DHBW Stuttgart
Duale Hochschule
Baden-Württemberg

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 12.04.2022

I. 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 einer Statemachine (siehe Abbildung 1) die Ampelphasen ermittelt werden.

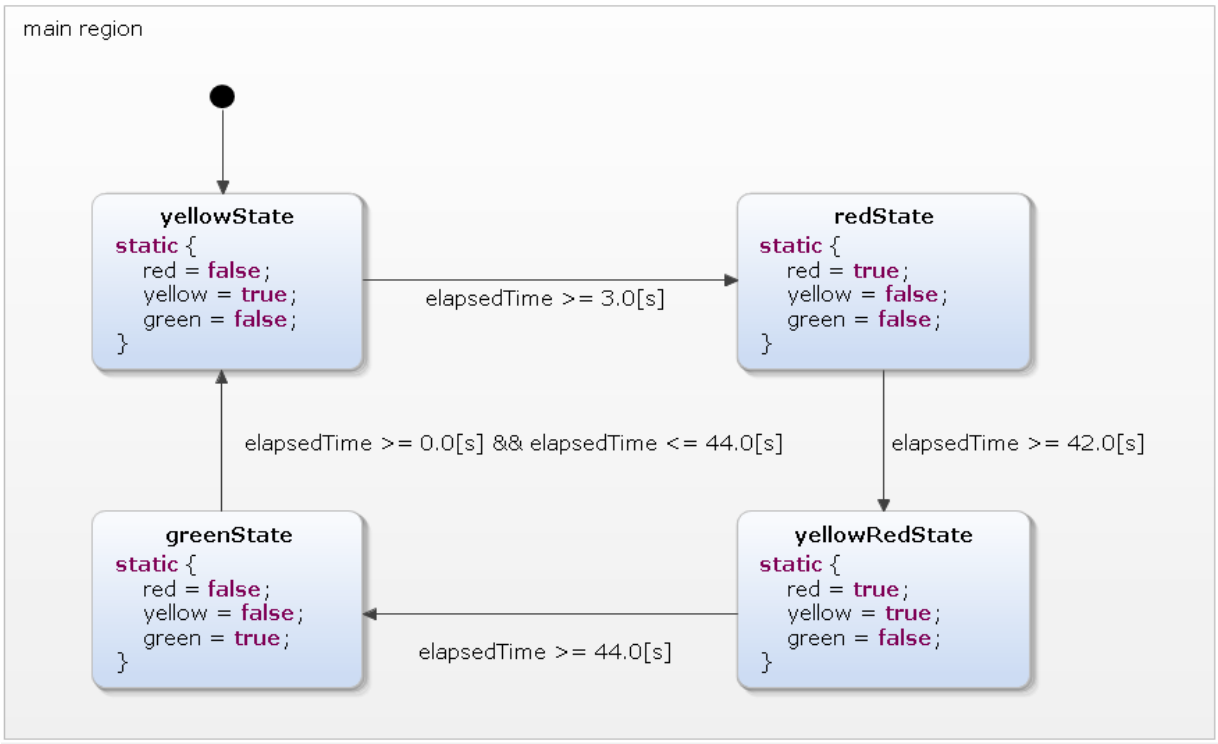


Abbildung 1: Light Statemachine

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

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

II. Car (R3, R1)

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

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

III. 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 innerhalb des Intervalls von [-2.0[m], 0.0[m]].

IV. Driver (R5, D2)

Der *driver* beschleunigt das Auto mittels ACC auf 50km/h. Die Bremskraft liegt dabei bei 70, damit liegt die Verzögerung laut *BrakeMomentum*-Kennlinie bei genau -2.5 [a] und damit noch innerhalb der gegebenen Anforderung R5.

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$\frac{m}{s^2}$ Beschleunigung betrachtet. Bei einer Geschwindigkeit von 50km/h ergab sich dabei ein Bremsweg von 38,59m. Als untere Abschätzung wurde die Geschwindigkeit von 50km/h und die Dauer der Gelbphase betrachtet. Dabei gab sich innerhalb der 3 Sekunden von Grün bevor Rot eine Strecke von 41,67m. (siehe Abbildung 6)

Daraus lässt sich schließen, dass man beim Ampelphasenwechsel von Grün auf Gelb bei einer ungefähren Distanz von $\geq$ 40m eine vollständige Verzögerung bis zum Stillstand durchführen kann und diese auch durchführen sollte. Bei dieser Distanz und Geschwindigkeit lässt sich dies, wie man anhand der Berechnung sieht, mit einer Beschleunigung von $>$ -2.5$\frac{m}{s^2}$ durchführen.

