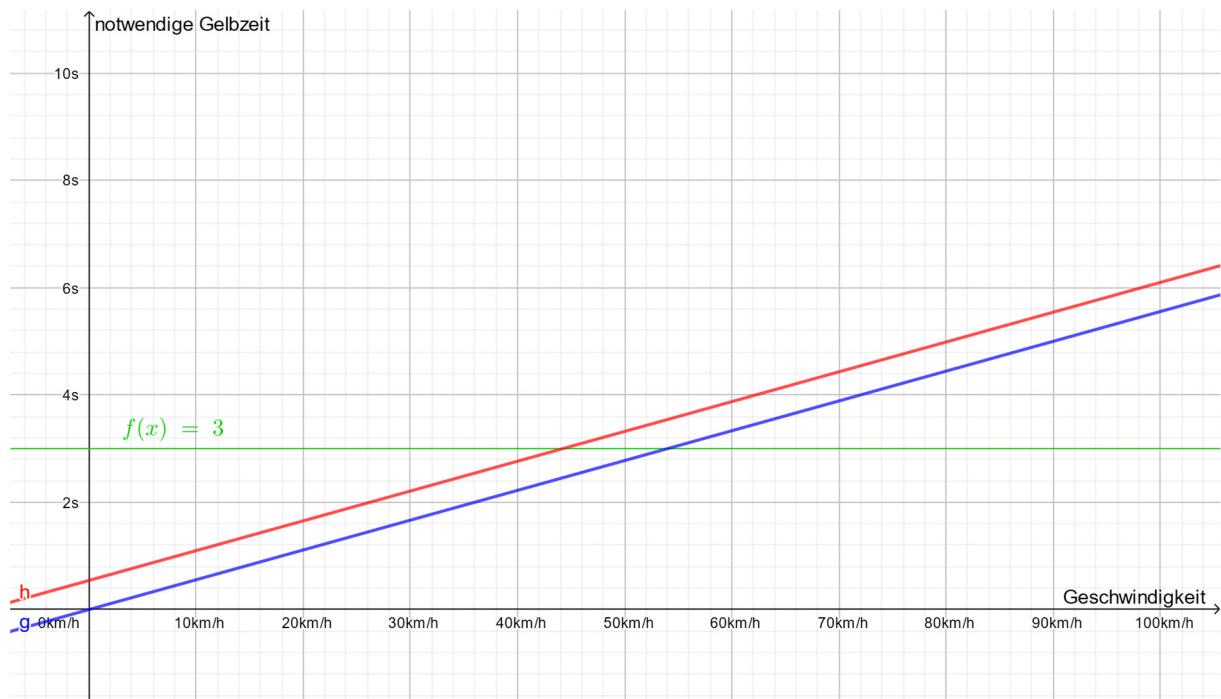


Abbildung 23 Notwendige Gelbzeit mit und ohne Reaktion

Für die konstante Maximalverzögerung von  $2,5 \text{ m/s}^2$  ist die notwendige Gelbzeit über dem Geschwindigkeitslimit / der gefahrenen Geschwindigkeit in Abbildung 23 dargestellt. Der blaue Graph gilt für die Gelbzeit ohne Reaktionszeit, der rote Graph für jene mit Reaktionszeit. Grün ist exemplarisch die gegebene Gelbzeit für 50 km/h dargestellt.

Für die Geschwindigkeiten von 30 km/h, 50 km/h, 70 km/h ergeben sich folgende Gelzeiten als Anforderung, mit und ohne Reaktionszeit:

Geschwindigkeit	Gelbzeit Reaktionszeit	Min.	ohne	Gelbzeit Reaktionszeit	Min.	mit
30 km/h	1,67s			2,20s		
50 km/h	2,78s			3,32s		
70 km/h	3,89s			4,43s		

Bei gegebener Gelbzeit von  $t_{Gelb} = 3s$  für  $v_{Start} = 50 \frac{km}{h}$  müsste somit mit mehr als  $2,5 \frac{m}{s^2}$  verzögert werden, um inklusive Reaktionszeit vor der Ampel anzuhalten.

