Development of a controller to switch between relative and absolute path for target vehicles in simulation scenarios

Olof Karlsson

Erik Fredin

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6139
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

Simulating vehicle scenarios for autonomous cars saves both time and money compared to physical testing. For this to work efficiently a sophisticated simulation software is often needed. Esmini is an open-source, lightweight vehicle simulator that can handle advanced traffic scenarios. When designing scenarios to be used for testing autonomous vehicles, it is often beneficial to set up relative vehicle behaviour to ensure that a hazardous situation arises. This could be that the car in front of the autonomous car drives a couple of kilometers per hour slower compared to the autonomous car. At some point the autonomous car must react to avoid a collision, e.g., by braking. To avoid having the car in front slowing down when the autonomous car brakes, a switch must be done such that the car in front has an absolute speed.

Hence, the purpose of the controller developed in this thesis is to decide when to switch from the relative setup and what the new control objective should be. This is implemented by simulating a short period into the future to observe how the scenario develops. The controller generates predictions using simulation data of the events that should transpire for every vehicle except the autonomous one. The expected vehicle states are then continuously compared to the actual vehicle states generated during the simulation. If the actual vehicle states deviate too far from the expected values, a switch occurs. A new objective is then given to the vehicle, depending on its behaviour before the switch and what type of relative setup previously existed.

The proposed solution is available in the official Esmini repository and has been tested in traffic scenarios with CSPAS, a platform where Volvo's active safety functions are simulated. The performance of the controller depends on how it is tuned. For all tested scenarios a satisfying result has been achieved.

Acknowledgements

We are deeply grateful for all the help we received and would like to extend our sincere thanks to Emil Knabe, Mikael Andersson and Angel Molina Acosta for their support at Volvo cars. We would would also like to offer a special thanks to Anton Cervin at LTH for the guidance and support we have been given.

Nomenclature

Symbols and Abbreviations

ACC	Adaptive Cruise Control
AD	Autonomous Drive
ASAM	Association for Standardization of Automation and Measuring Systems
API	Application Programming Interface
CMBB	Collision Mitigation By Braking

Terms and Definitions

CSPAS	A platform by Volvo Cars that can run active safety functions or AD software.
Ego Vehicle	The vehicle(s) which is(are) the focus of a scenario, i.e., the vehicle(s) under test. For evaluation of automated driving system, the Ego vehicle will be the one controlled by the system-under-test.
Esmini	Environment Simulator Minimalistic, see Chapter 2.2
Switch	The process where the controller removes the relative kinematic constraint between the target and the ego vehicle, such that the target's desired movement is defined in absolute terms, i.e., with respect to a fixed reference frame.
Target Vehicle	The vehicle(s) used to test collision scenarios with the ego vehicle. The target vehicle(s) is/are not used to test AD software, and instead serve as additional vehicles in traffic scenarios.

Contents

1. Introduction **11**
- 1.1 Overview . . . 11
- 1.2 Problem Description . . . 11
- 1.3 Research Questions . . . 13
- 1.4 Methodology . . . 13
- 1.5 Delimitation . . . 13

2. Background **14**
- 2.1 ASAM OpenSCENARIO . . . 14
- 2.2 Esmini - an OpenSCENARIO player . . . 18
- 2.3 Background on Simulation Model Testing . . . 30

3. Controller Design **34**
- 3.1 Controller Concepts . . . 34
- 3.2 Controller Implementation . . . 36
- 3.3 Integration of the Controller with Esmini . . . 44

4. Controller Validation **46**
- 4.1 Controller Validation Process . . . 46
- 4.2 Results . . . 52

5. Discussion **60**
- 5.1 Controller Concept and Prediction Algorithm . . . 60
- 5.2 Control Objective after the Switch . . . 61
- 5.3 Testing . . . 62

6. Conclusion & Future Work **67**
- 6.1 Conclusion . . . 67
- 6.2 Future Work . . . 68

Bibliography **69**

Appendices **71**
- A OpenSCENARIO xml code . . . 71

1

Introduction

1.1 Overview

The development for Active Safety and Autonomous driving (AD) features in cars is a very high priority for car manufacturers across the world. While active safety features are already included in many cars on today's market, level 5 autonomy road cars are expected to be available in the 2030s [Litman, 2021]. Most likely, the first autonomous vehicles will be available in the form of "ride-shares", as well as autonomous trucks and busses. Volvo Cars currently invests significantly into the research and development of both active safety as well as AD software. Physical testing for these is often expensive, time-consuming and in many cases not feasible at early stages of AD development. A solution to this is to test software using simulations. Volvo makes use of "Esmini" [Knabe, 2021a], an open-source OpenScenario [*ASAM OpenSCENARIO* 2021] player for traffic simulations of autonomous vehicles. This is in order to test AD and active safety software in traffic scenarios that could lead to a collision.

1.2 Problem Description

Simulations in Esmini typically contain an ego (AD) vehicle, and a so-called target vehicle which is used to define a scenario that can lead to a collision (i.e., "collision scenario"). There are two types of controllers that can be used to control the movement of the target vehicle in Esmini. The first is absolute control, where the vehicle is controlled such that its motion complies with a predefined trajectory that is fixed to the road. Relative control on the other hand means that the target vehicle is controlled so that the relative motion between target and ego vehicle satisfies some sort of condition. This could for instance be a fixed difference in distance or speed between the two vehicles.

A traffic scenario is defined in order to test how the ego vehicle reacts to a potential collision, and is significantly easier to define using relative control. However, defining the target vehicle's behaviour using relative control during the entire scenario can lead to unrealistic behaviour. For instance, if the ego vehicle triggers an emergency braking function in order to avoid a potential collision, the target vehicle may also brake and come to a stop, causing both vehicles to wait for each other in order to continue driving. This is unrealistic and undesirable behaviour in such a simulation. In order to avoid this, the controller controlling the target vehicle must switch from relative to absolute control at an appropriate time (see Figure 1.1). This is to ensure that the target vehicle acts independently of the ego vehicle's behaviour after the ego vehicle reacts to a potential collision. The problem is made more challenging by the fact that the ego vehicle is not controlled by Esmini, and is instead controlled by an external software. This means that Esmini does not have access to any data on the future states of the ego vehicle, i.e., any future motion of the ego is unknown to Esmini.

Esmini has always lacked a controller that supports switching from relative to absolute control for the target vehicle, and this is therefore a primary focus of this thesis. The controller's design is primarily focused on key issues such as when the switch should occur and what the control objective should be after the switch. These must be defined such that the controller successfully controls the target vehicle in a realistic fashion during many different types of scenarios. Because of this, experimental verification in order to evaluate the controllers performance is a key element of this Thesis. The switching controller is verified using different collision scenarios in Esmini, where the ego vehicle will be controlled by a software called CSPAS. CSPAS, which stands for Compiled Simulation Platform for Active Safety, is a vehicle simulator used by Volvo Cars which can execute code for autonomous driving functions. Testing the controller while the ego vehicle is controlled using CSPAS is vital, since this ensures that the controller can function even without any prior knowledge of the ego vehicle's behaviour.

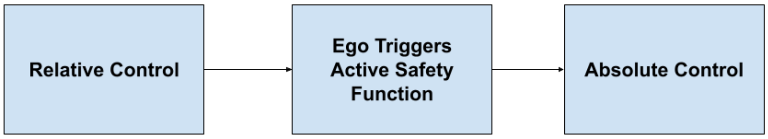


Figure 1.1 Flowchart of when the target vehicle switches from relative to absolute control.

1.3 Research Questions

The following research questions were formulated for this thesis:

- When should the target vehicle switch from relative to absolute control and vice-versa?
- How can standard behaviour by the target vehicle after the collision scenario be ensured?
- What vehicle state variables are to be used for determining when to switch control objectives?

1.4 Methodology

This thesis consists of the following steps:

- Scenario analysis
- Implementation of a simple controller
- Testing without CSPAS
- Evaluation of different controller concepts
- Implementation of the final controller concept
- Testing with CSPAS

In the scenario analysis phase of the thesis, different scenarios are analysed in order to observe and identify undesired behaviour. In addition, Esmini and OpenSCENARIO were thoroughly analysed in order to explore how a controller could be implemented. The next phase includes developing concepts for the controller, and testing these in Esmini using the scenarios from the initial phase. These controller concepts are then evaluated, and one is selected for further development. The controller is then fully developed, while also being tested on the Esmini scenarios. Continuous testing during development will be vital to determine how the controller performs. Once the final controller has been fully developed, it is then tested on various scenarios using CSPAS.

1.5 Delimitation

In this thesis, no algorithms for controlling the ego vehicle are developed. Developing safety functions or autonomous driving functions is out of scope. Hence, the work is solely focused on the switching controller for the target vehicle. Certain elements and/or features already included in Esmini may be modified in order to suit certain control objectives, however the main focus is the switching controller itself.

2 Background

2.1 ASAM OpenSCENARIO

Traffic simulations can be a safe, cheap and more efficient alternative to real-world testing, and have hence gained in popularity over the past years [Figueiredo, 2009]. In addition, they allow for the testing of more traffic scenarios in a shorter period of time, with less risk. Traffic simulators are generally distinguished into two different types. The first type are large scale traffic simulators, where several hundreds of vehicles are simulated in complex traffic scenarios. Due to the high number of vehicles simulated, this type of simulator typically does not simulate sensor data or kinematics/dynamics of individual vehicles. The second type of traffic simulator are small-scale robotic simulators, where most commonly one or very few vehicles are simulated at a time. Robotic simulators however provide much more detailed data on sensors and vehicle dynamics. The traffic simulation standard ASAM Open-SCENARIO provides a compromise between these two types of simulators, as it allows for the simulation of multiple vehicles, while at the same time providing the ability to simulate sensor and vehicle dynamics data for autonomous vehicles.

ASAM OpenSCENARIO is a standard that defines a file format which describes dynamic content for driving and traffic simulators [*ASAM OpenSCENARIO* 2021]. The standard makes use of road network descriptions from ASAM OpenDRIVE and has support for road surface profiles from ASAM OpenCRG. Together they complement each other to describe both static and dynamic content of vehicle simulation applications.

OpenSCENARIO is primarily used to describe complex, synchronized maneuvers that involve multiple entities such as vehicles, pedestrians and other traffic participants. A maneuver can be based on either actions or on trajectories. An example of an action could be a lane change or speed change while a trajectory could be a recorded driving maneuver. However the standard also specifies the description of the ego vehicle, driver appearance, pedestrians, traffic and environment conditions. OpenSCENARIO by itself does not function as a traffic simulator. Instead, software implementing the standard is required to run traffic simulations for autonomous vehicles.

Simplified OpenSCENARIO Structure

OpenSCENARIO features three fundamental and mandatory concepts that implementations of the standard must feature. The first concept is that so-called Road Networks are populated by instances of entity, and interact with these using a set of instructions defined by the storyboard. The second concept is that a scenario storyboard must contain at least one instance of story. Figure 2.1 shows a simplified illustration of the elements that are contained within a story and by extension of a storyboard. The most fundamental of these is a so-called action. An action defines the motion and/or position of an actor (e.g., a car, pedestrian) in Esmini. The third fundamental concept is that actions taken by actors are triggered by so-called conditions. Conditions along with triggers are used to define when actions and acts occur, and are thus the basic building block of OpenSCENARIO. [*ASAM OpenSCENARIO* 2021]

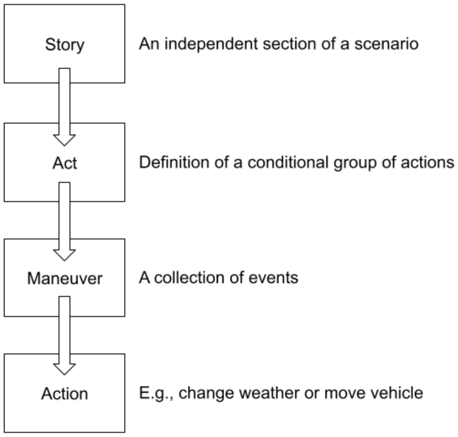

Story
An independent section of a scenario
Act
Definition of a conditional group of actions
Maneuver
A collection of events
Action
E.g., change weather or move vehicle

Figure 2.1 A simplified diagram showing the structure of storyboard.

The storyboard is used to define what happens and when, and encompasses all elements shown in Figure 2.1. The storyboard encompasses two fundamental building blocks, which are in turn made up of so-called storyboard elements. The first building block is Init, which defines initial conditions such as speed and position in the scenario. In Init, the behaviour of actors cannot be defined conditionally or relative to other actors. Story on the other hand encompasses all storyboard elements that occur as the scenario progresses. A storyboard element can either be an act, maneuvergroup, event and action. An act defines when a certain action in a scenario occurs through triggers and conditions, whereas a maneuver defines what is happening in the action. Maneuver groups define what actors take part in the action.

Actions are used in order to define the behaviour of dynamic elements in OpenSCENARIO. There are three types of actions in the OpenSCENARIO standard:

- PrivateActions
- GlobalActions
- UserDefinedActions

A user defined action can be used by user's to create or modify their own action using a script file. Global Actions are used to define non-entity related qualities, such as traffic signal-states, weather conditions or infrastructure. Private Actions on the other hand define qualities related to entities, such as position or speed of vehicles. They can either define lateral or longitudinal behaviour of entities, such as a longitudinal speed change or lateral lane change. Private actions can also define the behaviour of one entity relative to another. For instance the position of a car can be defined such that the car always holds a fixed distance to another car, or other entity.