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

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

			Proximity	
		> 100m	40m - 100m	0m - 40m
	Yellow	50km/h	Break until Green	50km/h
Traffic Light State	Red	50km/h	Break until Green	(Break until Green)
	YellowRed	50km/h	Break until Green	(Break until Green)
	Green	50km/h	50km/h	50km/h

Tabelle 1: Entscheidungsmatrix Driver

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

Falls sich die Ampel im Bereich von 40m bis 100m befindet, soll der *driver* das Auto abbremsen, um ein Rotlichtverstoß zu vermeiden und erst bei Grün weiterfahren. Schlägt die Ampel um, während das Auto 0m bis 40m entfernt ist, soll das Auto weiterfahren, da es nicht mehr rechtzeitig angenehm ($>$ -2.5 $\frac{m}{s^2}$) vor der Ampel anhalten kann, es vor der Rotphase rechtzeitig aber noch hinter die Ampel schaffen kann.

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

In Abbildung 2 lässt sich die daraus folgende Statemachine ableiten.

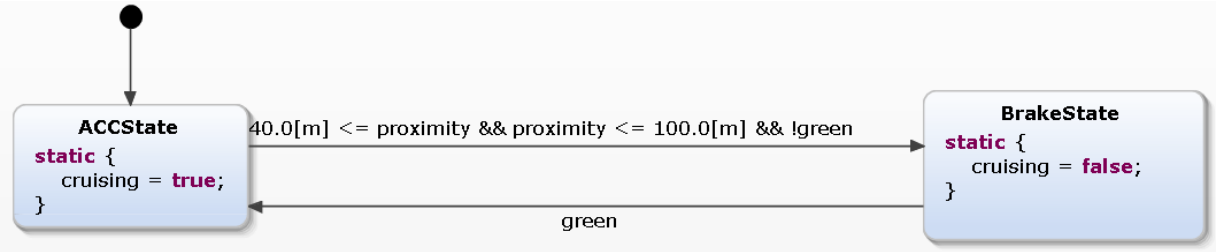


Abbildung 2: Driving Statemachine

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

Um nicht unnötig weit entfernt, sondern vor der Ampel stehenzubleiben, soll zum Zeitpunkt des *EdgeFalling*s von *cruising* ein *requiredBrake* passend genauso 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 verbleibenden Bremsweg, welche der *proximity* entspricht, berechnet.

Für das Kinematik Modell „Konstante Verzögerung mit Anfangsgeschwindigkeit“ gilt für den Bremsweg $s_{Br} = \frac{v^2}{2 \cdot a}$, damit berechnet sich die konstante negative Beschleunigung zu: $a = \frac{v^2}{2 \cdot s_{Br}}$.

Um die berechnete Beschleunigung in $\frac{m}{s^2}$ auf den benötigten Wert zu mappen, wurde die vorhandene Tabelle *BrakeMomentum* invertiert und als *InverseBrakeMomentum* im *driver* implementiert. Dafür wurde ein neuer Datentyp **type curve_a_real is table a -> real;** definiert.

Solange *cruising* false ist, wird *power* auf Null 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.

V. Flashlight and automated Driver (D4)

Siehe: Car (R3, R1), FlashLight (R4) und Driver (R5, D2).

