

**Graphische Programmierung &  
Simulation**

**Programmentwurf ASCET**

Nick Dressler (6870655)

Ruben Hartenstein (2746235)

DHBW-Stuttgart Fakultät Technik

Graphische Programmierung

& Simulation im 6. Semester

Dozent: Kai Pinnow

TINF-19ITA 19.04.2022

# TrafficLight Module (R1, R2, D3)

Um die Ampeln zu realisieren, wurde eine *TrafficLight*-Klasse erstellt. Diese Beinhaltet den Zustand ihrer Lichter, sowie ihre Position und die Entfernung zum Auto. Über die *deltaT*-Message, wird eine interne Sekundenanzeige aktualisiert über welche mithilfe von einer Statemachine (siehe Abbildung 1) die Ampelphasen ermittelt werden und.

Diagram

Description automatically generated

Abbildung : Light Statemachine

Ein zentraler *TrafficLightController* hält 3 Instanzen der *TrafficLight*-Klasse, initialisiert diese mit ihrer entsprechenden Position (300.0 [m], 500.0 [m], 900.0 [m]) und updated ihre Licht-Zustände sowie die Position des Autos auf der Runde. Immer wenn die Position des Autos geupdated wird, wird die neue *proximity* der Ampeln berechnet und der Parameter *isVisible* aktualisiert. Sobald die *proximity* zwischen 0 m – 100 m liegt, ist *isVisible* true, andernfalls false.

Ist eine der Ampeln sichtbar (Im Modell sowie in der Welt immer nur eine Ampel realisistisch) ist es dem *driver* möglich auf die *proximity* zuzugreifen und die Ampelphasen auszulesen.

# Car (R3, R1)

Das Auto startet bei Sekunde 0 bei 0 Meter. Mittels des *drivers* wird verhindert, das über eine rote Ampel gefahren wird.

Ist das Auto über die 1000 m Marke gefahren, wird es auf 0 m zurückgesetzt.

# FlashLight (R4)

Sobald ein Auto über eine Ampel mit rotem Licht (*redState*, *yellowRedState*) fährt wird ein Boolean impulsartig gesetzt (Im Experiment Environment als LED dargestellt). Als „drüberfahren“ gilt dabei eine Entfernung vom Auto zur nächsten Ampel von [-2.0[m], 0.0[m]].

# Driver (R5, D2)

Der *driver* beschleunigt das Auto mittels ACC auf 50 km/h. Die Bremskraft liegt dabei bei 70, damit liegt die Verzögerung laut *BrakeMomentum* bei genau -2.5 [a]. und noch innerhalb der Anforderung.

Um Bei einer nicht grünen Ampel passend zu bremsen, muss zunächst unterschieden werden, ab wann man Bremsen muss, beziehungsweise, bis wann man noch drüberfahren sollte.

Für eine obere Abschätzung wurde der Grenzfall von -2.5 Beschleunigung betrachtet. Bei einer Geschwindigkeit von 50 km/h ergab sich dabei ein Bremsweg von 38.59 m.

Als untere Abschätzung wurde die Geschwindigkeit von 50 km/h und die Dauer der Gelbphase betrachtet. Dabei gab sich innerhalb der 3 Sekunden von Grün bevor Rot eine Strecke von 41.67 m.

Daraus lässt sich schließen, dass man beim Ampelphasenwechsel von Grün auf Gelb bei einer ungefähren Distanz von 40 m eine vollständige Verzögerung bis zum Stillstand durchführen sollte. Bei dieser Distanz und Geschwindigkeit lässt sich das, wie man Anhand der Berechnung sieht, mit einer Beschleunigung von -2.5 durchführen.

Im Gegenschluss dazu sollte man bei einer Distanz < 40 m beim Ampelphasenwechsel von Grün auf Gelb weiterhin mit 50 km/h fahren und somit vor der Rotphase hinter der Ampel sein.