Since actions are singular elements describing the behaviour of entities, they must be combined in order to create more complex behaviour. Therefore, actions must be contained within events in the story sections of a scenario. However, in the Init section actions can be defined by themselves, without the need for being contained in an event. An event is started using a startTrigger, for which the event has to be in a stand-by state. Due to the fact that each event defined in a scenario corresponds to a single runtime instance, multiple instances of the same event cannot run simultaneously. [*ASAM OpenSCENARIO* 2021]

Catalogs and Controllers

Catalogs are used in order to create and store parameters or other information in OpenSCENARIO. The use of catalogs, which are in turn referenced in scenario files, prevents the user from re-writing the same parameter declarations multiple times. Instead, users can reference catalogs, which contain relevant information for the scenario. If a catalog is referenced by a scenario description, the parameter declarations in the catalog override those in the scenario description where applicable. A wide variety of different OpenSCENARIO elements can be defined in using a respective catalog, for instance vehicle, pedestrian and their controllers. Controllers can be used to control the behaviour of an entity in the longitudinal or lateral domain, and are assigned to either vehicles or pedestrians. They have three intended uses according to the OpenSCENARIO standard, specifying that an entity should be controlled by a system under test, e.g., AD or active safety function; defining an actor that should be more advanced and has more complex behaviour than can be described with actions; or assigning the control of an entity to a human. Controllers can either be part of the simulator implementing OpenSCENARIO, or they can be external. Controllers must be activated in an XOSC file using an activate controller action. It is useful to reference a controller via a controller catalog containing all controllers implemented by a simulator in such an action. The controller catalog in turn references the simulator files implementing the controllers in it.

Scenario Creation

OpenSCENARIO files (.xosc) are written in an XML format. It is worth noting that OpenSCENARIO files by themselves merely provide a set of instructions for sequences of behaviour that take place in the scenario, however they do not execute these scenarios. In order to simulate a scenario, a scenario engine implementing the OpenSCENARIO standard is needed (see Section 2.2). Part of an XOSC file is shown in Appendix A. The Init section in this includes teleport, activate controller and speed actions. The teleport action is used to define the referenced entities' initial position and the longitudinal speed action defines its initial speed. An activate controller action is used in order to assign a specific controller to a certain entity, by referencing the desired controller in the controller catalog. An example of a longitudinal speed action is given below:

```
<Private entityRef="Ego">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsDimension="time" dynamicsShape="
                     step" value="0"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="8.333333333333334"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
```

Listing 2.1 Longitudinal Speed Action example.

This type of action references the entity "ego", and hence defines the behaviour of this entity. In addition, the dynamics dimension and shape can be specified by the action. In this case, these are "time" and "step", respectively, meaning that "ego" will obtain the desired output speed value directly, without any acceleration phase. Other actions, such as distance actions also use similar keywords in order to specify the behaviour of referenced actions. In addition, longitudinal speed actions can be used in both the Init and Story sections of a scenario, and therefore serve as a good example of how an action is implemented in the XOSC format.

In order to ease the creation of scenarios, Volvo Cars has created a Python-based scenario generation framework called scenariogeneration [Andersson, 2021]. Scenariogeneration includes packages for the generation of both XOSC and XODR (describing road networks) files in the XML format. The goal of this is to provide a more user-friendly method for generating OpenSCENARIO files, eliminating the learning curve that can be associated with the XML format.

2.2 Esmini - an OpenSCENARIO player

Esmini is a minimal vehicle scenario simulation software. It can be used to simulate different situations where AD software can be tested. Esmini uses the OpenSCENARIO standard for describing dynamic events and has some support for road networks in OpenDRIVE format.

Esmini was originally developed as a project intended to explore and get familiar with the emerging OpenSCENARIO data format [Knabe, 2021a]. There is a wide support for different platforms which was one of the goals for the project. Tool integration and portability was also high priority in the development. Since its purpose has never been for production use, even though the license allows it, the code quality can be lacking. Furthermore, full coverage of the OpenSCENARIO standard is missing. This is due to the project being developed on demand and being defined by the research scope of the original project.

Esmini Structure

Esmini can be broken down into a few key players. There is the Scenario engine which runs the program, a Gateway that is used for accessing data, a player that provides a high level API for controlling the scenario in a custom player application and a viewer that the renders frames. A frame sequence can be seen in Figure 2.2. Here we can see that controllers are handled separately from the scenario engine and more importantly they are executed after the default controllers and actions. The importance of this will be shown later. We can note that an external app calls the player which executes a timestep of the simulation. The external app would be any AD or active safety function software that is to be tested in the simulation.

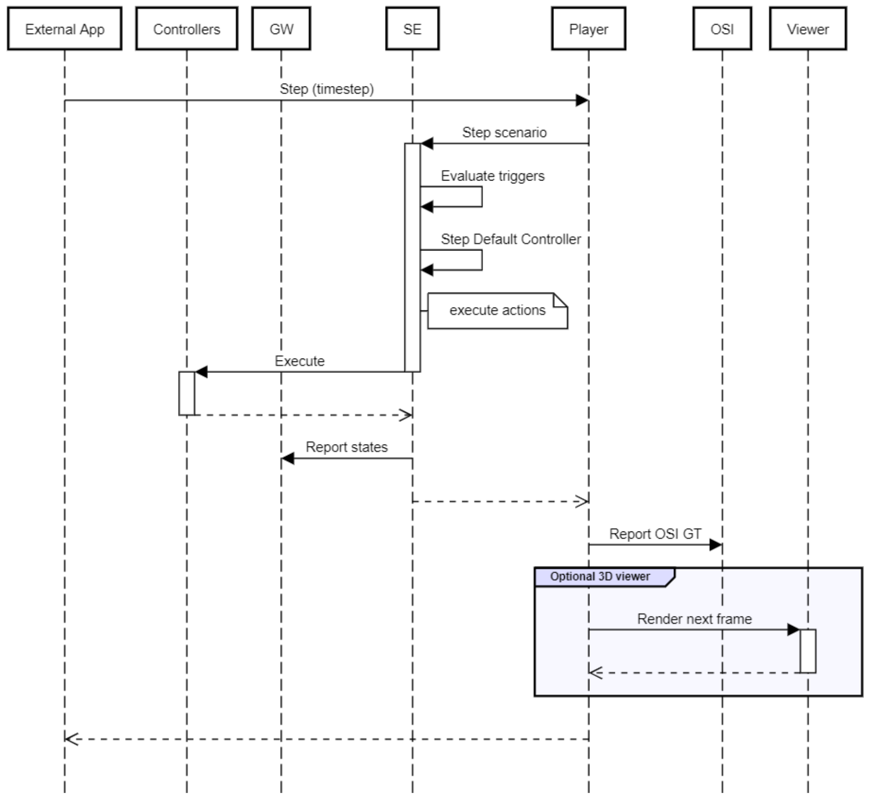


Figure 2.2 Frame sequence Esmini - High level [Knabe, 2021a].

In Figure 2.3 an overview of the class hierarchy is given. This arrow, which can be seen in the figure, is a standard aggregation arrow. Aggregation means that ownership is not implied [Jillre and Jesong, 2016].

Only the ScenarioEngine in the UML diagram observed in Figure 2.3 is of interest for this thesis. For that reason, it is the only class that will be further explained of the seven in the figure.

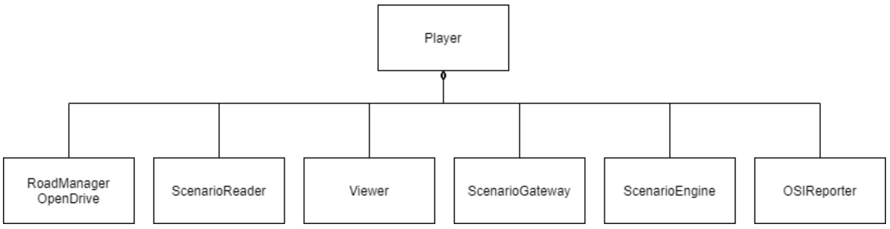


Figure 2.3 Esmini high level UML class diagram [Knabe, 2021b].

The Scenario Engine

Figure 2.4 shows that the classes catalogs, entities and storyboard are linked through aggregation to the class ScenarioEngine. The storyboard and catalog classes are defined by the OpenSCENARIO standard and implemented as such. Entities is a class containing all objects in the scenario.

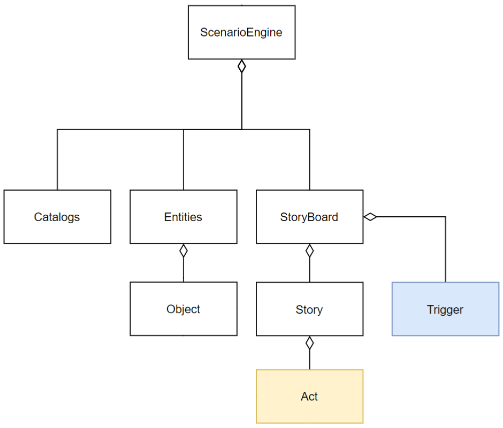


Figure 2.4 Esmini UML class diagram for the scenario engine [Knabe, 2021b].

Actions and Events in Esmini

Both actions and events are defined by the OpenSCENARIO standard. As previously mentioned there are three types of actions whereof only two are supported in Esmini. UserDefinedActions is as of early spring 2021 not implemented. In Figure 2.5 it is explained how events are linked to other classes. An event can contain one or more actions and has a trigger which starts it. It is also clear that there are no links to any object at this level since the action can be either global, not acting on an object, or private, acting on an object. However if we step up one level in the class diagram there is the maneuver group which links a maneuver to an object. This can be seen in Figure 2.6. All these links are unidirectional, meaning that e.g., an action does not know which event they are part of. Since they are also linked through aggregation a lower class in the diagram can outlive a higher one by e.g., being assigned to another event in the case of an action.

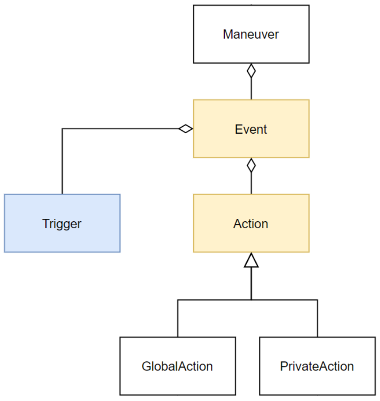


Figure 2.5 Esmini UML class diagram for actions and events [Knabe, 2021b].

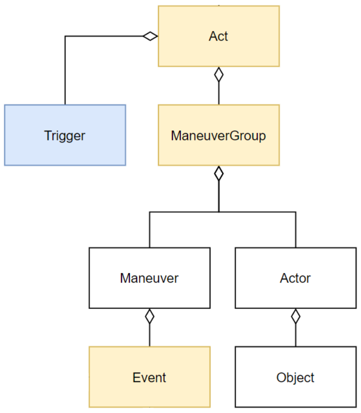


Figure 2.6 Esmini UML class diagram for maneuver and maneuver group [Knabe, 2021b].

OpenSCENARIO defines plenty of actions but only some of them have been implemented in Esmini. There are currently three types of global actions implemented whereof only one is from OpenSCENARIO and that is a set parameter action. The two others are traffic swarm and environment/infrastructure actions. The private actions which are actions on a specific entity are:

- Longitudinal speed
- Longitudinal distance
- Lateral lane change
- Lateral lane offset
- Assign controller
- Activate controller
- Teleport
- Assign route
- Follow trajectory
- Synchronize

All these actions can have different setups with relative or absolute targets and other flags as being continuous, i.e., being a non-ending action. OpenSCENARIO features a large number of different private actions setups [*ASAM OpenSCENARIO* 2021]. The most relevant of these that have been implemented will be presented in Table 2.2. All actions in the table directly control movement of an object. Several of these actions goals are self explanatory, e.g., longitudinal speed action with the continuous flag set to true and relative target will keep a relative speed to an object regardless of what happens. The assign route action will technically not move the vehicle but force it to move along a predefined route meaning lateral motion from the route is prohibited. Instead the vehicle will be moved with a special move along route call and the speed can still be controlled by actions. The same is true for the follow trajectory action.

The most interesting actions for this thesis are those which can be set up relative to another object and are executed during more than one timestep, which are the following:

- Longitudinal speed

- Longitudinal distance
- Synchronize

The longitudinal speed action can be set up such that a relative speed, defined as the desired difference, towards another object is achieved. Worth noting is that a speed action can have four different shapes (step, linear, cubic, sinusoidal), which can be seen in Table 2.1. Both cubic and sinusoidal must according to the OpenScenario standard start with a gradient of zero [*ASAM OpenSCENARIO* 2021]. This is the shape the transition between the old speed and the new speed, specified by the action, will have. When the speed is to be achieved can be specified in three ways: at a predefined constant rate, in a predefined time, or in a predefined distance. A predefined rate is, in a linear case, equivalent to the acceleration that the change in speed have.

The longitudinal distance action is a little more complex than the speed action, but in short it will ensure that a relative distance is achieved with constraints available on both acceleration and deceleration. It does this by calculating how to manipulate the speed to reach the target distance. The synchronize action will be covered in the next section.

Table 2.1 The different shapes a speed profile in a speed action can take.

linear	$f(x) = f_0 + rate \cdot x$
cubic	$f(x) = A \cdot x^3 + B \cdot x^2 + C \cdot x + D$
sinusoidal	$f(x) = A \cdot sin(x) + B$
step	not applicable, instantaneous

Table 2.2 A selection of Esmini actions and their settings.