VI. Unit Tests (D5)

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

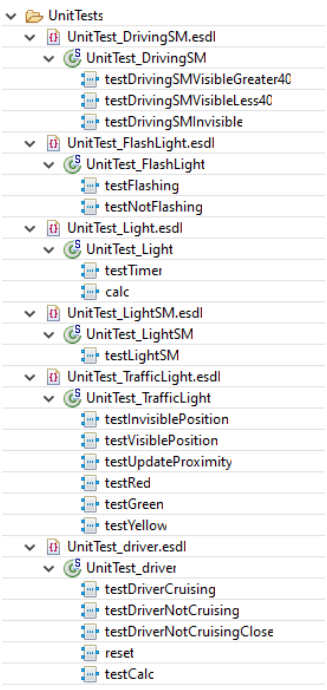
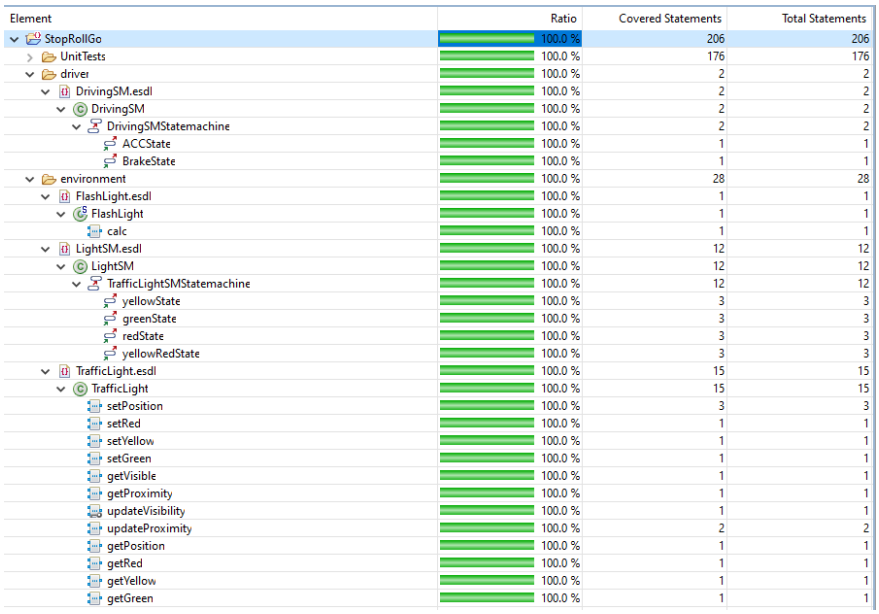


Abbildung 4: Übersicht UnitTests



Element	Ratio	Covered Statements	Total Statements
StopRollGo	100.0 %	206	206
UnitTests	100.0 %	176	176
driver	100.0 %	2	2
DrivingSM.esdl	100.0 %	2	2
DrivingSM	100.0 %	2	2
DrivingSMStatemachine	100.0 %	2	2
ACCState	100.0 %	1	1
BrakeState	100.0 %	1	1
environment	100.0 %	28	28
FlashLight.esdl	100.0 %	1	1
FlashLight	100.0 %	1	1
calc	100.0 %	1	1
LightSM.esdl	100.0 %	12	12
LightSM	100.0 %	12	12
TrafficLightSMStatemachine	100.0 %	12	12
yellowState	100.0 %	3	3
greenState	100.0 %	3	3
redState	100.0 %	3	3
yellowRedState	100.0 %	3	3
TrafficLight.esdl	100.0 %	15	15
TrafficLight	100.0 %	15	15
setPosition	100.0 %	3	3
setRed	100.0 %	1	1
setYellow	100.0 %	1	1
setGreen	100.0 %	1	1
getVisible	100.0 %	1	1
getProximity	100.0 %	1	1
updateVisibility	100.0 %	1	1
updateProximity	100.0 %	2	2
getPosition	100.0 %	1	1
getRed	100.0 %	1	1
getYellow	100.0 %	1	1
getGreen	100.0 %	1	1

Abbildung 3: Code Coverage fachliche Klassen

Ausnahmen

Light

Da es sich dabei um eine statische Klasse handelt, und ihre zu testende Funktion @Thread annotiert ist, kann diese 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 diese 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 ASCET-*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*s zu halten und Informationen zu delegieren. Jede Funktionalität, die im *TrafficLightController* beinhaltet ist, wurde bereits separat getestet. *TrafficLightController* benötigt demnach keine eigenen Tests mehr.