Aus diesen Erkenntnissen wurde daraufhin eine Entscheidungsmatrix (siehe Tabelle 1) erstellt. Die Matrix unterteilt sich in die Unterschiedlichen Ampelzustände sowie die unterschiedlichen, ausschlaggebenden Abstände des Autos zur nächsten Ampel.



Tabelle : Entscheidungsmatrix Driver

Aus der Entscheidungsmatrix lassen sich 2 Zustände (50km/h, Break until Green) ablesen. Für den Fall, dass die Ampel mehr als 100 m (für das Auto nicht sichtbar) entfernt ist, fährt der *driver* mittels ACC 50 km/h. Sobald sich eine Ampel im sichtbaren Bereich befindet, kann unterschieden werden, ob die Ampel Grün oder nicht Grün ist. Im Grünen Zustand fährt der *driver* ungestört weiter.

Falls sich die Ampel im Bereich von 40 m bis 100 m befindet, sollte der *driver* das Auto abbremsen, um ein Rotlichtverstoß zu vermeiden und erst bei Grün weiterfahren. Schlägt die Ampel um, während das Auto 0 m bis 40m entfernt ist, sollte das Auto weiterfahren, da es nicht mehr rechtzeitig angenehm  
( -2.5 ) vor der Ampel anhalten kann, es vor der Rotphase rechtzeitig aber noch hinter die Ampel schaffen kann.

Die 2 geklammerten Zustände sind nicht erreichbar, da im Bereich von 40 m bis 100 m, sobald die Ampel nicht mehr Grün ist, abgebremst wird. Dadurch lässt sich die gesamte Entscheidungsmatrix in 2 Zustände (█, █) aufteilen, die mittels der *proximity* und des Wertes *green* unterschieden werden.

In Abbildung 2 lässt sich die daraus folgende Statemachine sehen.

Table

Description automatically generated

Abbildung : Driving Statemachine

Dabei wird *cruising* entweder gesetzt oder nicht gesetzt. Ist *cruising* true, so regelt das ACC das Auto auf 50km/h. Andernfalls wird abgebremst, um vor der Ampel stehen zu bleiben.

Um nicht unnötig weit entfernt, sondern vor der Ampel stehen zu bleiben soll zum Zeitpunkt des *EdgeFalling*s von *cruising* ein *requiredBrake* so berechnet werden, dass das Auto unmittelbar vor der Ampel zum Stehen kommt.

Für die *requiredBrake* wurde zunächst die benötigte negative Beschleunigung für die gegebene Geschwindigkeit von 50 km/h und dem auszustehenden Bremsweg, welche der *proximity* entspricht, berechnet.

Für das Kinematik Modell „Konstante Verzögerung mit Anfangsgeschwindigkeit“ gilt für den Bremsweg , damit berechnet sich die konstante negativen Beschleunigung zu: .

Um die berechnete Beschleunigung in auf den benötigten Wert zu mappen, wurde die vorhandene Tabelle *BrakeMomentum* invertiert und als *InverseBrakeMomentum* im *driver* implementiert. Dafür wurde ein neuer [**type curve\_a\_real is table a -> real;**](https://github.com/inf19150/ASCET_Projekt/blob/5b7e8cc43c2a38b0ccdf74cbd11a694e4883b1d1/workspace_eclipse/StopRollGo/resources/CharTableTypes.esdl#L4)definiert.