Action	**Settings (time, space, domain)**	**Controls axis**	**Action Ends**
Longitudinal SpeedAction	target is "absolute" or "relative" with continuous = "false"	Longitudinal	on reaching the speed
Longitudinal SpeedAction	target is "relative" with continuous = "true"	Longitudinal	never
Longitudinal DistanceAction	*continuous = "false"*	Longitudinal	by reaching the targeted distance
Longitudinal DistanceAction	*continuous = "true"*	Longitudinal	never
Lateral LaneChangeAction	not applicable	Lateral	by reaching the lane
Lateral LaneOffsetAction	*continuous = "false"*	Lateral	by reaching the targeted lane offset
Lateral LaneOffsetAction	*continuous = "true"*	Lateral	never
SynchronizeAction	not applicable	Longitudinal	controlled vehicle reaching the target point
Routing AssignRouteAction	not applicable	None	immediately
Routing FollowTrajectoryAction	not applicable	Lateral/Both	by reaching the end of the trajectory

The Synchronize Action

The synchronize action is the most advanced private action in Esmini, especially in terms of implementation. This action does not, compared to many of the others, describe a current or desired current state but rather a future state. This means there are several ways of reaching this state and the solution is not trivial. However the problem is simplified to one dimension. The actions goal is to synchronize arrival at two points, with the additional option of specifying a speed for the vehicle at that point. An example can be seen in Figure 2.7. At point 2 one could then specify a speed for the target vehicle. [*ASAM OpenSCENARIO* 2021]

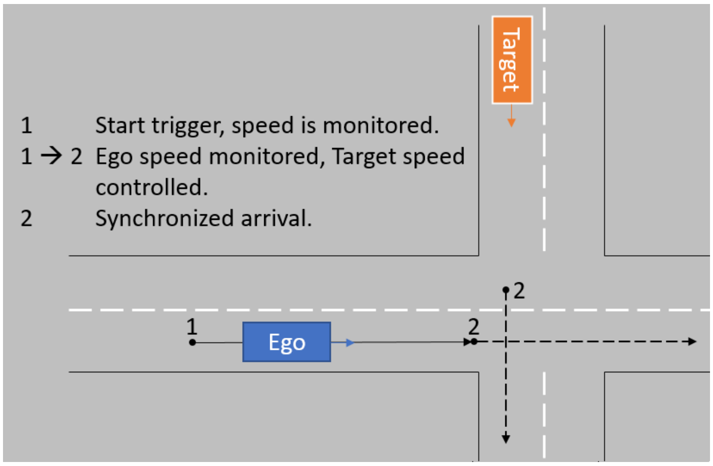

Target
1 Start trigger, speed is monitored.
1 → 2 Ego speed monitored, Target speed controlled.
2 Synchronized arrival.
2
1
Ego
2

Figure 2.7 Synchronized arrival at an intersection.

To calculate the desired motion for the vehicle under control by the action one has to divide the problem into different cases. In Esmini this is done such that the magnitude of acceleration will be minimized [Knabe, 2021a]. This achieves smooth operation with regards to acceleration and deceleration. To prevent a solution with negative speeds, stops have been implemented to still achieve the desired result. Knabe et.al. claim this solution maximizes the chance of succeeding even if the vehicle is also constrained by external vehicle dynamics or brake limits.

The first two main cases of the solution include either solving for the desired motion using a specified or an unspecified speed. For unspecified speed the acceleration is assumed constant over a time period t and calculated from:

$$s = v_0 t + \frac{1}{2} a t^2 =>$$

$$a = \frac{2(s+v_0t)}{t^2}$$

where v_0 is the current speed of the action entity, s is the remaining distance for the action entity and t is the estimated remaining time for the master object. The new speed is the calculated according to $v = v_0 + a \cdot dt$, where dt is the sample time.

In the case of a specified final speed the task is to control the speed profile such that we arrive just in time. Note that both cases have a constrained starting speed which is the current speed of the entity. To cover a distance s in a specific time t the average speed v_a must be equal to s/t. One could also describe it as the integral, or area, of v_a over the time t should be equal to the distance s as seen in Figure 2.8.

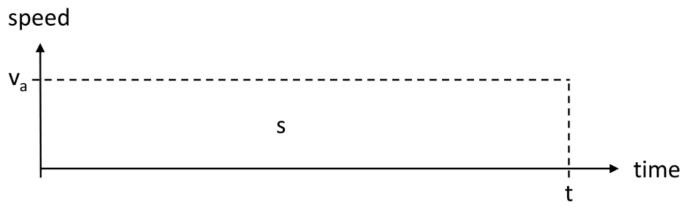


Figure 2.8 Average speed needed to cover the distance s in the time t.

This method will be divided further into four subcases where different conditions apply. It is assumed that the acceleration or deceleration is constant. If both acceleration and deceleration are needed to reach the desired state they are assumed to be equal in magnitude. Since the master entity can change speed at any time the acceleration needs to be reevaluated each step. The first subcase is with a linear speed profile meaning the target can be reached with a constant acceleration according to Figure 2.9.

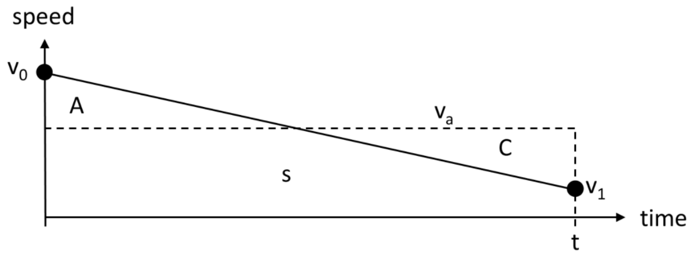


Figure 2.9 Constrained starting and final speed with linear speed profile.

The acceleration can simply be calculated according to $a = (v_1 - v_0)/t$. It is worth noting that the area A and C are equal resulting in the same distance covered as in Figure 2.8.

The second subcase is for when a single linear acceleration or deceleration is not enough, but rather one of each is required. This is referred to as a non-linear speed profile and can be seen in Figure 2.10.

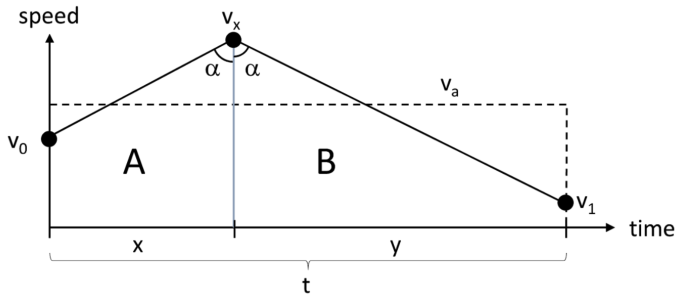


Figure 2.10 Constrained starting and final speed with non-linear speed profile.

This is in reality a new more complex constant acceleration case, for phase x, together with the linear case, for phase y. The absolute acceleration is assumed to be the same for both phases. To calculate the acceleration needed to reach speed v_x a system of equations are set up as follows:

$$\begin{aligned} A &= v_0 x + (v_x - v_0)x/2 \\ B &= v_1 y + (v_x - v_1)y/2 \\ t &= x + y \\ s &= A + B \\ (v_x - v_0)/x &= (v_y - v_0)/y \end{aligned}$$

where t, s, v_0, v_1 are known and A, B, x, y, v_x are unknown. By solving for x and v_0 the acceleration can then be calculated according to $a = (v_x - v_0)/x$. When v_x is reached Esmini switches to the linear case.

The third subcase is a variation of the non-linear case but with a defined stop, meaning that to prevent backwards movement a stop is needed as can be seen in Figure 2.11.

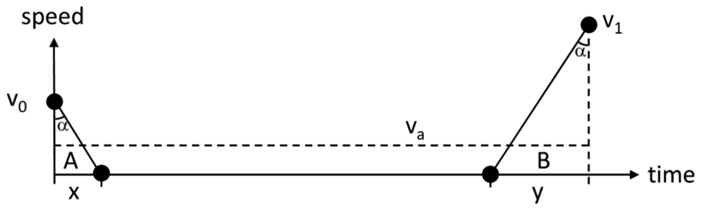


Figure 2.11 Constrained starting and final speed with non-linear speed profile and a defined stop.

When the remaining time allows for a linear transition, i.e., y time left, Esmini handles this as a linear case. Since the final speed for phase x is zero the equation system becomes:

$$\begin{aligned} A &= v_0 x/2 \\ B &= v_1 y/2 \\ s &= A + B \\ v_0/v_1 &= x/y \end{aligned}$$

where s, v_0, v_1 are known and A, B, x, y are unknown. Solve for x and the acceleration can then be calculated as $a = -v_0/x$. The absolute acceleration is assumed to be the same for both phases just like in the previous subcase.

The last subcase is a non-linear speed profile with an undetermined stop, meaning that the master entity is for any reason not moving and the final speed should be zero. This should result in an immediate stop as can be seen in Figure 2.12. When the master entity starts moving the action will enter one of the previously described modes. [Knabe, 2021a]

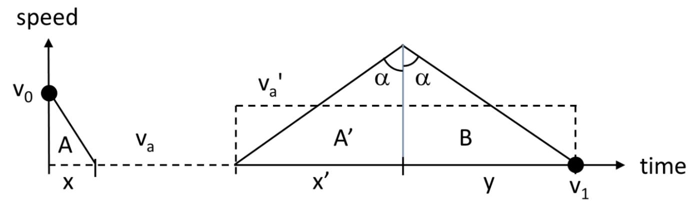


Figure 2.12 Constrained starting and final speed with non-linear speed profile and a undetermined stop.

Controllers in Esmini

Controllers exist in the OpenSCENARIO standard but unlike the actions there are no subtypes defined. All controllers use the same API defined in an interface in Esmini [Knabe, 2021a]. This API defines that every controller is attached to an object which it also controls. It also, among others, specifies a step function that will be called by the scenario engine if said controller is activated. The default controller is however not handled as a controller in Esmini but rather by itself. Any references to a controller will therefore not apply to the default controller.

Every controller has a mode that manages how to deal with actions. This mode decides whether a controller should have complete control over the object it is attached to or if actions also can manipulate it. If the controller has the mode override it will, as the name suggests, override all actions. This is the default setting. The other mode is additive where the actions movements are simply added to the controllers, or rather vice versa since actions are performed before controllers are evaluated as mentioned previously.

As defined by the OpenSCENARIO standard, a controller can be assigned on either the longitudinal, lateral or both domains in Esmini. In Esmini these domains are handled differently through an API. You can either move along the current road, path or trajectory, i.e., longitudinally, or move laterally from the current longitudinal path. As an example, a lane change will only move a vehicle laterally and depends on the vehicle being moved longitudinally elsewhere. This longitudinal movement could be by an action or by the default controller.

A Scenario Engine Frame Sequence

For every frame sequence, or step, the scenario engine handles a lot of information as was shown in the start of the chapter, in Figure 2.2. The length of the step is either fixed, i.e., it is set by the user, or calculated by the scenario engine. All initial actions are stepped forward if still applicable, all triggers are evaluated, and all states are updated. Storyboard elements can be in three states; standby, running and complete. Furthermore all storyboard elements are evaluated if they are to be started and actions are stepped. Then the so called default controllers are stepped which basically means that entities not controlled by actions nor controllers move forward along the road or path with a constant speed. Lastly the controllers are evaluated and manipulate affected entities. This is a full explanation of all dynamic content handled, in order, by the Esmini scenario engine.

Esmini Reference Frame

Esmini uses a global coordinate system with three parameters, x,y and z. The orientation of these will depend on the scenario but the x and y coordinate are always defined as the horizontal- or ground-plane. Z is defined as the vertical displacement, i.e., height, from this plane. Entities in this frame of reference will in most cases have, in addition to these coordinates, a coordinate called s. This parameter is the distance travelled along the current road network path or trajectory, which is a path you constrain a vehicle to travel along. If there is no road or trajectory assigned to the object, this value s will be zero or undefined. Each entity also has a velocity and a heading in the global coordinate system. The heading is defined in degrees from the x-axis.

Esmini vehicle model

In Esmini all entities are modelled as point masses. They have a position in the global three dimensional coordinate system, a heading angle in the xy(ground)-plane as well as pitch and roll angles. Furthermore they have a speed in the heading direction. There is also support for more advanced models that can be updated through an external application, e.g., an AD software. This could be that the when the car brakes, more weight is transferred to the front suspension which would mean that the pitch angle increases and the front is pushed closer to the ground.

Through a simple API one can manipulate the entities on the ground plane along a road, path or trajectory. The method call MoveAlongS uses a distance as an argument and will move the entity along the current road, handle intersections according to a junction strategy type, random by default, and ensure the entity follows the roll and pitch of the road. The length of this step is determined with the known simulation step time for the frame sequence, entity speed and any dynamic change, e.g., acceleration.

2.3 Background on Simulation Model Testing

Testing simulation software models can be broadly defined as revealing and identifying errors in the simulation model [Balci, 1995]. In practice, this means that test data for different test cases is collected from the simulation software. Simulation model testing can be comprised of either verification, validation, or a combination of both. A paper by R.G. Sargent adopts the definition of validation as being “substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model” [Sargent, 2004]. Hence, validation attempts to investigate the accuracy of the mathematical model used in a particular simulation. On the other hand, verification can be defined as “ensuring that the computer program of the computerized model and its implementation are correct” [Sargent, 2004]. Verification therefore attempts to investigate the accuracy of the computer program, rather than the mathematical model itself. The combined process of validation and verification can be referred to as VV&T testing or simply model testing [Sargent, 2004].

There are a number of different VV&T techniques that can be used for testing simulation software. These include, but are not limited to [Balci, 1995]:

- Informal Methods:
  - Face Validation
  - Inspection
  - Reviews
- Static Methods:
  - Consistency Checking
  - Data Flow Analysis
  - Semantic Analysis
- Dynamic Methods:
  - Black-Box Testing
  - Debugging
  - White-Box Testing
- Symbolic Methods
- Constraint Methods
- Formal Methods

This thesis focuses mainly on testing using dynamic methods, more specifically black-box testing and white-box testing.

White-Box Testing

White-Box testing (or white-box validation) assumes that the individual components (e.g., source code) for the simulation software are completely accessible. The testing method is then executed in an attempt to analyse different subsystems of the simulator, and to validate whether these are sufficiently accurate [Thaler, 2021]. This can be very useful in order to gain a detailed understanding of the system, and which components of it need further improvement. In addition, the understanding of the model itself and the simulation as a whole can be greatly improved through this. Hence, using this method of testing, both the logic of the model and the behaviour against the real world can be validated. Jonthan Thaler outlines several approaches for implementing white-box testing practically, including but not limited to [Thaler, 2021]:

- Stepping through different functions/components of the simulation software

- Creating predictions of what the outcome of a certain subsystem will be, and then stepping through said subsystem in order to validate the prediction
- Setting up condition such that certain desired events take place (events in this case do not refer to OpenSCENARIO/Esmini events)
- Evaluating the system performance under extreme conditions
- Tracing

Black-Box Testing

In contrary to white-box testing, black-box testing is performed on systems where there is no knowledge of the individual components of the system. Instead, the overall behaviour of the model is considered [Robinson, 2004]. There are two main approaches to black-box testing, which are making a comparison to real-world data and making a comparison to other simulation models. The latter is particularly useful if there is no access to reliable real-world testing data for the model that is being validated. However, this does not preclude the use of comparing the model to other simulation models if there is reliable real-world data available, as the use of both of these methods can further increase confidence in the validated model. There is however unfortunately no real-world data available that corresponds to the simulations conducted in Esmini, and hence this thesis focuses on comparing the system to other simulation models.