VII. Experiment Environment (D6)

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

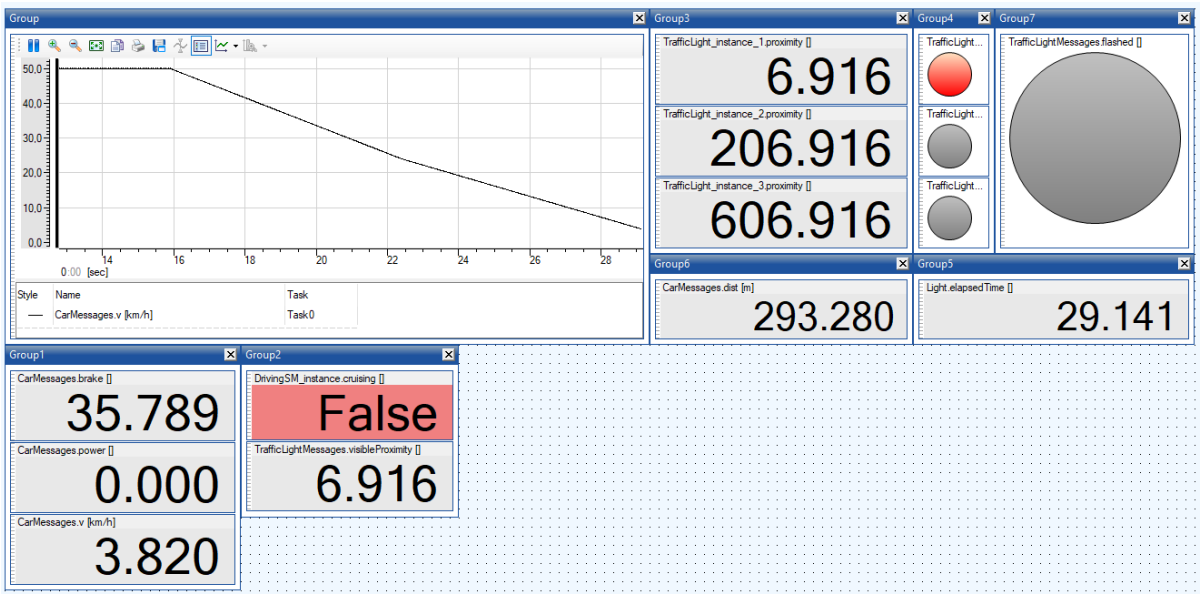


Abbildung 5: Experiment Environment bezüglich des Systemtests

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 modulo 60, als Edit Box

VIII. Estimation Excel Sheet (D11)

Aufgaben [min]	Actual time	Best Case	Most likely	Worst case	<T>	var
Devops einrichten	38	5	10	25	11,67	11,11
Aufgabe verstehen & erklären	8	5	10	20	10,83	6,25
Aufgabenplan & Aufgabenabschätzung erstellen	18	10	12	20	13,00	2,78
Durchführbarkeit theoretisch evaluieren	29	20	30	45	30,83	17,36
TrafficLight Modul implementieren	131	60	100	150	101,67	225,00
Flashlight Modul implementieren	48	15	35	45	33,33	25,00
Modulintegration	72	30	45	100	51,67	136,11
unit tests implementieren	116	90	150	60	125,00	25,00
Experiement designen & aufsetzen	30	10	15	30	16,67	11,11
Diskussion gelbphasentiming (*)	43	30	45	60	45,00	25,00
Disukssion V2X (*)	18	15	25	40	25,83	17,36
Dokumentation PDF	172	100	120	160	123,33	100,00
Human-Factor impact (*)	13	10	15	20	15,00	2,78
Reflect (*)	21	15	30	45	30,00	25,00
Total time	757				633,83	629,86
				2-Sigma Range	583,64	684,03

IX. Zusatzaufgaben

Wie sollte das Timing bzgl. der Gelbphase für unterschiedliche Geschwindigkeitsbeschränkungen gewählt werden? (D7*)

Da die Sichtweite zur Ampel konstant bleibt muss, um dem *driver* die Möglichkeit zu geben zu reagieren, bei zunehmender Geschwindigkeit die Dauer der Gelbphase ebenfalls verlängert werden. Bei unserem momentanen Beispiel mit einer Geschwindigkeit 50km/h und einer Gelbphasendauer von 3s, steht dem *driver* ein theoretischer Spielraum von $\Delta s = 3.08m$ zur Verfügung. Diese 3.08m legt er mit seiner Geschwindigkeit in ca. $\Delta t = 0.222s$ zurück. Um dem *driver* bei anderen Geschwindigkeiten die gleiche Reaktionszeit zu gewährleisten, sollte Δs abhängig von der Geschwindigkeit gemacht werden. Mit ausgewähltem Δs, lässt sich dann für die Richtgeschwindigkeit v eine entsprechende Gelbphasendauer berechnen (siehe Abbildung 7).