Die notwendige Verzögerung ergibt sich mit:

$$t_{Gelb} \geq \frac{1}{2} * \frac{v_{Start}}{a_{max}} + \left( 0,18s + 0,29s + \frac{a_{max}}{10,6 \frac{m}{s^2}} * 0,31s \right)$$

Demnach gilt für den Grenzfall bei  $t_{Gelb} = 3s$  und  $v_{Start} = 50 \frac{km}{h}$ :

$$3s = \frac{0,5 * 50 \frac{km}{h}}{a_{max}} + \left( 0,18s + 0,29s + \frac{a_{max}}{10,6 \frac{m}{s^2}} * 0,31s \right)$$

$$2,53 s = \frac{0,5 * 50 \frac{km}{h}}{a_{max}} + \frac{a_{max}}{10,6 \frac{m}{s^2}} * 0,31s$$

$$2,53 s * a_{max} = 25 \frac{km}{h} + \frac{a_{max}^2}{10,6 \frac{m}{s^2}} * 0,31s$$

$$\frac{a_{max}^2}{10,6 \frac{m}{s^2}} * 0,31s - a_{max} * 2,53s + 25 \frac{km}{h} = 0 \mid * \frac{1060 m}{31 s^3}$$

$$a_{max}^2 - \frac{13409}{155} * \frac{m}{s^2} * a_{max} + 237,46 \frac{m^2}{s^4} = 0$$

Nach der  $pq$ -Formel ergibt sich als minimale notwendige Verzögerungsrate:  $a_{max} = 2,838 \frac{m}{s^2}$ .

## D13: Reflexion

### Planung:

Die Planung des Projekts konnten wir aufgrund der vielen kleinen Projekten im Verlauf der Vorlesung schnell abschließen. Das Ziel so wie das Vorgehen war eindeutig und aufgrund der geringen Komplexität und Anzahl an Klassen konnte die Projekthierarchie und Struktur schnell definiert und erstellt werden.

Nach der Definition und Erstellung der Projektumgebung und Struktur wurden zuerst die Funktionalitäten auf die unterschiedlichen Klassen verteilt. Im Anschluss wurde eine vorläufige UnitTest-Implementierung mit festgelegten Variablennamen definiert. Diese musste leider im Verlauf der Arbeit stärker überarbeitet werden (vgl. Abschnitt Tests). Im Anschluss wurde die Funktionalität in mehreren Iterationen implementiert. Zuletzt wurde die Dokumentation basierend auf parallel erstellten Notizen ausgearbeitet. Die Erstellung der Dokumentation nahm hierbei einen Großteil des Arbeitsaufwandes in Kauf und wurde bei der Planung durch fehlende Erfahrung zudem zeitlich falsch eingeschätzt.

Die Implementierung der Funktionalität wurde im Verlauf der Projektarbeit zusammen als Gruppe erstellt, wohingegen die Umsetzung der Unit-Tests, die theoretische Bearbeitung von Zusatzaufgaben, sowie die Erstellung der Experimentierumgebung in der Gruppe aufgeteilt wurden.

Die Gruppenaufteilung hat dabei sehr gut funktioniert und durch paralleles Arbeiten konnte im Vergleich zur Implementierung der Funktionalität deutlich an Zeit eingespart werden.

### **Modellierung:**

Das konkrete Vorgehen beim Modellieren der Aufgabenstellung wurde bereits in Kapitel 2 näher erläutert. Zusammenfassend lässt sich sagen, dass bei der Modellierung der Schwerpunkt auf Vollständigkeit und mathematische und physikalische Gesamtheit und Korrektheit gesetzt wurde. Hierfür wurden sowohl für die theoretischen als auch praktischen Aufgaben realistische Annahmen und mathematische Beweise geführt, die die Richtigkeit und Vollständigkeit des Gedankenguts in vollem Umfang bestätigen sollen.

Für die Aufteilung der Klassen wurde eine realistische Kapselung der Funktionalitäten verschiedener Objekte (Ampel, Fahrzeug, Flashlight, Driver...) und deren Kommunikation untereinander berücksichtigt. So wird zum Beispiel der aktuelle Ampelzustand auch nur über eine Nachricht an den Driver gesendet, wenn dieser in Sichtweite ist und nicht im Driver der Abstand evaluiert.

### **Anforderungen:**

Die gestellten Aufgaben und Anforderungen mussten im Verlauf der Projektarbeit, aufgrund ihrer knappen Formulierung, in der Gruppe des Öfteren diskutiert werden. Aufgrund unterschiedlicher Interpretation der Aufgabenstellung mussten hierbei meist mehrere Lösungsansätze erarbeitet und bewiesen werden. Des Weiteren wurde eine große Zahl an Annahmen getroffen, die im Kontext der Aufgabe realistisch erschienen, jedoch keinen, falls trivial waren.