For validation by comparing the model to other simulation models, the comparison is usually made to models that intend to simulate the same or similar systems. It can also be useful to compare the simulation model to a mathematical model, as long as the mathematical model can make predictions with sufficient accuracy. In this case, the simulation model should be simplified if needed, so that adequate comparisons can be made.

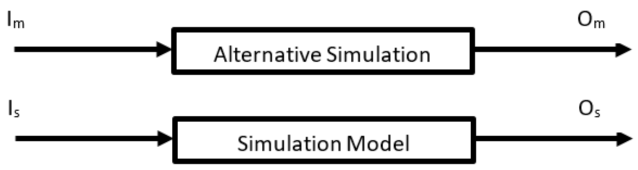


Figure 2.13 Black-Box Validation: comparison to another simulation model [Robinson, 2004]

Figure 2.13 shows an illustration of the method of comparing the simulation model to another simulation. I_s, I_m represent the input variables for the simulation model intended to be validated and the simulation that it is validated against, respectively, while O_s, O_m represent the outputs of each simulation. The main hypothesis, H_1, of this validation technique is then if $I_s = I_m$, then $O_s \approx O_m$. Hence, both simulation models are given the same input variables, and their outputs are measured against each other.

3

Controller Design

3.1 Controller Concepts

Several controller concepts were explored and partially tested, with the aim of selecting a final version to be implemented in Esmini. Each controller concept was designed to address key research questions, such as when the switch should occur and what the control objective is after the switch.

Motion Analysis Controller

A first approach to designing the controller was to analyze the dynamic behaviour of the ego vehicle. The aim of this was to detect behaviour where the ego vehicle reacts to an unsafe traffic condition. This could for instance be emergency braking in a 4-way crossing or a lane change. Since reactions to unsafe traffic scenarios by the ego vehicle often involve braking maneuvers, an early strategy for switching was to switch when a large longitudinal deceleration by the ego vehicle was detected. Since vehicle acceleration is not directly accessible through the object instance of the vehicle in Esmini, it was approximated by differentiation of the velocity at each timestep (Equation 3.1).

$$a_k = \frac{v_k - v_{k-1}}{t_k - t_{k-1}} \tag{3.1}$$

The acceleration is then compared to a threshold value. If the absolute value of the acceleration exceeds the threshold, the target vehicle will switch from relative to absolute control (see Listing 3.1).

```cpp
if (mode_ != Mode::MODE_OVERRIDE)
  {
    if (fabs((egoSpeed - prev_ego_speed) / timeStep) > threshold)
    {
      std::cout << "Now I'm using absolute control!" << std::endl;
      mode_ = Mode::MODE_OVERRIDE;
    }
  }
```

Listing 3.1 Switching mechanism in motion analysis controller concept.

The *mode_* object refers to whether the controller can override actions in order to control its referenced entity. While the *mode_* object is set to MODE_ADDITIVE, active actions will control the motion of the target vehicle rather than the controller itself. When switching occurs, *mode_* is set to MODE_OVERRIDE, meaning that the controller takes control of the target vehicle. The controller then controls the target vehicle such that it continues at the same speed that it had before the switch. It does so using a point-mass model that follows the given trajectory:

$$x_k = x_{k-1} + v_{k-1} \times (t_k - t_{k-1})$$

Action Controller

Another controller concept that was explored involved analyzing Esmini actions in order to determine when to switch, and what the control objective should be after the switch. This controller concept attempts to analyze the actions of a scenario in order to create a prediction for how the target vehicle will behave. Throughout the simulation, this predicted states for each time step are then compared to the actual states of the target vehicle. Algorithm 1 illustrates a pseudocode for how actions can be "predicted" using a "step function" built into Emsini for each action. By using the step function, the target vehicle state can be predicted for the entire action. It is important to note that this "prediction algorithm" is taking place within a single timestep of the actual simulation itself.

```
target_copy = target.copy()
while action is active do
    action.step
    speed_est = target_copy.speed
    x_est = target_copy.x
    y_est = target_copy.y
end
```

Algorithm 1: Basic pseudocode for predicting an action.

After the loop in algorithm 1, the predicted states are continuously compared to the target vehicles actual states. A simple switching mechanism can be implemented by comparing whether the x and y states of the target vehicle deviate significantly from their prediction. In order to do so, a position error is calculated for each timestep (Equation 3.2). If the error exceeds a threshold, the vehicle switches to absolute control.

$$error = \sqrt{(\hat{x}_k - x_k)^2 + (\hat{y}_k - y_k)^2} \tag{3.2}$$

This concept was also tested by implementing the pseudocode into C++ code in Esmini. Unfortunately, this test ran into several issues. The first issue was that the loop in Listing 3.2 would in some cases run indefinitely, and thus cause the whole scenario simulation to get stuck. The reason for this is that the activate controller action does not have a step function, and hence never becomes inactive. Thus, the while-loop continues indefinitely in this case. Another issue was that there were cases where no actions were active, and hence no pre-simulation could be obtained.

Evaluation of Controller Concepts

The motion analysis controller proved to work quite well for the scenario that it was tested on (see Appendix A for OpenSCENARIO code). However, this type of motion analysis is simply too simplistic to function properly on other types of scenarios. For instance, if the ego vehicle reacts slowly to the motion of the target vehicle rather than abruptly, the ego vehicle's deceleration will never be very large. This could prevent the controller from switching in instances were a switch should occur. Further, the target vehicle simply continues with the same constant velocity after the switch that it had prior to the switch. However, some scenarios may require the control objective to be more sophisticated than simply holding a constant velocity. For instance in crossings, the target vehicle may be required to turn into another street. Hence, the final controller must take actions that occur after the switch into account.

The action controller had several issues when tested in Esmini and would need to be rewritten in order to function sufficiently. However, since the action controller analyses actions that occur in a scenario, it has a lot more potential to control the target vehicle such that it behaves according to the scenario description. The ability to complete scenario actions even after the switch occurs is a key benefit of the action controller. Another benefit is that it has the potential to switch even when the ego vehicle reacts slowly to a potential collision and hence does not exert a large deceleration. Because of this, the action controller concept was chosen to be implemented in further detail in Esmini. This implementation seeks to address issues that the previous action controller had, while maintaining the general concept of extrapolating actions in order to decide when a switch should occur.

3.2 Controller Implementation

Since the concept of predicting vehicle states using actions has been found to have more potential, this concept is implemented for the final controller. This means that the final controller continuously predicts the x,y position as well as the speed of the target vehicle. If the actual position and speed deviate from their respective counterparts by a threshold set by the user, the controller will switch from relative to absolute control. In addition, the controller will only predict the states for a limited time ahead. This time, known as the "prediction horizon" is also set by the user (in seconds). Overall, this implementation results in a controller using predictions for switching, with three separate tuning parameters.

Vehicle Modelling

The predictions for the target vehicle are modelled using a point-mass model. This is due to the fact that Esmini also models the target vehicle as a point mass. The reason for this is that the goal of Esmini simulations is not to investigate the dynamic behaviour of the target vehicle. Instead, the aim of the controller is to control the movement of the target vehicle such that it reflects a realistic traffic scenario. Hence, a point-mass model (also known as kinematic bicycle model) is the most suitable model for generating predictions. Assume that the state of the vehicle can be modelled by:

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.3}$$

where x, y represent the position of the vehicle, while θ represents its heading. All three variables are defined with respect to the global coordinate system outlined in Section 2.2. The point mass model is then given by Equation 3.4 [Umit Ozguner and Redmill, 2012].

$$\begin{aligned} \dot{x} &= V_s \cos(\theta) \\ \dot{y} &= V_s \sin(\theta) \\ \dot{\theta} &= r \end{aligned} \tag{3.4}$$

This point mass model is already implemented in Esmini as part of the position class, and this existing function is hence used in the controller code.

Prediction algorithm

The point mass model presented in Section 3.2 is used in order to predict the kinematic motion of the target vehicle. This is however only one part of the prediction algorithm. The other part makes use of the step function of any active actions associated with relevant objects. The prediction algorithm combines these two elements, which is illustrated in Figure 3.1.

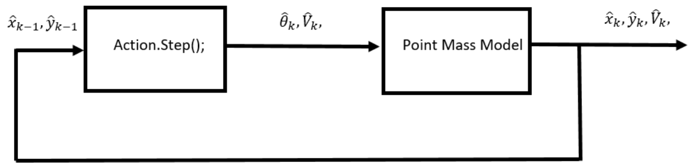

$\hat{x}_{k-1}, \hat{y}_{k-1}$
Action.Step();
$\hat{\theta}_k, \hat{V}_k,$
Point Mass Model
$\hat{x}_k, \hat{y}_k, \hat{V}_k,$

Figure 3.1 Block diagram illustrating how the prediction algorithm makes use of actions and the point mass model to predict vehicle states.

The previous states are used as input to the action's step function. Note that during the first time-step of the prediction algorithm, the actual vehicle states are used instead. This is then used in the step function in order to calculate the heading and the speed of the vehicle. The values for position, heading and speed are then used in the point mass model (Equation 3.4) in order to predict the position in the next time-step. Values for x,y and speed are saved for each time-step.

Algorithm 1 was altered to make use of the block diagram in Figure 3.1. The result of this is Algorithm 2, which shows the pseudocode for predicting actions. Rather than pre-simulating actions for their entire lifetime, this algorithms uses a so-called prediction horizon for which it pre-simulates the target vehicle. For instance, if the prediction horizon is set to 1 second, predictions for 1 second into the future will be generated by the algorithm. Using a prediction horizon rather than pre-simulating entire actions or scenarios is computationally more efficient while still providing enough data for the controller to perform a switch.

```
for all objects do
    pos = copy_pos(object)
    speed = save_speed(object)
end
for all actions do
    copy(action)
end
for prediction_horizon do
    for all objects do
        if object has active action of the right type then action.step
        steplen = timestep * speed
        pos.use_point_mass_model(steplen)
        save(x,y,V)
    end
end
Return objects to original state
```

Algorithm 2: Algorithm for predicting target vehicle behaviour.

The start of the algorithm copies all objects and actions, in order to ensure that no elements in Esmini are altered by the pre-simulation algorithm. In addition, the algorithm disregards types of actions that do not need to be pre-simulated. For instance, activate controller actions do not need to be pre-simulated since they are merely used for activating a controller and do not affect the target vehicles motion. The actions simulated will then affect various vehicle parameters, most notably the heading and speed. For instance, a longitudinal speed action will change the speed of the vehicle, whereas a lateral lane change action will gradually change the heading of the vehicle such that it changes lanes. After all actions were called using their respective step function, the point mass model (Equation 3.4) is used in order to pre-simulate the target vehicles motion.

Switching Mechanism and Controller Tuning

In the initial time step of the simulation, the pre-simulated states are generated for a prediction horizon of t seconds ahead of the target vehicles current state. It was decided that the global x,y-position as well as speed are used as state variables. However, other variables such as heading or z-position could also be used as state variables. As the simulation is running, the controller compares the pre-simulated state values to the target vehicles actual state values using Equation 3.2 for the duration of the prediction horizon. After prediction horizon time t (for instance 1 second) has elapsed, a new prediction is calculated using Algorithm 2 for the next time steps within time t. This cycle continues throughout the simulation, and is illustrated by Figure 3.2. If the error at any point in time exceeds a given threshold value, the controller switches from relative to absolute control.

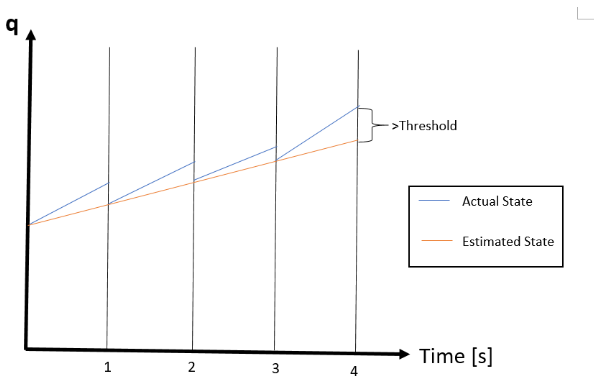


Figure 3.2 Conceptual visualization of how the prediction algorithm is used for switching.

In Figure 3.2, the prediction horizon is set to 1 second, and hence a new prediction is calculated every second. While this is a constant value throughout a single simulation, both the prediction horizon and error thresholds can be used as tuning parameters in order to adjust the controller to a certain scenario. Position and speed each have separate error threshold values that can be tuned for different scenarios. A shorter prediction horizon has the advantage of reacting to potential actions that become active faster. For instance, if an action becomes active between two calculated predictions, that action will be taken into account for the next prediction after 0.5 seconds if the prediction horizon is 1 second. However, if the prediction horizon is 3 seconds, it will take 1.5 seconds until the new action is accounted for in future predictions. In addition, predictions are not fully accurate, and will hence deviate from the actual states as illustrated in Figure 3.2. A longer prediction horizon may cause the controller to switch at an incorrect time because the prediction deviated too much from the actual state. In an ideal case, the prediction should only deviate slightly from the actual state until the ego vehicle triggers an emergency braking function or otherwise reacts to a potential collision. A longer prediction horizon however has the advantage that it may capture reactions by the ego vehicle happening over a longer period of time. For instance, the ego vehicle's AD software may sense a collision scenario 10 seconds in advance, and slow down over a longer period rather than emergency braking quickly. In this case, a shorter prediction horizon would cause a failure of the controller to switch at an appropriate time. The values for the speed and position error thresholds can be adjusted in order to achieve a switch at a desired time.