Wie kann die Verkehrssituation verbessert werden, wenn V2X eingesetzt wird oder R5 flexibler ist? (D9*)

Bei einer zukünftigen Voraussicht bzgl. Ampelphasen durch V2X kann ein optimaler Bremsvorgang eingeleitet werden, um z.B. im Fall von Elektrofahrzeugen eine maximale Energierückgewinnung in den Akkumulator mittels Rekuperation zu ermöglichen. Ebenso kann z.B. an einer Kreuzung von unterschiedlich stark befahrenen Straßen eine Priorisierung vorgenommen werden, da die Ampelanlage über die Anzahl der in Zukunft zu passierenden Fahrzeugen auf der jeweiligen Straße informiert ist. Damit ist es möglich, unnötige oder unnötig lange Rotphasen zu schalten und den Verkehr auf den Hauptverkehrsstraßen mit verhältnismäßig langen Grünphasen zu priorisieren.

Begründung über den Einfluss menschlicher Wahrnehmung bzw. Reaktion oder anderer Verzögerungen (D12*)

Bei einem menschlichen Fahrer muss neben der Bremsdauer auch noch die Reaktionsdauer, also die Zeitspanne zwischen Auslösen eines Ereignisses z.B. Ampel schaltet auf Rot und Eintritt der Fahrzeugverzögerung durch Betätigung des Bremspedales beachtet werden. Diese kann sehr stark variieren. So ist diese bei jungen Fahrern typischerweise geringer und bei älteren Menschen etwas länger. Diese Schwankungsbreite muss berücksichtigt werden. Beispielsweise sollte ein solcher manueller Fahrer von einem vollautonomen System nicht mit einem Referenzwert bzgl. Reaktionszeit, sondern vielmehr mit Worstcase Referenzzeiten angesehen werden.

Reflektion: Welche anderen Beobachtungen oder Kommentare gibt es bezüglich der Planung, der Modellierung, den Anforderungen, den vorgeschriebenen Funktionen oder Ihre Lösung, dem Testen und des gewählten grafischen Ansatzes?

- Keine Konstruktoren in ESDL-Klassen
- Testen von statischen thread-annotierten Methoden nicht einfach möglich. Workaround: Einfügen einer Kopie des Inhalts der entsprechenden Funktion in eine Test-Funktion. Dies führt aber zu Bad (Code-) Smells, da bei jeder Änderung mehrere Stellen im Code angefasst werden müssen.
- Das Einrichten einer Versionsverwaltung mit Git bedarf umfassende Kentnisse bzgl. gitignore-Dateien. Manche Dateiformate von ASCET erzeugen bspw. einen Zeitstempel, obwohl keine inhaltliche bzw. fachliche Änderung am Projekt vorgenommen wurde. In diesem Fall muss die lokale Kopie im Anschluss manuell resettet werden.
- Eine bidirektionale Konvertierung von esdl zu bd wäre schön. Aus Informatiker Sicht ist der Zugang über Code anstelle einer grafischen Modellierung für die Implementierung einer Funktionalität manchmal naheliegender. Hier wäre evtl. der Erhalt des grafischen Layouts nach einer Umwandlung von esdl zu bd wünschenswert.