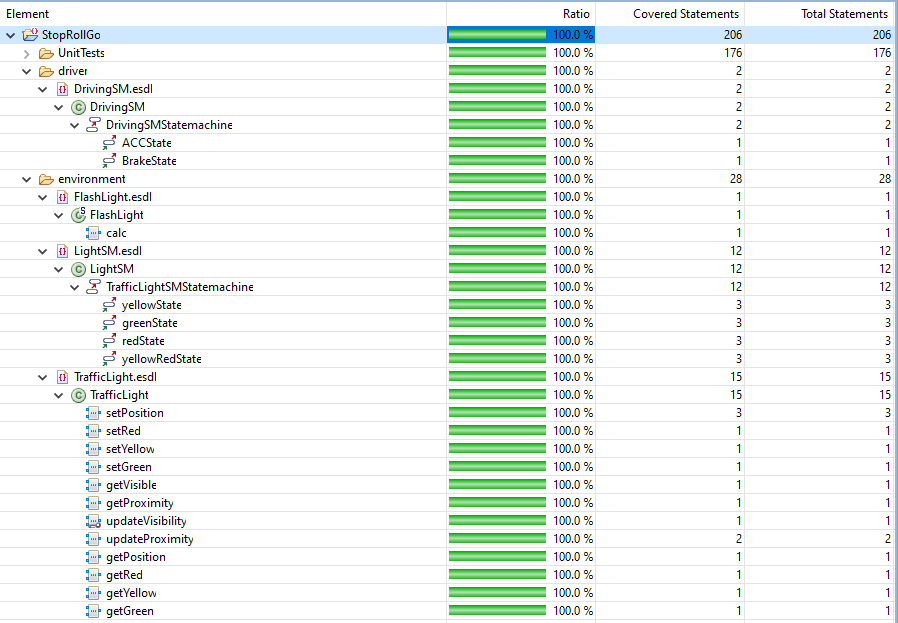
Solange *cruising* false ist, wird *power* auf 0 und *brake* auf *requiredBrake* gesetzt. Das Auto kommt damit ca. 6 m vor der Ampel zum Stehen. Sobald die Ampel grün ist, wird wieder beschleunigt.

# Flashlight and automated Driver (D4)

Siehe: Car (R3), FlashLight (R4) und Driver (R5, D2).

# Unit Tests (D5)

Es wurden zu jeder Klasse entsprechende UnitTests geschrieben. Bis auf 3 Ausnahmen wurde bei den Tests eine Code Coverage von 100% erreicht.

**Graphical user interface

Description automatically generated with low confidence**

Abbildung : Code Coverage fachliche Klassen

Abbildung : Übersicht UnitTests

**Ausnahmen:**

Light  
Da es sich dabei um eine statische Klasse handelt, und ihre zu testende Funktion @Thread annotiert ist kann sie nur vom Scheduler aufgerufen werden und nicht einfach mittels UnitTest getestet werden.

Da der Großteil der Logik von UnitTest\_Light sich in einer Statemachine befindet welche bereits separat getestet wird, wurde die restlich Logik des Sekundenzählers in eine eigene Funktion innerhalb des UnitTests kopiert, um sie dort zu testen.

driver  
Da es sich dabei um eine statische Klasse handelt, und ihre zu testende Funktion @Thread annotiert ist kann sie nur vom Scheduler aufgerufen werden und nicht einfach mittels UnitTest getestet werden.

Ein Teil der Logik von UnitTest\_driver befindet sich in einer Statemachine welche bereits separat getestet wird. Eine weitere Funktionalität, *EdgeFalling*, wurde aus der *SystemLib* importiert. Dabei kann davon ausgegangen werden, dass diese hinreichend getestet wurde. Den Rest der Logik, wie das ACC und die Berechnung des *requiredBrake* Wertes wurde wieder in eine eigene Funktion innerhalb des UnitTests kopiert, um sie dort zu testen.

TrafficLightController  
Beim *TrafficLightController* handelt es sich ebenfalls um eine mit @Thread annotierte, statische Klasse.

Der Controller ist nur dafür verantwortlich die Instanzen der *TrafficLight*szu halten und Informationen zu delegieren. Jede Funktionalität, die im *TrafficLightController* beinhaltet ist, wurde bereits separat getestet. *TrafficLightController* benötigt demnach keine eigenen Tests mehr.

# Experiment Environment (D6)

Im Experiment Environment befindet sich eine übersichtliche Darstellung aller relevanten Informationen (siehe Abbildung 5).