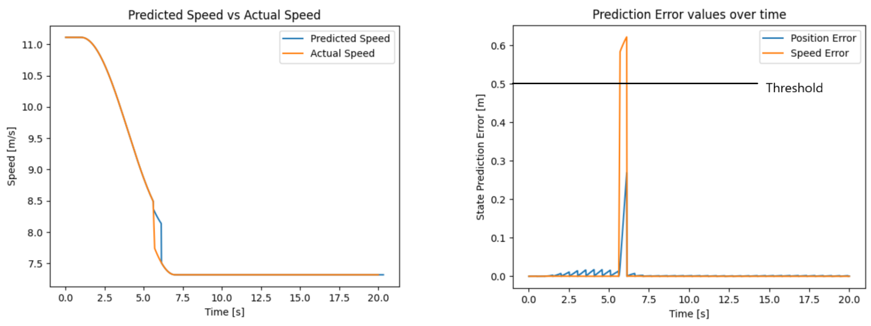


Figure 3.3 Example of prediction algorithm used for switching.

Figure 3.2 shows an example of the prediction algorithm during a lane change scenario. In this scenario, the ego vehicle uses a collision mitigation by braking (CMBB) function to avoid a collision with the target vehicle (see Section 4.1 for more detailed description). Because the prediction algorithm simulates active actions, the speed is predicted with near-perfect accuracy until around 5.4 seconds. The reason for this is that in this example, the ego vehicle is controlled by Esmini rather than an external software. Therefore, the controller has access to actions for all objects (including the ego vehicle), and can therefore predict their motion with a very high degree of accuracy. After 5.4 seconds, the ego vehicle triggers a speed action that causes it to slow down. Since the target vehicle's speed is defined relative to the ego vehicle's speed, the target vehicle will also reduce its speed. Due to the prediction horizon of 1 second, this is however not taken into account by the prediction algorithm immediately, and hence the error in speed estimation rises above the specified threshold. This causes the target vehicle to switch from relative to absolute control. In general, relative scenarios that include a higher acceleration/deceleration by the ego (and by extension the target) vehicle, are tuned to have higher thresholds. The reason for this is that the speed and position errors will be higher as a results of the greater change in speed. Tuning parameters for common relative Esmini scenarios are outlined below. Note that tuning parameters for each of these can vary depending on rate of change in speed, type of active safety function used etc.

In Figure 3.2, the controller is tuned using the default parameters for the controller, which are as follows:

- Prediction horizon = 1 second
- Position error threshold = 1.8 m
- Speed error threshold = 0.5 m/s

These tuning parameters ensure adequate controller performance for scenarios using CMBB and ACC safety functions (these scenarios are described in more detail in Section 4.1). Since the position error is not very sensitive in these scenarios, the speed error is mainly used for switching. Setting this to 0.5 m/s, along with the standard prediction horizon of 1 second, the controller will switch as the active safety function is triggered by the ego vehicle.

For scenarios using the relative distance action, the tuning parameters can be set to the following values:

- Prediction horizon = 1.1 s
- Position error threshold = 2.0 m
- Speed error threshold = 2.0 m/s

These values cause controller to switch both using the position error threshold and the speed error threshold. The tuning parameters may however vary depending on how the scenario is set up, and what active safety function is used.

Scenarios using the synchronize action can be tuned using the following parameters:

- Prediction horizon = 1 s
- Position error threshold = 2 m - 5 m
- Speed error threshold = 2 m/s - 5 m/s

Errors generated during the synchronize action tend to be large compared to other scenarios, and hence both thresholds are set to relatively large values. The exact value for each threshold depends on what additional actions the scenario contains, how large any rate changes in speed are and what safety functions are used.

Control Objective after the Switch

Defining the control objective after the controller has switched from relative to absolute control is not trivial. It depends on what was happening before the switch and what the goal of the simulation is. Initially the desired behaviour was defined as the behaviour that would occur given that no active safety function was engaged. This proved achievable for the simpler actions with well defined behaviour, for the synchronize action however a different approach is used.

In the general case, the controller will create a new action with a speed goal that tries to mimic a believed future steady state. In the simplest case, which is a longitudinal speed action with a relative target, the target is simply converted to an absolute speed action. This is done by retrieving the ego's target speed and then setting the new target as the retrieved speed. Esmini will automatically convert the relative target speed to an absolute target speed. If the controller switches quickly enough, meaning it does not allow the target to slow down, it ensures that the correct speed will be achieved at the right time. This is because all dynamics are the same as the original action that was defined by the user. It will also be indistinguishable from a speed profile where an active safety function was not activated. This can be seen in Figure 3.4, where the blue and orange curves represent a scenario where an emergency brake function was activated. The green and red curves show the same scenario but without a safety function. There is is no difference between the orange and red curve due to the switch occurring instantaneously. If the controller would not have switched, the orange curve would have started dropping at the same rate as the blue curve and would have fallen to negative five.

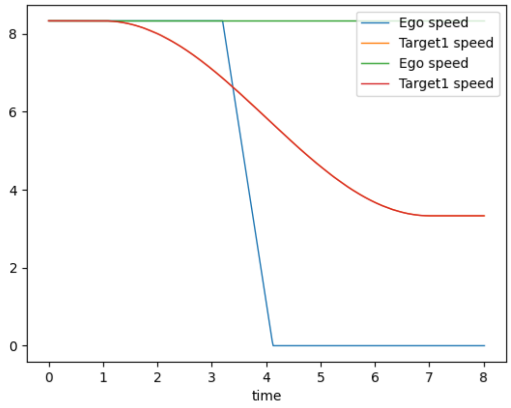


Figure 3.4 Example of a simple emergency brake scenario with 2 overlapping runs, one with (blue and orange) and one without (green and red) the braking activating. The orange curve cannot be seen due to being identical to the red curve.

If the target vehicle is instead constrained by a relative longitudinal distance to another vehicle, a new longitudinal speed action must be created. In this case it is assumed that the target is either close to reaching steady state conditions, or is close to doing so. Any dynamics from the current action that the controller is switching from is copied to the new one. If there are none, the maximum allowable acceleration for the target to reach the actions goal is set to 10 m/s^2. This is due to a number being needed to call the actions step function, however it will not affect the speed profile since the current speed is used. Since a new action has been created, it must be added to the event that contains the longitudinal distance action to ensure it is visible for the scenario engine. The longitudinal distance action is then removed, and from the scenario engine's perspective nothing has changed. It continues to step the function through a base class, unaware of which type of private action it is or was.

The last case is for the synchronize action, which is more complicated. There are several modes during the lifetime of a synchronize action, as shown in Chapter 2.2. The current mode influences whether the controller will switch from relative to absolute control. For the case where a final speed is specified, the controller will only switch from the synchronize action if it is in linear subcase. All other subcases will only occur further from the endpoint. It is assumed that the ego will not react in the beginning of the action. Since the final speed is specified and we know at which point to achieve this speed, the longitudinal speed action is created such that it will be achieved in the current distance to this point. The transition will either assume a linear or a cubic shape, depending on whether the target is currently accelerating or decelerating. A linear shape is selected if the target is accelerating, since this will prevent the gradient of the speed from resetting to zero when the new action starts. If the target is decelerating, a cubic shape is chosen such that an abrupt change in the direction of the speed profile is avoided. In the case where a final speed is not specified no constraints on whether a switch should occur or not are applied.

3.3 Integration of the Controller with Esmini

The scenario file can assign a specific controller to the target vehicle using an assign controller action. It does so by referencing the controller in the controller catalog, where each controller in Esmini is listed. The controller catalog provides the controller's type name, which is used to assign the controller to the target vehicle. Figure 3.5 shows this with a controller called "MyController" as an example.

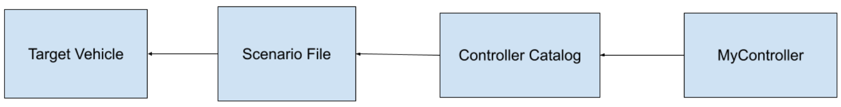


Figure 3.5 Flowchart of how a controller is integrated into Esmini.

Identifying the Ego

The controller has access to a vector with every object in the scenario. It can therefore access data about every other object in the scenario. Since the ego vehicle is defined by the user, identifying it is not always trivial. Solely relying on the name for identifying the ego vehicle is not sufficient, as the user may use any name to define the ego vehicle while creating the scenario description. However, since the convention is to name the ego vehicle "Ego", the controller attempts to search for any vehicles with this name. If there is no vehicle named ego, the controller instead searches for a vehicle that is controlled externally. This implies it is the vehicle on which active safety software is tested, since it is controlled by an external software. As a fallback if all the above fails, it will default to the first vehicle added in the scenario file. Normal practice is that the ego vehicle is listed as the first vehicle in the scenario file.

Esmini Modifications

Much of the functionality needed to implement the controller is already available in Esmini. The only area not supported is the ability for controllers or objects (e.g., a vehicle) to manipulate actions. This was previously handled exclusively by the scenario engine. To resolve this problem, a vector with events containing at least one private action each, is added to each object. This means that the scenario engine is modified in order to update and create these vectors. Additional modifications are made such that initial actions, which are handled differently in Esmini, can be accessed. One of these updates can be seen in Listing 3.2. This code will run every timestep for every existing event.

```
if (event->IsTriggable() || event->IsActive())
{
  for (size_t n = 0; n < event->action_.size(); n++)
  {
    OSCAction* action = event->action_[n];
    if (action->base_type_ == OSCAction::BaseType::PRIVATE)
    {
      OSCPrivateAction* pa = (OSCPrivateAction*)action;
      if (!pa->object_->containsEvent(event))
      {
        pa->object_->addEvent(event);
        break;
      }
    }
  }
}
```

Listing 3.2 One of the Esmini scenario engine modifications.

4

Controller Validation

4.1 Controller Validation Process

Controller Validation during the Design Process

Throughout the design process, various elements of the controller are being tested in order to sufficiently validate them. This especially includes testing the prediction algorithm, but also the code for the control objective after the switch. The scenarios used for this type of testing include the same scenarios that will be used for the final controller validation, see Section 4.1. These scenarios are fully defined using Esmini, and therefore data on all past, current and future motion of all vehicles is fully accessible. Because of this, and because smaller segments of the controller are being tested using debugging, this type of testing corresponds to white-box testing outlined in Section 2.3. The results are evaluated qualitatively rather than quantitatively, since this eases controller development and is sufficient in order to evaluate how well individual segments of the controller function.

Final Controller Validation

In contrary to the controller validation done during the design process, the final controller validation process takes a more rigorous and quantitative approach to evaluating the controllers overall performance. The validation process outlined here includes comparing the outputs of two different simulation models with the same input, and is hence quite similar to black-box testing outlined in Section 2.3. During the final controller validation, the ego vehicle is controlled by CSPAS. The contents and future outputs of CSPAS are unknown, which is why a black-box testing approach is used for the final controller validation.

CSPAS is a vehicle simulator developed by Volvo Cars that can execute autonomous driving and/or active safety functions. CSPAS controls all active safety (AS) functions for the ego vehicle, and Esmini hence serves as an environment for testing how well AS functions in CSPAS perform. The goal of the controller developed in this thesis is to control the target vehicle in Esmini scenarios such that it behaves in a realistic manner regardless of the behaviour exhibited by the ego vehicle. Since Esmini is used to test autonomous driving and active safety functions from CSPAS, it is feasible to evaluate the performance of the controller while controlling the ego vehicle using CSPAS.

Previously, the aim of the controller has been broadly defined as switching from relative to absolute control in order to ensure "realistic" behaviour by the target vehicle. However, a more precise metric rather than simply "realistic" behaviour is required in order to adequately evaluate the performance of the controller. During an Esmini scenario, the ego vehicle should react to various traffic scenarios, as well as to the behaviour of the target vehicle. The target vehicle should however largely exhibit constant behaviour regardless of what actions the ego vehicle performs. In other words, the behaviour of the target vehicle should remain as constant as possible within a given scenario, regardless of what behaviour other vehicles may exhibit. Therefore, the chosen approach for evaluating the performance of the controller is to analyze how consistent the behaviour of the target vehicle is. In order to do so, data on the state variables (i.e., x,y position and speed) are recorded for a given scenario where the ego vehicle does not perform any AS functions. The same scenario is then tested again, where the ego vehicle does make use of such AS functions such as adaptive cruise control (ACC) or collision mitigation by braking (CMBB). The state variables from both simulations are then compared. Speed and position error values between the two scenarios are calculated (in percent) as shown in Equation 4.1 and Equation 4.2.

$$Error_{pos} = \frac{\sqrt{(x_{noAS} - x_{AS})^2 + (y_{noAS} - y_{AS})^2}}{\sqrt{x_{noAS}^2 + y_{noAS}^2}} \times 100 \tag{4.1}$$

$$Error_{speed} = \frac{|v_{noAS} - v_{AS}|}{v_{noAS}} \times 100 \tag{4.2}$$

If the state variables from both simulations only deviate by small margins (i.e., the percentage errors are small), the controller is deemed to have achieved its aim of controlling the target vehicle in a realistic fashion. This approach builds on the assumption that the target vehicle exhibits realistic behaviour even if the ego vehicle does not make use of any AD functions. This is in fact the case for most scenarios defined using relative actions, as a lack of triggered AD functions will imply that the target vehicle will not react to any sudden motion by the ego vehicle.

Throughout the testing process, the controller is verified by testing the following active safety features and/or esmini actions in their respective scenarios:

- Collision Mitigation by Braking (CMBB)

- Adaptive Cruise Control (ACC)
- Scenario using the Synchronize Action
- Scenario using the Relative Distance Action

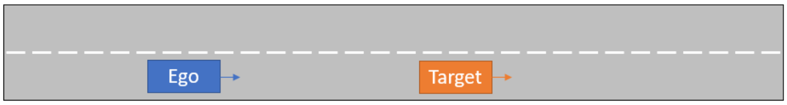
One reason for choosing these tests is to ensure that the controller functions with prevalent active safety functions included in CSPAS, such as ACC and CMBB. In addition, scenarios for testing the controller were chosen such that all three relative actions in Esmini are tested using the controller developed in this thesis. This includes the synchronize action, the relative distance action and the relative speed action. The relative speed action is tested when testing the CMBB and ACC scenarios. These scenarios are also tested using CSPAS, in order to test the controller with the autonomous driving software that Volvo Cars uses. The scenario using the synchronize action is however not tested using CSPAS. This is due to the fact that the scenario file is written in such a way that the ego vehicle would not behave as needed if controlled by CSPAS.