### **Funktionen:**

Die vorgeschriebenen Funktionen wurden im vollen Umfang umgesetzt. Bei der Umsetzung wurde jedoch die eigentliche Funktionalität teilweise erweitert und zum Beispiel eine adaptive Bremsregelung implementiert, anstatt mit einem konstanten Bremswert zu bremsen. Einzelheiten zu den unterschiedlichen Implementierungen und Funktionserweiterungen werden in den jeweiligen Kapiteln näher aufgearbeitet.

### **Lösungsansatz:**

Für die Ausarbeitung und Implementierung des Lösungsansatzes wurden mehrere Entwicklungsiterationen durchlaufen. Hierbei wurde in erster Linie auf Funktionalität, Simplizität und Vollständigkeit gesetzt. In späteren Iterationen wurden dabei Implementierungen weiter ausgebaut, die Komplexitätsdichte der einzelnen Module leicht erhöht, die Funktionalität perfektioniert und die Kommunikation zwischen den Modulen vereinfacht. In einigen kleinen Fällen mussten hierbei auch mehrere Blockdiagramme überarbeitet und ausgebessert werden. Die zu Anfang geplante Struktur und Hierarchie wurde jedoch über die gesamte Projektlaufzeit aufrechterhalten und nur modulinterne Implementierungsentscheidungen mussten refactort werden.

### Tests:

Die Umsetzung der Unit-Tests ließ sich durch die früh festgelegte Projektstruktur im Vorhinein implementieren. Im Verlauf der Implementierung der eigentlichen Funktionalität mussten hierbei jedoch sehr große Änderungen durchgeführt werden, da die zu testenden Klassen eine funktionale Abhängigkeit von der „Systemlib“ aufwiesen. Ein Großteil der Unit-Tests wurde aufgrund dessen im Nachhinein noch einmal nach implementiert.

### Persönliches Empfinden Grafische Programmierung mit ASCET:

Der grundsätzliche Gedanke des grafischen Programmierens ist es unserer Meinung nach Funktionalitätsimplementierungen durch Nicht-Informatiker zu ermöglichen sowie eine anschauliche Modellierbarkeit für Regelungssysteme im Zustandsraum zu bieten. Aufgrund unserer Zuneigung zu textbasiertem Code für prozedurale Programmierung und der Möglichkeit Funktionalitäten und Bedingungen unabhängig von der Verfügbarkeit grafischer Bausteine zu erstellen, lässt sich die Erfahrung mit ASCET als eher eingeschränkt beschreiben. Durch die Implementierungen von Funktionalitäten durch Blockschaltbilder ist die Verwendung von klassischen Versionierungstools wie zum Beispiel „git“ erschwert. Aufgrund dessen wurde im Verlauf der Projektarbeit der aktuelle Stand immer wieder als Archive-file exportiert und separat gespeichert. Die Schwierigkeit des Debuggens und das anspruchsvolle Hilfe-Menü erschweren zudem die Problemlösung bei Fehlern. Durch die proprietäre Verwendung von ASCET besteht zudem keine öffentliche User-Base, die bei Problemen weiterhelfen kann. Als positiv empfunden wurde der gute Code-Generator mit leicht verständlichem Code, die Übersichtlichkeit bei Regelkreisen und vor allem die Experiment Environment zum anschaulichen Testen des Programms. Dennoch überwiegen unserer Meinung nach eher die Nachteile von ASCET für ein solches Projekt. Dem muss jedoch hinzugefügt werden, dass durch die tägliche Arbeit mit Programm-Code auch ein gewisses Grundverständnis bezüglich textbasierter Entwicklung besteht und die Arbeit mit Blockschaltbildern eher als neue Herausforderung empfunden wurde. In großen Projekten sowie Projekten anderer Ausprägung, an denen nicht nur Softwareentwickler arbeiten, ist die grafische Entwicklung jedoch eine gute Möglichkeit Fachkenntnisse einfach in Software umzusetzen.

## Quellen

- (1) <https://www.bussgeldkatalog.org/gelbe-ampel/>
- (2) <https://www.bosch-mobility-solutions.com/de/loesungen/vernetzung/v2x-vernetzungsloesungen-lkw/>
- (3) <https://auto-talks.com/can-v2x-communication-range-be-too-long/>
- (4) <https://unfallanalyse.hamburg/index.php/ifu-lexikon/bremsen/definitionen/>
- (5) <https://www.bussgeldkatalog.org/reaktionszeit/>