Delft University of Technology
Department of Cognitive Robotics
Bachelor Thesis

AVA, a prototype for speech-based assistance in automated driving

Gitte Hornung (4443535)
Daan Koetzier (4460979)
Tessa Talsma (4365631)
Bob van der Windt (4445503)
Supervisors: Dr. Ir. J. de Winter and Ir. J. Stapel

December 19, 2019

Abstract

In this paper, a speech-based assistant for partially automated driving is presented. AVA, the acronym for *Automated Vehicle Assistant*, has the goal of increasing the situational awareness of the driver to provide a better possibility of manual interference when driving in partially automated vehicles. The system does this by talking about objects in the vicinity of the car. This design-based research is based on iterations followed by test runs and offline reiterations. First, the problem of prioritizing objects to mention is tackled and later logic and the wishes of people are simulated. System relevance and understandability are then evaluated with human evaluation. Participants in possession of a driving licence are presented pictures about specific situations and prioritize objects, with the aim of getting the algorithm to generate the same prioritization. In the end, the prototype is able to mention relevant objects by filtering out stationary objects, give priority to objects that rapidly appear into vision, depend the detection-speech box on its own speed and provide special attention to vulnerable road users. Furthermore, the possibility of the implementation of eye-tracking is evaluated through the relevance of the commands of the system. At last, the final prototype of AVA is demonstrated by means of an accompanying video.

1 Introduction

Automated vehicles promise to drastically reduce road accidents, improve traffic flow, and reduce fuel consumption and emissions in the future [17]. However, for economical, technical, legal and psychological reasons, vehicle automation is not directly available to the market. It is incrementally introduced through advanced driver-assistance systems (ADASs) such as adaptive cruise control, intelligent speed adaptation, lane-keeping assist systems and lane-change decision aid systems. These systems rely on a limited number of hardware components that allow partially automated driving on the public road [16].

Currently, partially automated vehicles still require a constantly attentive driver and staying vigilant has proven to be a challenge [3]. Keeping a driver situationally aware in automated vehicles with the aim of giving the possibility of successful manual intervention, could be achieved by a system acting as a co-driver. It can be assumed that appropriate feedback reduces the time needed for a successful takeover as it could allow the driver to anticipate the need to intervene [3]. A speech-based system that can talk to the driver about the objects it sees in its environment will thereby also create understanding in the perception of the vehicle and thus trust in the automation [17].

1.1 Speech-based assistance

A comparable system has already been developed in 2019, namely the on-demand intersection-assistant CORA. Researchers from Toyota have created an interactive system, which allows the driver to literally call on CORA and request her assistance, whenever the driver has difficulty overseeing the traffic situation at an intersection. The system will then give information on the traffic on the intersection and possible gaps for entering the intersection. Hereby, CORA already gives the driver the ability to partially transfer the task of monitoring the traffic and solves the shortcomings of earlier sensor technology in establishing an adequate environment perception [9]. Shortcomings of the technology are that the system is only focused on intersections and dangerous situations can still occur when a driver has not yet called on CORA for help.

1.2 Full assistance co-driver

The goal of the present study is the design of a talkative, human like, co-driver that aims to increase the situational awareness of the driver by giving constant information about the surrounding traffic and road users. The psychological experience of people with this system, is left out of this research. Only human logic on the visual experience of people, is in scope. This prototype is from now on called AVA, an acronym for *Automated Vehicle Assistant*. This design is based on iterative tests and is later validated using the wide array of data recorded during test runs with a modified Toyota Prius, made available by the Delft University of Technology.

This paper will start by listing the requirements for a successful co-driver in section 2, where the method will also be explained, consisting of iterations, test runs and offline reiteration. After that, the design choices will be presented in section 3. The results, accompanied by detailed explanations on each iteration, will be described in section 4. Section 5 validates the final system one more time and a demonstration will be presented, in section 6 the discussion and future recommendations can be found and lastly the conclusion in section 7.

2 Method

2.1 Materials

In the following subsections, the materials originally provided for this research are presented.

2.1.1 Toyota Prius

A Toyota Prius, modified in 2015, equipped with automatic transmission and implemented with eye-tracking cameras and stereo vision cameras, has been used for the development and testing of AVA. Using a differential GPS, the car can determine its position and movement with respect to the world.

2.1.2 Object detection

The stereo vision cameras that have been implemented in the car, use visual odometry and have an upper limit of 10 Hz. The system is able to recognize and distinguish different road users and compute their coordinates. The cameras of this vehicle cover a total angle of 60° in which

objects can be detected. A known disadvantage of this system, is that it uses cameras to distinguish the object, so image deficiencies can lead to faults in object detection. Additionally, concerning cyclists, the system detects the bicycle and the person riding it individually. As a result, a cyclist can be erroneously mentioned as a person.

2.1.3 Smart Eye

In order to detect the direction of the drivers gaze relative to the direction of the car, the vehicle is equipped with a Smart Eye eye-tracker. Funke [6] compares five different eye tracking systems, where the Smart Eye has the highest percentage of usable data. A deficiency is that if the face can not be properly distinguished, the gaze data will falter.