The scenario using a relative distance action was initially not planned to be tested with CSPAS. Instead it is tested manually using a similar method as with the scenario using a synchronize action, by triggering a hard brake manually through an action. An attempt for testing it with CSPAS was made, after doing some major modifications to the scenario. Unfortunately, due to a bug in the esmini code and lack of time, these tests could not generate any results and are hence not included in this report.

Collision Mitigation by Braking (CMBB) Scenario

This is the simplest scenario in combination with the least complicated safety function. It only contains two vehicles and the sequence of events can be seen in Figure 4.1. A dangerous situation will arise since the ego will catch up to the target.

Initial speed ego: 30 km/h
Initial speed target: 30 km/h



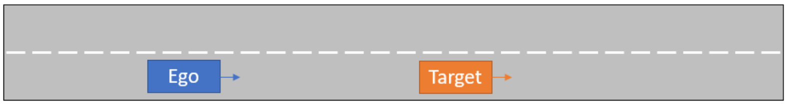
Scenario sequence:
Target:
At 1 second(s): Set target speed = ego speed – 1, (Relative speed)
Ego:
At Close proximity to target: Activate emergency brake function

Figure 4.1 Description of the CMBB scenario.

Adaptive Cruise Control (ACC) Scenario

Not considering the change of safety function, Figure 4.2 is the same scenario as the previous one with a minor difference in relative speed for the target.

Initial speed ego: 30 km/h
Initial speed target: 30 km/h



Scenario sequence:
Target:
At 1 second(s): Set target speed = ego speed – 0.5, (Relative speed)
Ego:
At a certain distance to the target: Activate adaptive cruise control

Figure 4.2 Description of the ACC scenario.

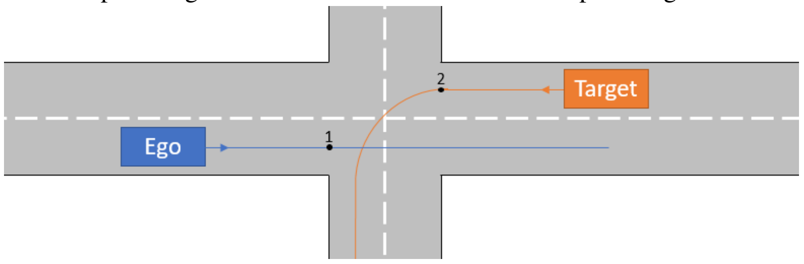
Scenario using the Synchronize Action

Scenarios using the synchronize action are more complicated to set up than normal scenarios. Because of this the testing for this scenario is conducted on one of the example scenarios in Esmini, which is modified in order to activate a CMBB function at a desired point in time.

For a switch to occur before the synchronize actions ends, the switch is required to take place before the vehicles reach point 1 and 2, respectively, as seen in Figure 4.3. The synchronize action ensures they both reach these points at the same time and will then be finished. Unfortunately, it is then clear that to test switching from a synchronize action one must induce a switch before there is any evidence that a dangerous situation will arise. This is why this scenario could not be tested with CSPAS. However, inducing a switch early has no implications for the performance of the controller since the prediction algorithm does not make any assumptions on where or when a safety function is likely to be activated.

Initial speed ego: 36 km/h
Initial speed target: 36 km/h

Final speed ego: 29 km/h
Final speed target: 29 km/h



Scenario sequence:

Target:
At 0 second(s): Activate synchronize action. Achieve final speed at point 2, when ego arrives at point 1

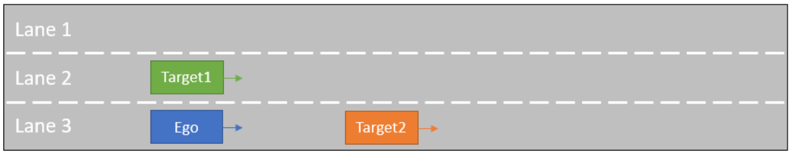
Ego:
At 2 second(s): Slow down to final speed
At 13 meters from intersection: Slow down to 0 m/s with a deceleration of $9m/s^2$ (Mimicked CMBB function)

Figure 4.3 Description of the synchronize action scenario.

Scenario using the Relative Distance Action

For the scenario in Figure 4.4, the switching controller is assigned to Target 2. This vehicle will try to keep a fixed distance to ego vehicle. The purpose of this test is to see how the controller performs while switching from a relative distance action, since new behaviour and target is based on what type of action was active. A dangerous situation will arise when Target 1 tries to switch lane, requiring the ego to react.

Initial speed ego: 80 km/h
Initial speed Target 1: 80 km/h
Initial speed Target 2: 80 km/h



Scenario sequence:
Target 1:
At 1 second(s): Change to lane 3 such that the lane change is completed in 6 seconds
Target 2:
At 0 second(s): Reach and keep 15 meters distance to ego.
Ego:
At lateral distance of 2.2 m to Target 1: Slow down to 0 m/s with a deceleration of $9m/s^2$ (CMBB)

Figure 4.4 Description of the relative distance action scenario.

4.2 Results

In Figure 4.5 and Figure 4.7, the speed, and in some cases the position, oscillate to a certain extent. This is due to the fact that CSPAS was not tuned properly to these scenarios prior to testing, and hence the ego vehicle (and by extension the target vehicle) displayed this oscillatory behaviour. However, the amplitude with which the speed oscillates is between 0.1-0.2 m/s, which was deemed to be quite low. Hence, these test results were used regardless, as the oscillation was too small to impact the behaviour of the target vehicle.

Collision Mitigation by Emergency Braking (CMBB)

Relative Error Graphs:
The relative error graphs (Figure 4.5) show the states of the target vehicle with- and without the use of the CMBB active safety function, as well as their relative errors. In Figure 4.5 it can be seen that the y-position of the target vehicle is identical for the active safety function scenario and the scenario without the active safety function. This is simply due to the fact that the target stays in the same lane during the entire scenario. The x-position deviates increasingly over time between the two scenarios, but the difference is quite small (only around 5% at most). The speed error is at first 0%, but after the triggering of the active safety function, the speed decreases in the scenario using the active safety function, thus resulting in a speed error of around 10%.

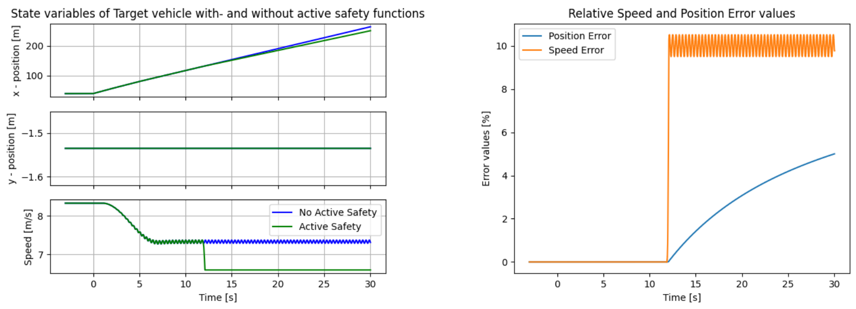


Figure 4.5 Relative error for the scenario run with & without CMBB function.

Prediction Graphs:
Figure 4.6 visualizes the prediction algorithm for the CMBB scenario. The figures show the predicted vs actual states for position and speed, respectively, as well as the errors generated from both speed and position. The active safety function is triggered at around 12 seconds, causing a rapid increase in both errors. Because the speed error exceeds the speed error threshold (denoted in green), the target vehicle switches from relative to absolute control. The position error was set to 1.8 m, and hence it is not used for switching in this scenario. This is also the reason why it is not visible on the error graph.

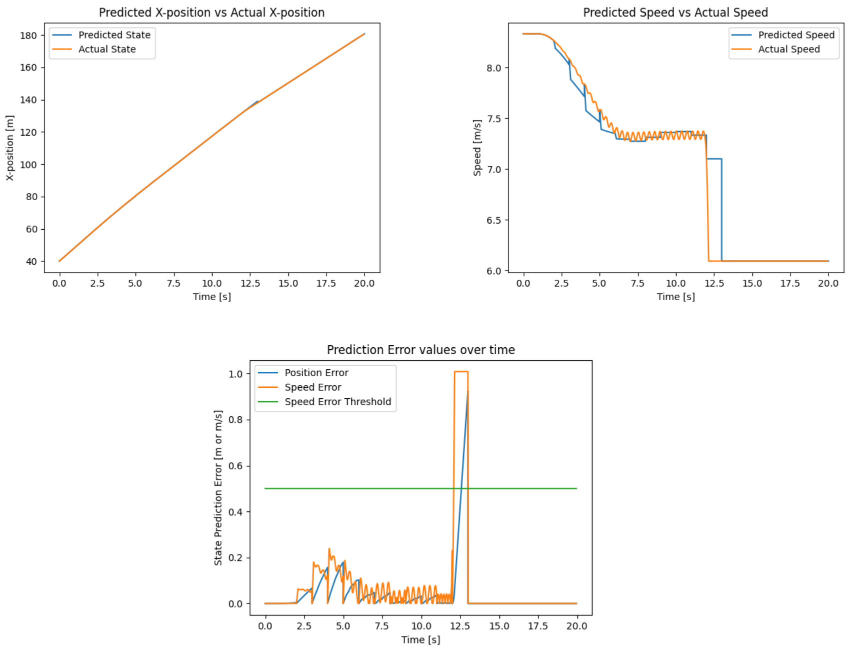


Figure 4.6 Prediction algorithm for a CMBB scenario.

Adaptive Cruise Control (ACC)

Relative Error graphs:

Similarly to the CMBB scenario, the y-position for the ACC scenario is constant regardless of whether or not the ACC active safety function is triggered. The x-positions deviate increasingly over time, reaching a maximum difference of 7% (see Figure 4.7). However, the ACC scenario differs from the CMBB scenario in that, in the version of the scenario with no active safety function, the speed decreases. In the version with an ACC active safety function the speed instead increases, leading to a speed error around 9.5% - 11%.

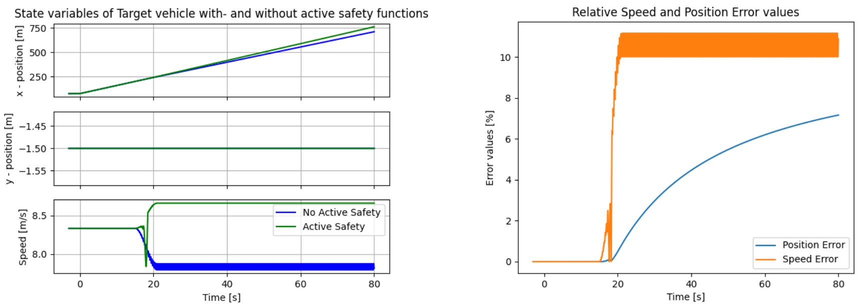


Figure 4.7 Relative error for the scenario run with & without ACC function.

Prediction graphs:
Figure 4.8 illustrates the prediction algorithm for the ACC scenario. The predicted x-position follows the actual x-position of the target vehicle with a near perfect accuracy. Both the predicted speed and the actual speed of the target vehicle experience a rapid decrease as the ACC function is triggered, followed by a sharp increase and a final value of around 8.6 m/s. The prediction however decreases to a value significantly lower, leading to a speed error peak of around 1.1 m/s. Since the speed error threshold is 0.5 m/s, the controller switches from relative to absolute control because of this. Because the prediction for position is quite close to the actual values, and the position error threshold is set to 1.8m, the controller does not use position error in order to switch.

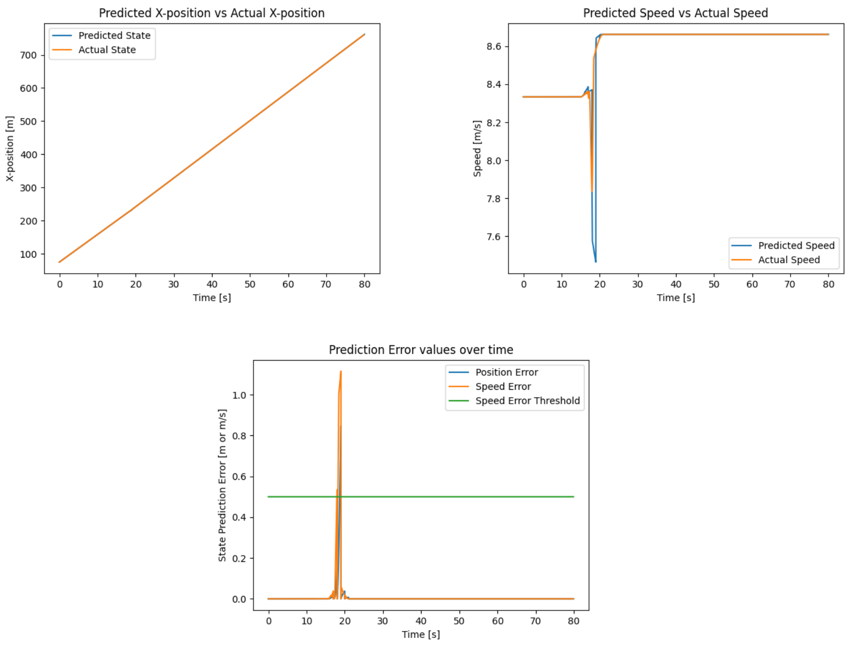


Figure 4.8 Prediction algorithm for the ACC scenario.

Scenario using the Synchronize Action

Relative Error Graphs:

In Figure 4.9 the x,y position of the version of the scenario with the active safety function follow those without the active safety function closely. The values for the y-position for the scenario without safety function are delayed by about 0.5 seconds behind the y-position for the scenario without. This is because the speed for the scenario with CMBB function drops significantly at around 6 seconds, before regaining the same speed as the scenario without CMBB function. In contrast to the "CMBB scenario", where the two vehicles were constrained by a relative speed action, here the speed changes very rapidly, creating a speed error of up to 70%. However, after the switch, the speed error drops to 0% and the position error starts to gradually decline.