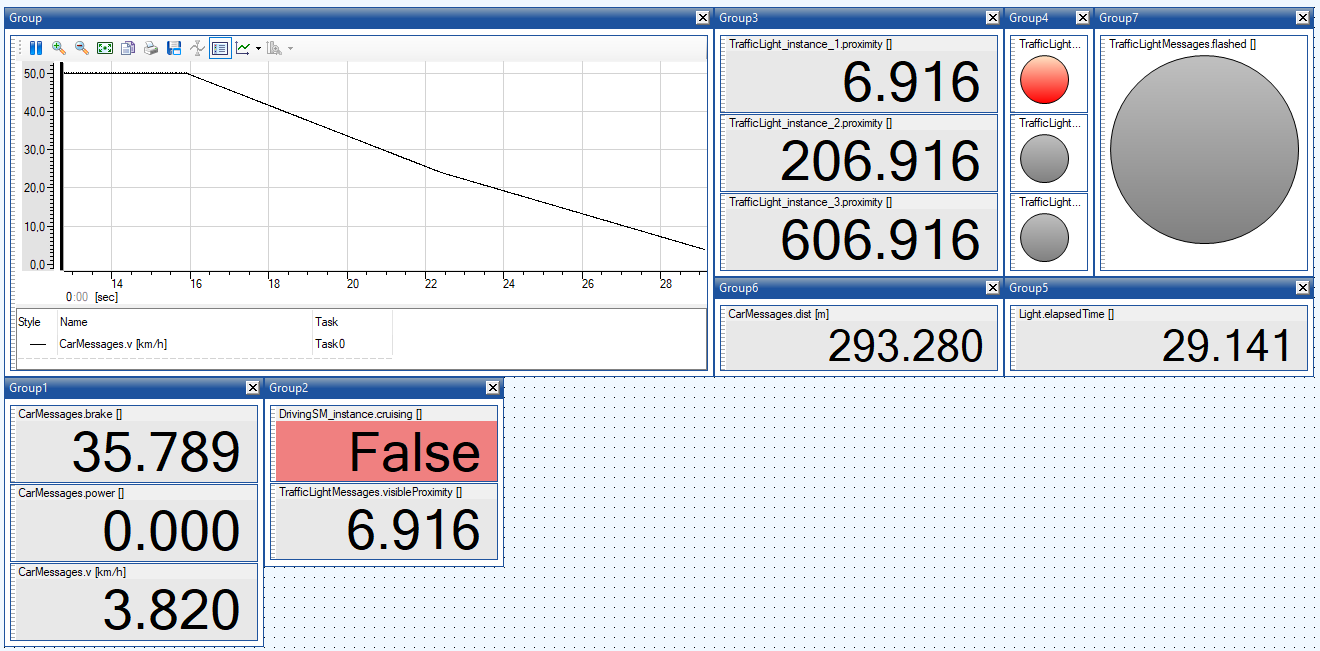


Abbildung : Experiment Environment bezüglich des Systemtests

Dargestellt sind:

* CarMessages.v: Geschwindigkeit des Autos, als Oszilloskop und Edit Box
* CarMessages.brake: Bremswert des Autos, als Edit Box
* CarMessages.power: Leistung des Autos, als Edit Box
* DrivingSM\_instance\_cruising: Zustand des Autos (ACC oder Ampel abhängig), als Edit Box
* TrafficLightMessages.visibleProximity: Abstand des Autos zu einer sichtbaren Ampel, als Edit Box
* TrafficLight\_instance\_X.proximity: Abstand des Autos zu der entsprechenden Ampel, als Edit Box
* TrafficLightMessages.red/yellow/green: Zustand des Ampellichts, als LED in entsprechender Farbe
* TrafficLightMessages.flashed: Signal für Blitzlicht bei Rotlichtverstoß, als LED
* CarMessages.dist: Entfernung auf der Strecke [0m, 1000m], als Edit Box
* Light.elsapsedTime: Vergangene Zeit deltaT %60, als Edit Box