2.1.4 Code state of the art

Due to previous research at the department of Intelligent Vehicles, at the Delft University of Technology, a previously written ROS (*Robot Operating System*) node was made available for this study. This provided code [15] was already able to (1) detect the type of object, (2) determine the location of an object in the world-frame and car-based frame, but not yet the velocities of these objects and (3) determine the angle of the drivers gaze via gaze data. The provided code is marked in appendix E (line 434 - 810).

2.2 Requirements

Based on the existing capabilities of already existing speech-based assistance systems in cars, such as navigation applications and CORA [9], the largest challenge of a constantly assisting co-driver is to solve the aforementioned problem of staying vigilant and trying to keep a driver situationally aware. Therefore, the additional requirements established for this prototype are:

(i) The co-driver must only provide information on relevant and active road users.
(ii) The co-driver must provide information about its surroundings in a timely manner.
(iii) The co-driver must provide messages that are in line with the thinking process of the driver.
(iv) The voice of the co-driver must be intelligible.
(v) The co-driver must provide information comprehensible for the driver.

2.2.1 Desirables

It is also desirable to come to a fully professional and complete prototype. Therefore the desirables in this study are established as follows:

(vi) The driver is able to directly distinguish the mentioned object by the co-driver, from other objects in its surroundings.
(vii) The system acts and talks like a human co-driver would and has a pleasant voice.

2.3 Measurement protocol

AVA will be optimised via the measurement protocol described in the following subsections and as visualised in figure 1.

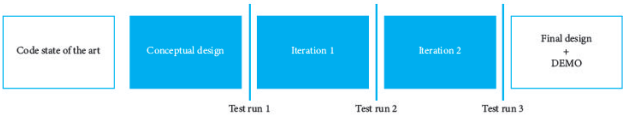


Figure 1: Overview of the iterations and versions

2.3.1 Iterations

Firstly, the existing capabilities of the provided ROS node, as mentioned in 2.1.4, will be improved and built upon to create a conceptual design. In this design, the co-driver should be able to detect objects, along with their location and urgency, and notify the driver about the approximate location of the most urgent object in traffic.
Following that, the first iteration will improve the functionality of the system by filtering out irrelevant objects and mentioning objects in time. The data flow as described later in this paper will be left untouched in this version.
The second and last iteration of this design, will improve upon the wishes of people and human logic will be added to the choices that AVA makes.
The iterations lead to a final design. Within this design the possibility of using eye-tracking will be explored.

2.3.2 Test runs

As can be seen in figure 1, each version of AVA is succeeded by a test run in the Toyota Prius. Appendix A contains the routes for the test runs in between iterations. All routes contain at least an

intersection, a roundabout, a section with high speed and one with low speed, necessary to test outcomes of AVA.

2.3.3 Reiteration

The test runs are recorded and can be evaluated at any time by reviewing the data in a ROS bag. A ROS bag is a package from a real test drive, consisting of eye-tracking data, object detection data, sensor data and camera footage. As the system makes no distinction between the bag and a real test drive, the system can be verified and optimised, without a new physical test run. From the bag a simplified representation can be established, as shown in figure 2, containing the location and type of the detections and a gaze vector, simulating the direction in which the driver is looking.

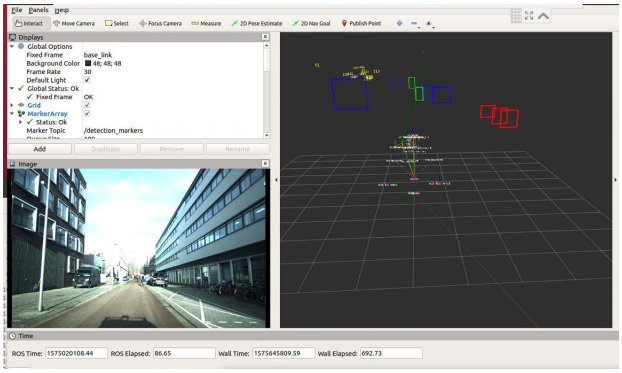


Figure 2: A screen capture of a test run in ROS

2.4 Validation

Requirement (i), (ii), (iii) and (vi) will be evaluated for each test run in the manner explained in the following section. Requirement (iv), (v) and (vii) will be validated through the literature study presented in section 3.

2.4.1 (i) The co-driver must only provide information on relevant and active traffic users

Objects are observed to be irrelevant because of two reasons: (1) the object is a false positive or (2) the object is not an active traffic participant. A false positive is defined as when AVA gives a message about an object that has been detected, but that object does not truly exist. Secondly, examples of inactive traffic participants are parked cars and bicycles. It is aimed that the system has a percentage of 85% relevant messages.
This requirement will be tested after each test run by counting both the false positives and the mentioned inactive traffic users in the recorded data. The first is done using the recording of the test drive to accurately determine if it is truly a false positive. The latter is done by extracting the detected parked cars from the test drive data and comparing it to the list of mentions by AVA.