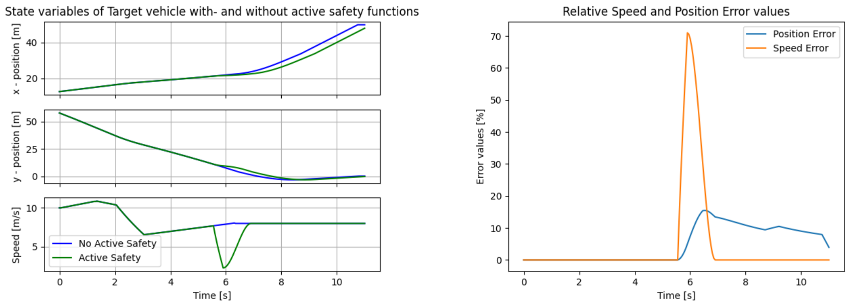


Figure 4.9 Relative error to the scenario run with & without an active safety function.

Prediction graphs:
The predictions for the position of the vehicle still follows the actual vehicle position somewhat closely compared to the predictions for speed. Position errors do however still reach values of around 2 meters. The synchronize action follows relatively complicated speed profiles, and hence the predictions for speed are not very accurate. This results in large speed error peaks, the first of which occurs before the CMBB function is triggered and has a value of 3.9 m/s. When the CMBB function is triggered however, the speed error becomes over 5 m/s, exceeding the speed error threshold and causing the controller to switch from relative to absolute control. It is worth noting that the speed predictions never follow the curve for the actual speed values. Instead, the prediction algorithm reads the value of the actual target vehicle speed at the start of each prediction, and predicts this same speed throughout the entire prediction horizon.

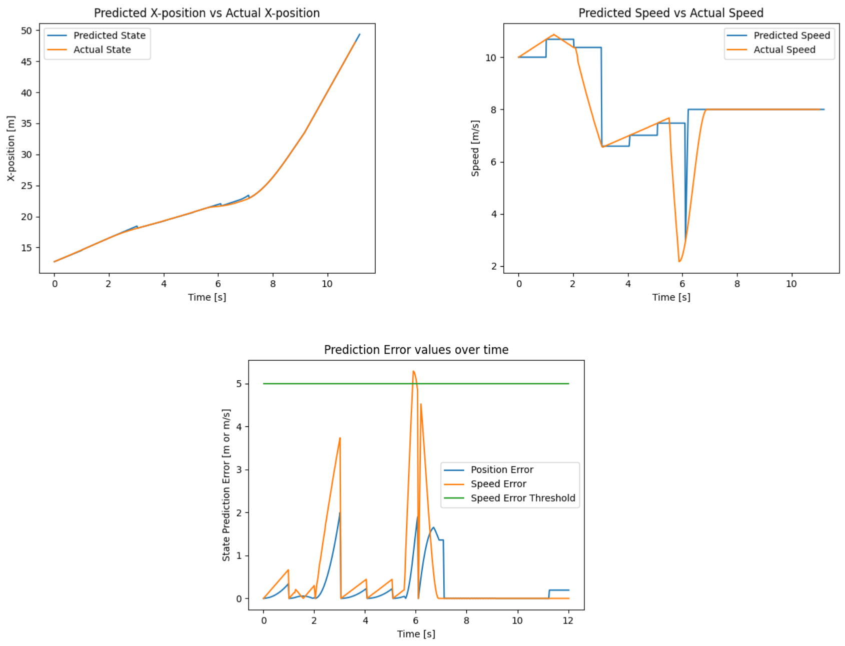


Figure 4.10 Prediction algorithm for the scenario using a synchronize action.

Scenario using the Relative Distance Action

Relative Error Graphs:

In Figure 4.11 the relative speed and position error values can be seen increasing to 4% and 3% respectively. In the graph for the state variables it is clear that a minor deviation from the x and y coordinate occurs. For the speed a static error can be seen compared to the run without a safety function. The active safety function engages at approximately 2 seconds of run time.

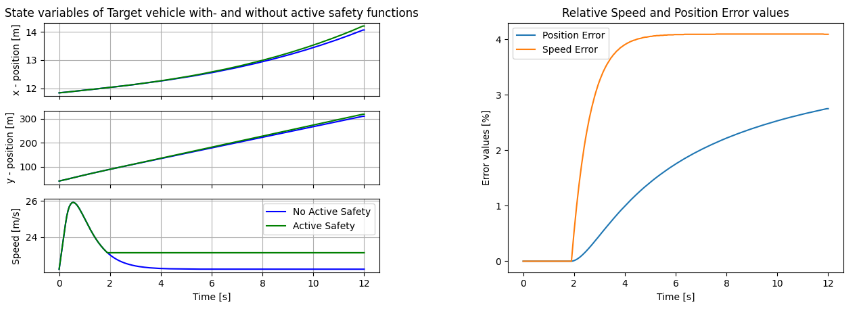
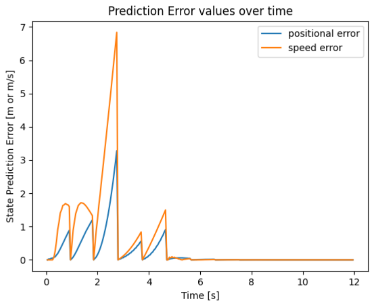


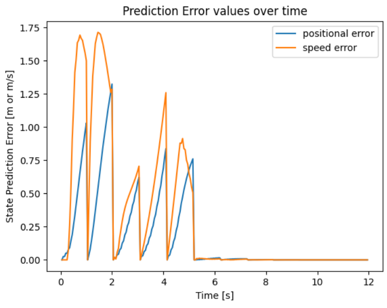
Figure 4.11 Relative error for the scenario run with & without an active safety function.

Error graphs with different prediction horizons:

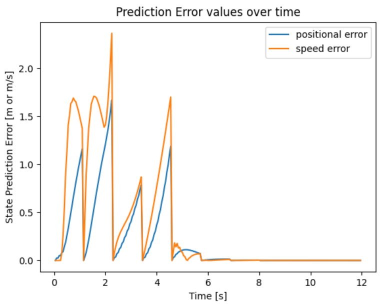
In Figure 4.12 the same relative distance scenario is shown three times with different prediction horizons. In graph a), with a horizon of 0.9 seconds, a large error occurs at roughly 2 seconds where the active safety function engages. In graph c) it is less noticeable and in graph b) it is difficult to determine where the active safety functions activates.



(a) Horizon = 0.9 s



(b) Horizon = 1.0 s



(c) Horizon = 1.1 s

Figure 4.12 Error values with different prediction horizons for a relative distance action CMBB scenario.

5 Discussion

5.1 Controller Concept and Prediction Algorithm

The controller concept was chosen so that the controller is able to switch from relative to absolute control in reaction to various types of active safety functions. The motion analysis controller was implemented as a first step, and showed satisfactory performance with the CMBB scenario. This concept did however receive the feedback from Volvo employees that the controller would only be capable of reacting to specific types of motion (such as rapid braking during CMBB). Hence, the prediction concept was developed in order to switch in case any "unexpected" behaviour was detected by the controller. This way, the controller can detect any active safety function, and not just those that had their kinematic behaviour programmed into the controller. Because of this objective, the controller was tested on a variety of different scenarios featuring different active safety functions and Esmini actions. The relative error graphs from the results (Section 4.2) show that the controller concept fulfilled its goal of reacting to various different types of active safety functions. The largest error generated during any of these scenarios is during the synchronize action, where the speed error rises rapidly before stabilizing after around 1 second. This is due to the fact that when an active safety function is triggered during a scenario using the synchronize action, the target vehicle will rapidly decelerate. The controller will switch as a result, and the target vehicle will accelerate in order to regain its desired speed. While this behaviour may not be perfectly smooth, the behaviour of the target vehicle is acceptable in order to adequately test active safety functions on the ego vehicle while avoiding circular dependencies between the two vehicles. Overall, judging by these results, the prediction concept was successful at determining when to switch from relative to absolute control in a variety of different scenarios.

The main drawback of the prediction concept for switching is the need for tuning the controller. The controller features three different tuning parameters, which are the prediction horizon, speed error threshold and position error threshold. Although all three parameters have default values (1 second, 1.5 m/s and 1.5 m respectively), tuning is necessary in order to adjust the controller to different scenarios. The primary method for tuning the controller when testing has been by changing the speed threshold and using the same value for the positional threshold. It is often beneficial since the difference in the position, i.e., the difference in the travelled distance, is the integral of the difference in speed. Hence, the speed error will always be more sensitive to changes in kinematic behaviour. Scenarios using longitudinal speed actions in combination with a CMBB or ACC function will likely experience satisfactory controller performance without tuning. This is simply due to the fact that a scenario of this type was used for rapid testing of the code while programming the controller. For other types of scenarios however, the controller will likely need at least some tuning. Users have to adjust three different parameters, and need at least some knowledge of how the switching algorithm works in order to tune effectively. In practice, users will try to use this controller as a means of ensuring reliable behaviour by the target vehicle, while testing active safety functions on the ego vehicle. Hence, excessive time spent tuning will likely come as somewhat frustrating to users who are more interested in optimizing the performance of the ego vehicle. In order to mitigate this problem, a tuning guide has been developed, where the optimal tuning parameters are given for each of the scenarios tested in this thesis. These parameters roughly correspond to those outlined in Section 3.2, and were determined through experimentation. In addition, the controller can be used for standard ACC and CMBB scenarios with the default values outlined in Section 3.2. Given that the controller and Esmini are intended to be used by engineers accustomed with programming, control theory and vehicle dynamics, this will hopefully significantly ease the tuning process for this controller.

5.2 Control Objective after the Switch

After the switch, the new control objective is defined using the latest action that was active for the target vehicle. The difficulty of defining the control objective greatly depends on what type of action this is. For the longitudinal speed action, CMBB and ACC scenarios, the solution is quite trivial since Esmini handles conversions from relative to absolute. As mentioned previously, there is also no need to create a new action. The shortcomings of this approach are that no history of the speed is saved, resulting in that the new target speed will be the current speed plus the relative component, defined by the action. If the switch is to happen when the target has started slowing down, there is no way of knowing that the absolute target speed, generated by Esmini, will be too small. Thus, it causes a static relative error in speed compared to when running the scenario without a safety function.

The approach for the relative longitudinal distance action is similar to that of the longitudinal speed action. Both solutions use current absolute speed as a base for the new speed. This results in the same weakness in both actions, which is that a static speed error is introduced if the switch is not instantaneous. However, the creation of a new action that is needed for a switch from this type of action has further implications. If the target vehicle is in a steady state, no change can be observed from the speed profile. But if the target is changing speed when a switch occurs, e.g., it did not switch quickly enough to not slow down with the ego, an abrupt change in the gradient of the speed curve will occur. This does not impact the end result in a major way since the new target speed will, by definition, be the current speed. Thus, it will transition from a negative gradient to zero in a single timestep or point, but it will stay at zero. This is not the case for the synchronize action. It can have a final speed specified for the target when it is created. If this is the case, the controller knows were the target should be and at what speed. If the target starts decelerating, it can then regain the lost speed. This results in a speed profile that often loses speed during the switch but it will regain it before the action ends. This can be seen in Figure 4.11b, where the newly created longitudinal speed action perfectly matches the run without active safety after a second.

5.3 Testing

Testing Process

The testing process has overall been fairly successful in extracting results that reflect on the controller's performance in a fair manner. Because the aim of the controller was originally both loosely defined and unconventional, finding a fair way to accurately assess the controller's performance was not trivial. The idea of measuring the relative error as described in Section 4.1 was chosen due to the fact that ideally, the target vehicle would display constant behaviour regardless of which safety functions the ego vehicle uses. This, in combination with the prediction graphs, gives an overview both of how the controller concept works for different scenarios as well as the kinematic behaviour of the target vehicle. Both of these insights are valuable in assessing the controller. In addition, the scenarios that were chosen for testing provide data for different useful test cases. This is because the CMBB and ACC active safety functions are the most commonly used active safety functions and the four test cases include each of the relative actions currently in existence in Esmini. These include the longitudinal speed action (ACC and CMBB scenario), the longitudinal distance action and the synchronize action. Other scenarios using the same relative actions with the controller may mimic the behaviour of the target vehicle in the scenarios tested in this thesis. Further testing would however be required in order to verify the extent to which this is true.

On the other hand, there are some drawbacks with how the testing process was carried out. The foremost of these is that only the ACC and CMBB scenarios were tested using CSPAS, while the other two scenarios were tested using only Esmini. Because the ultimate goal is to use the controller while testing active safety functions with CSPAS, this means that the controller's performance has not been fully tested for these scenarios. While the test results using only Esmini can still give a good indication of the controllers performance, better assessments could have been made if the controller was tested using CSPAS for all scenarios. This was not done due to the fact that the synchronize action scenario would not work with CSPAS, for the reasons explained in Section 4.1. Similarly, the longitudinal distance action scenario ran into issues when testing with CSPAS, and there was not enough time to adjust either of these scenarios such that useful data could be extracted when testing with CSPAS. This does ultimately mean that the data given for the longitudinal distance and synchronize scenarios only shows how the controller performs for an ideal scenario. More insight could however have been gained if these scenarios also had been tested using CSPAS.

ACC and CMBB Test Cases

Judging by the relative error graphs for the CMBB and ACC scenarios (Figure 4.7 and Figure 4.5), the controller was quite successful in assuring adequate behaviour of the target vehicle during these test cases. The relative position error is at most 5% for the CMBB scenario and at most 7% for the ACC scenario. The relative speed error was around 10% and 11-12% for CMBB and ACC respectively, which translates to an abolute speed error of 0.8 m/s for ACC and 0.7 m/s for CMBB. The error in speed for both of the scenarios can be explained by the control objective after the switch from relative to absolute control. The target vehicle is controlled such that it continues to drive with the speed that it has at the time of the switch. Figure 4.5 shows that if an active safety function is triggered by the ego vehicle, the target vehicle will react by slowing down, which triggers a switch. The target then continues with the speed that it has had at the time of the switch. However, if there is no active safety function triggered, the target simply continues at the same speed that it had previously. This generates a difference in speed between the two tests, which in turn causes a difference in position. However, this difference is hardly noticeable when running these scenarios in Esmini. The difference in speed and position that these results show do not hinder the evaluation of any active safety functions tested in Esmini with CSPAS. Hence, the controller is reasonably successful in ensuring stable behaviour by the target vehicle.