X. Anhang

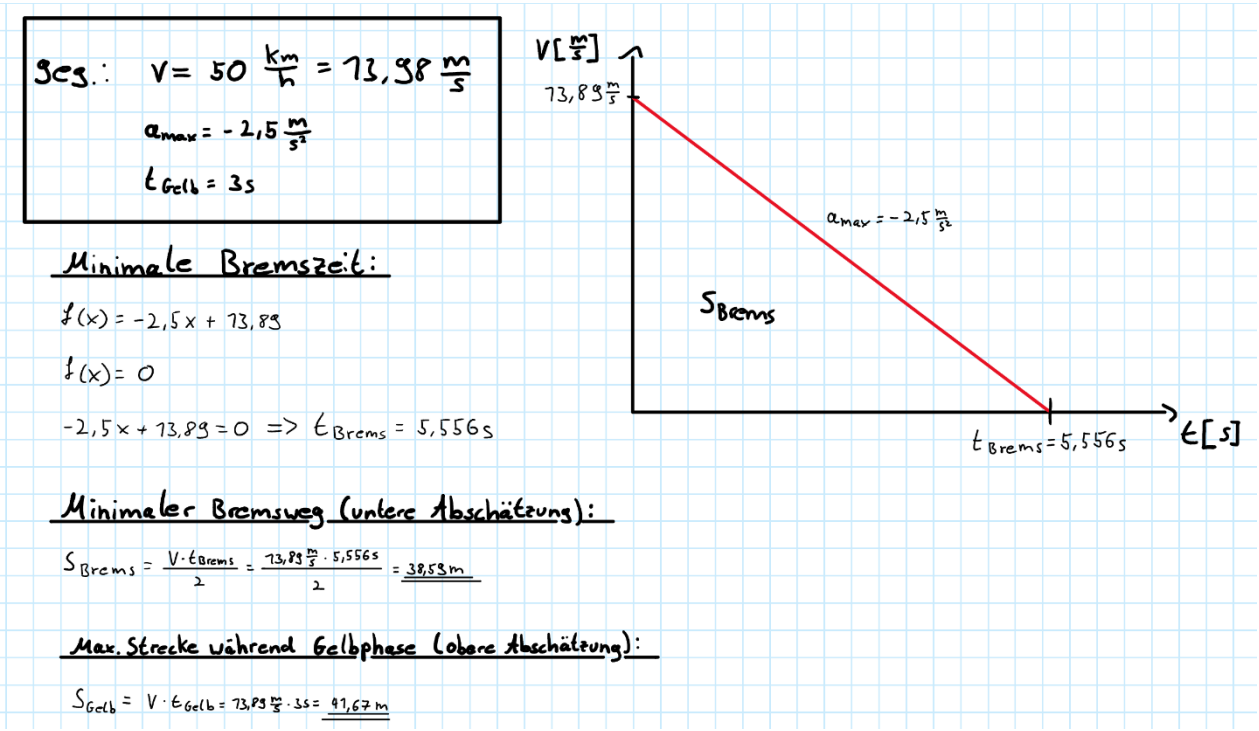


Abbildung 6: Abschätzung bzgl. Machbarkeit Bremsverzögerung

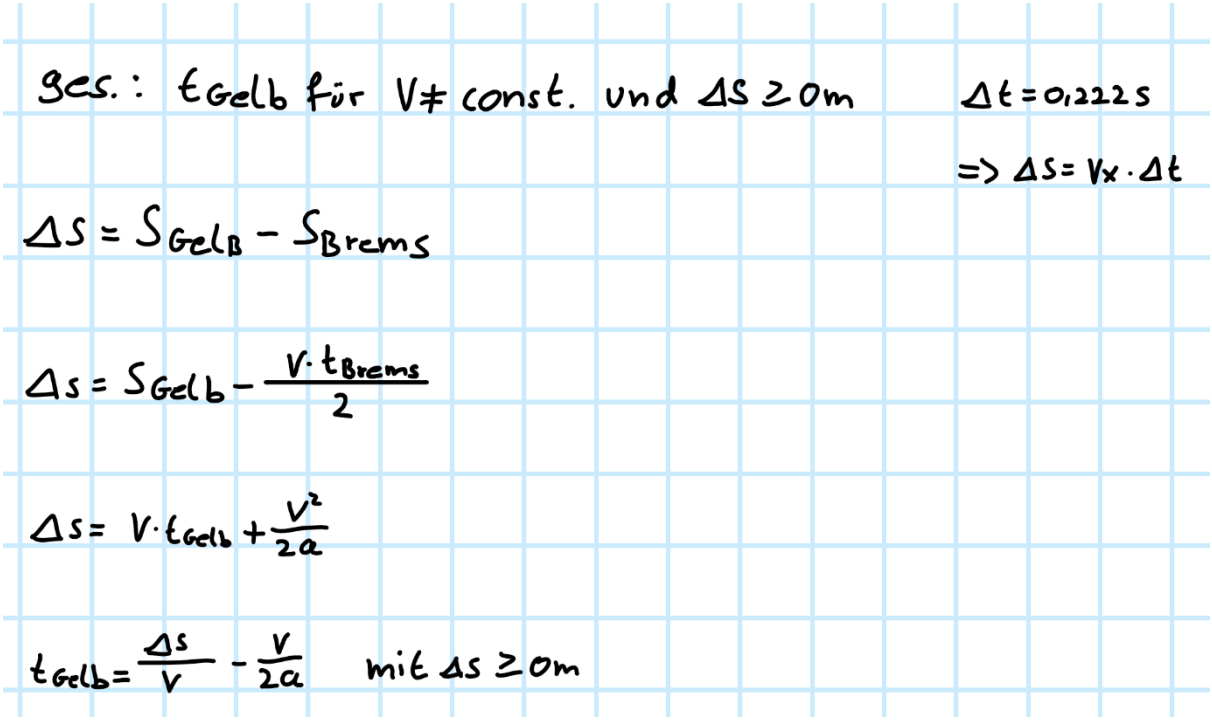


Abbildung 7: Gelbphasendauer für variable Geschwindigkeit