The prediction graphs for the CMBB and ACC scenarios (Figure 4.6 and Figure 4.8) illustrate the switching mechanism for the controller in each respective scenario. They also show the effect of controller tuning. The switch in both cases is caused by the speed error exceeding its threshold of 0.5 m/s. This difference in predicted speed vs actual speed is in turn caused by the ego vehicle triggering its active safety function, something that is not accounted for in the prediction. When, for instance, CSPAS triggers a CMBB function, the ego vehicle decelerates quite rapidly, causing a sharp rise in the speed error. The speed error is significantly higher at this point in time than previously, which is why the controller works very well for these types of scenarios. Tuning the controller to a prediction horizon of 1 second and a speed error threshold of 0.5 m/s ensures that the controller switches at an optimal time, but, the speed error threshold could be tuned between 0.2-1 m/s. Scenarios that generate prediction errors in speed or position earlier, before the active safety function is tuned, could however be significantly harder to tune. These earlier prediction errors could for instance be caused by either a new action that is triggered and not immediately incorporated into the prediction algorithm. The result of this would be that the controller either switches too early if the thresholds are set low, or does not switch at all if the thresholds are set too high. However, even in this case tuning could hopefully be used in order to mitigate this problem. Tuning the prediction horizon could change when the new action is incorporated into the prediction algorithm, and hence decrease the initial error. However, this level of tuning would likely require the use of prediction graphs and knowledge of how to tune the controller, and would ultimately not be very friendly. On the other hand, the most common active safety features tested with CSPAS are CMBB and ACC features. It is hence a satisfactory result that the controller functions as intended with these scenarios while using the default tuning parameters.

Scenario using the Synchronize Action

In Figure 4.9 the results from the scenario using synchronize action are shown. No oscillations in the error graph to the right occur since this scenario is run exclusively with Esmini. This means that no external software is controlling the ego. The relative speed error observed increases quickly to 70% due to the threshold being set high. This is since the ego is changing speed before entering the intersection, creating a large increase error that is not created by the mimicked safety function. The increase in error before the intersection can be seen in Figure 4.10, in the graph for prediction error values over time. This extra action that the ego is executing exists to make the scenario more complicated and it gives the ego the same speed as the target. Another interesting observation is that the position error never fully recovers as the speed does, even though the targets goal is to reach the old target at the same point. This is due to the fact that the controller didn't compensate for the lost distance it should have travelled when it was slowing down. The loss in speed when the switch was to occur can be seen in Figure 4.9, in the graph to the left. The graph also shows the loss in the x position compared to the run without an active safety function, or in this case without a action mimicking an active safety function. It is clear the position cannot be regained without achieving a higher speed for the vehicle. The small plateau at the end of the x position graph is likely due to the road ending and the vehicle stopping. Esmini in this case does not update the speed, which explains why it is left at the steady state it has achieved.

An apparent problem with running the controller without an external software is that the ego vehicle's actions, that are needed for it to move, will be included in the prediction simulation. This is seldom a problem for the test cases. The only time this could be a problem is if the active safety function engages at the time or just before a new simulation starts. This would result in the newly started action that mimics an active safety function is included in the prediction and thus, no error would be generated.

Scenario using the Relative distance Action

The problem mentioned above is what happened for the scenario using a distance action in Figure 4.12(b). At about 1.96 seconds the mimicked safety function starts, allowing for a short increase in speed error that is hardly noticeable in the graph. If the horizon is increased to 1.1 seconds, the speed error has sufficient time to increase further as can be observed in Figure 4.12(c) in the same figure. If the horizon instead is shortened as with Figure 4.12(a), the speed error will increase to approximately 7 meters per second. This is due to the fact that the simulation started just before the ego activated the action that is to simulate a safety function. Furthermore the speed error then drops significantly the next timestep, due to the new action, that is stopping the ego, being included in the simulation.

In Figure 4.11 the relative error values are shown in the graph to the right. In this scenario the relative error reaches a maximum of 4% for the speed and close to 3% for the position. This is due to the same reason discussed with the ACC and CMBB cases. When a switch occur the behaviour of the target is changed such that the speed error will stop growing. In this case the controller will switch when the target vehicle has a speed of roughly 23 meters per second. This results in the relative position error continuing to increase since the target is moving at a lower speed compared to the scenario where it didn't slow down due to a safety function.

From Figure 4.12 it is clear that the horizon is of high importance when tuning. To achieve adequate results, the horizon should be set such that the active safety function does not trigger in close proximity to a scenario time equivalent to a multiple of the horizon time. An example of this can be seen in Figure 5.1. However, the size of this time span, that is to be avoided, has not been investigated in this thesis.

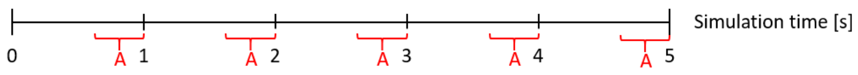


Figure 5.1 A five seconds scenario timeline with one second prediction horizons, showing an area, A, were it is undesirable to do a new prediction. Note that this figure is for demonstration purposes only and in no way indicates the exact time span that should be avoided.

6

Conclusion & Future Work

6.1 Conclusion

The aim of this project was to develop a controller to switch from relative to absolute control of the target vehicle when an active safety function is triggered by the ego vehicle. If the controller does this successfully, the target vehicle should exhibit constant behaviour, regardless of whether a safety function is triggered or not. A prediction algorithm was chosen as a concept for switching because it was believed to switch at the correct time for a wider range of different scenarios than alternative controller concepts. The results confirm that it is able to switch from relative to absolute control, at the time that an active safety function is activated, for four different scenarios commonly used in Esmini. Tuning using 2-3 different tuning parameters is however required for it to do this for the synchronize action and the longitudinal distance action. While tuning expands the controller's ability to switch correctly for a wider range of scenarios, it does so at the expense of user-friendliness.

The new control objective after the switch is absolute, meaning that the kinematic behaviour of the target vehicle will no longer be defined relative to any entity in the scenario after the switch. In a scenario with a safety function, where the target is controlled by a properly tuned controller, one can expect it to have similar kinematics after the switch as in the same scenario without any safety function. This is true assuming no major dynamic events occur before the switch is supposed to take place. The controller could however switch early if major dynamic changes do take place before the triggering of an active safety function, as this could generate a prediction error higher than the specified thresholds. In this case, the target vehicle may experience undesired kinematic behaviour. However, with correct tuning, the switch will likely occur at the correct time, that is slightly after an active safety function is triggered. If this is the case, the control objective after the switch leads to overall realistic behaviour of the target vehicle.

6.2 Future Work

The controller developed can be further analyzed by testing a larger variety of scenarios using CSPAS, including the longitudinal distance and synchronize scenarios. This is a time consuming process that includes several steps and requires the assistance of CSPAS developers for some of these steps, including executing the tests. However, further testing on a wider variety of different scenarios would give a better overview of how well the controller performs in different conditions. This may be quite valuable, as the controller may be intended to be used for scenarios not tested in this thesis. Further testing would validate the controller for a wider range of scenarios, as well as finding the right tuning parameters for each of these scenarios.

Furthermore, improvements can be made to the controller and its prediction algorithm. A possible improvement to the prediction algorithm is to incorporate more data from the actual Esmini simulation in order to make predictions. Since the controller has access to all data associated with a scenario except for the ego vehicle's behaviour, it could be of interest to restart the prediction when a new action is started. This could possibly minimize large unwanted errors that can cause an premature switch. In addition, it could significantly ease tuning for some scenarios. It does however make the prediction horizons inconsistent which could cause other problems. Overall, the prediction algorithm should ideally predict the target vehicle's states with a high degree of accuracy until an active safety function is triggered, at which point a large error should cause the controller to switch. Incorporating more data from Esmini into the controller therefore has a lot of potential to improve the controller's overall performance, as well as make it more user friendly.

Bibliography

Andersson, M. (2021). *Scenariogeneration*. URL: https://www.overleaf.com/project/6005485827ab87a34c0efd7a (visited on 2021-03-24).

ASAM OpenSCENARIO (2021). URL: https://www.asam.net/standards/detail/openscenario/ (visited on 2021-01-25).

Balci, O. (1995). "Principles and techniques of simulation validation, verification, and testing". URL: https://ieeexplore-ieee-org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=478717 (visited on 2021-04-01).

Figueiredo, M. C. (2009). *An approach to simulate autonomous vehicles in urban traffic scenarios*. URL: https://core.ac.uk/download/pdf/143407198.pdf (visited on 2021-03-26).

Jillre and Jesong (2016). *Uml class diagrams: reference*. URL: https://docs.microsoft.com/sv-se/previous-versions/visualstudio/visual-studio-2015/modeling/uml-class-diagrams-reference?view=vs-2015 (visited on 2021-03-20).

Knabe, E. (2021a). *Environment simulator minimalistic (esmini)*. URL: https://github.com/esmini/esmini (visited on 2021-01-18).

Knabe, E. (2021b). *Esmini uml diagram*. URL: https://viewer.diagrams.net/?highlight=0000ff&layers=1&nav=1&title=esmini_class_diagram.xml#Uhttps%5C%3A%5C%2F%5C%2Fdrive.google.com%5C%2Fuc%5C%3Fid%5C%3D1z5TM6o-RryOGl1l-lRgiiv9ZMcrw0jBZ%5C%26export%5C%3Ddownload (visited on 2021-06-07).

Litman, T. (2021). *Autonomous vehicle implementation predictions implications for transport planning*. URL: https://www.vtpi.org/avip.pdf (visited on 2021-03-26).

Robinson, S. (2004). *Simulation: The Practice of Model Development and Use*. John Wiley Sons.

Sargent, R. (2004). "Validation and verification of simulation models". URL: https://ieeexplore-ieee-org.ludwig.lub.lu.se/document/1371298/authors#authors (visited on 2021-06-08).

Thaler, J. (2021). *Computer simulation*. URL: `https://homepages.fhv.at/thjo/lecturenotes/sim/index.html` (visited on 2021-06-08).

Umit Ozguner, T. A. and K. Redmill (2012). *Intelligent and autonomous road vehicles*. URL: `http://citr.osu.edu/ECE/osuact/class/ControlIssues.pdf` (visited on 2021-03-26).

Appendices

A OpenSCENARIO xml code

```
<Storyboard>
  <Init>
    <Actions>
      <Private entityRef="Ego">
        <PrivateAction>
          <LongitudinalAction>
            <SpeedAction>
              <SpeedActionDynamics dynamicsDimension="time" dynamicsShape="
                  step" value="0"/>
              <SpeedActionTarget>
                <AbsoluteTargetSpeed value="8.333333333333334"/>
              </SpeedActionTarget>
            </SpeedAction>
          </LongitudinalAction>
        </PrivateAction>
        <PrivateAction>
          <TeleportAction>
            <Position>
              <LanePosition laneId="-1" offset="0" roadId="1" s="25"/>
            </Position>
          </TeleportAction>
        </PrivateAction>
        <PrivateAction>
          <ActivateControllerAction lateral="true" longitudinal="true"/>
        </PrivateAction>
      </Private>
      <Private entityRef="Target1">
        <PrivateAction>
          <LongitudinalAction>
            <SpeedAction>
              <SpeedActionDynamics dynamicsDimension="time" dynamicsShape="
                  step" value="0"/>
              <SpeedActionTarget>
                <AbsoluteTargetSpeed value="5.333333333333334"/>
              </SpeedActionTarget>
            </SpeedAction>
          </LongitudinalAction>
        </PrivateAction>
        <PrivateAction>
          <TeleportAction>
            <Position>
              <LanePosition laneId="-1" offset="0" roadId="1" s="60"/>
            </Position>
          </TeleportAction>
        </PrivateAction>
        <PrivateAction>
          <ActivateControllerAction lateral="true" longitudinal="true"/>
        </PrivateAction>
      </Private>
    </Actions>
  </Init>
```

Appendices/pythonscenario2.xosc

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* June 2021 *Document Number* TFRT-6139
Author(s) Olof Karlsson Erik Fredin	*Supervisor* Emil Knabe, Volvo cars, Sweden Mikael Andersson, Volvo cars, Sweden Angel Molina Acosta, Volvo cars, Sweden Anton Cervin, Dept. of Automatic Control, Lund University, Sweden Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Development of a controller to switch between relative and absolute path for target vehicles in simulation scenarios

Abstract

Simulating vehicle scenarios for autonomous cars saves both time and money compared to physical testing. For this to work efficiently a sophisticated simulation software is often needed. Esmini is an open-source, lightweight vehicle simulator that can handle advanced traffic scenarios. When designing scenarios to be used for testing autonomous vehicles, it is often beneficial to set up relative vehicle behaviour to ensure that a hazardous situation arises. This could be that the car in front of the autonomous car drives a couple of kilometers per hour slower compared to the autonomous car. At some point the autonomous car must react to avoid a collision, e.g., by braking. To avoid having the car in front slowing down when the autonomous car brakes, a switch must be done such that the car in front has an absolute speed.

Hence, the purpose of the controller developed in this thesis is to decide when to switch from the relative setup and what the new control objective should be. This is implemented by simulating a short period into the future to observe how the scenario develops. The controller generates predictions using simulation data of the events that should transpire for every vehicle except the autonomous one. The expected vehicle states are then continuously compared to the actual vehicle states generated during the simulation. If the actual vehicle states deviate too far from the expected values, a switch occurs. A new objective is then given to the vehicle, depending on its behaviour before the switch and what type of relative setup previously existed.

The proposed solution is available in the official Esmini repository and has been tested in traffic scenarios with CSPAS, a platform where Volvo's active safety functions are simulated. The performance of the controller depends on how it is tuned. For all tested scenarios a satisfying result has been achieved.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316	*ISBN*

Language English	*Number of pages* 1-72	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/