2.4.2 (ii) The co-driver must provide information about its surroundings in a timely manner

An object is considered to be mentioned in a timely manner if a collision can be avoided. According to McGehee [13] in 1 second a driver is aware of a new situation and within 1.5 seconds a driver is able to make a decision to avoid an accident. AVA is designed as an informational system and not necessarily as a warning system, so the driver would seldomly be necessitated to take action. Therefore 1 second is used as the minimal value for the time to collision. The time to collision is defined as how long it will take to collide with an object if no action is undertaken and can be calculated with the distance to an object and the current vehicle speed. If the time to collision is greater than 1, an object is seen as mentioned in a timely manner.
This elaborate validation will only be done for the final design. For the iterations, an object will be seen as untimely if it has passed at the time AVA mentions it.

2.4.3 (iii) The co-driver must provide messages that are in line with the thinking process of the driver.

This requirement is validated by means of a questionnaire with 6 pictures of traffic situations encountered on the third test run. Participants are asked to determine the order in which they would like to have the traffic objects mentioned by a co-driver. The objects are specified by the colour of the surrounding box and a short description, e.g. "the car in the left lane". An example of a situation can be found in figure 3. The questionnaire is taken among 47 participants. Each one of the participants is in possession of a driving licence and currently studying at the Delft University of Technology. The complete validation questionnaire can be found in

appendix C.1. During the validation, a fraction of direct commands of AVA could not be determined due to a data gap in the recorded eye tracking. More about this will be described in section 6. The urgency was manually determined according to the logic of AVA.

Figure 3: Question 5 from the second questionnaire

2.4.4 (vi) The driver is able to directly distinguish the mentioned object by the co-driver, from other objects in its surroundings

To measure the achievement of this desirable, six additional questions are asked in the above mentioned questionnaire. The participants are given a situation, such as in figure 4, along with the message given by AVA, in this case "person left". Hereafter, they are asked to indicate which object AVA is referring to. The objects are specified in the same manner as the questions from section 2.4.3. The remaining questions can be found in appendix C.2.

Figure 4: Recognition question from validation questionnaire

3 Design choices

In this section, the choices preceding the iterations and test runs are explained through literature. Each referring to a requirement in section 2.2.

3.1 Information transfer

Although visual notification is extensively used as a form of communication in the automotive industry, auditory information has proven to be a better alternative to visual communication in the automotive context [9]. As the system only communicates via speech, the driver can fully focus their visual attention on the actual driving task [9]. Additionally, for systems giving low-urgency feedback to the driver, auditory signals are received as most preferred [1]. Out of the various auditory signals, spoken messages are more accepted than abstract sounds [1]. From this, it is concluded to let AVA give verbal messages to the driver.

3.2 (iv) The voice of the co-driver must be intelligible

In the first version of AVA, eSpeak is implemented, which is an open source software speech synthesizer. eSpeak uses a formant synthesis method to provide the basics of a large variety of languages with limited fluency [4]. After iteration 1, the second version of AVA is implemented with Google API [14]. Google API can articulate better and sounds more realistic because of WaveNet, a deep neural network for generating raw audio waveforms. The model is probabilistic, autoregressive and can be trained on data, therefore providing a more human-like pronunciation. We can conclude that the Google API speech engine is more intelligible than other open sourced engines.
For further intelligibility, the speech engine is kept from interrupting itself. This is done by stopping the speech engine from starting a new command, until the previous command has been said.

3.3 (v) The co-driver must provide information comprehensible for the driver

Listening comprehension can be divided into the speech rate, the number of items per message and the interference period.
Firstly, it is proven that listening comprehension will

not be affected drastically by a word rate from 125 to 250 words per minute, but it declines rapidly above 250. This is because time is required for the listener to perceive the words [8]. The applied Google speech engine speaks with a rate of 180 to 220 words per minute. According to the aforementioned range, AVAs speech rate is comprehensible.
Secondly, an experiment explained by Fleming [5] states that error rates, defined as drivers not recalling the full auditory message while driving, are linearly related to the number of units in the message. Perceptual memory, involved with the immediate readout of information, for example when people glance at something, has a typical capacity of 5-17 items. The research concludes that a brief message is preferred by drivers. Typically the messages of AVA contain 2 to 3 basic items, as seen in figure 5. This is considered a comprehensible number of items and additionally keeps the messages from getting too long in a way that objects could be missed, because AVA is already speaking.
Lastly, the interference period is defined as the silent period between two messages. For AVA it is fixed at 2.5 seconds, because, as mentioned before, objects could be missed and a longer interference period would not be able to adapt to rapidly changing traffic situations.

Set of announcements from AVA
cyclist right
person in front
car left
bus right

Figure 5: Set of possible announcements of the system

3.4 (vii) The system acts and talks like a human co-driver would and has a pleasant voice

To make AVA have a pleasant voice, a female voice is used. Although for various reasons, female voices are found to be less annoying than male voices concerning take-over requests in automated driving [1]. Secondly, making AVA sound human like is achieved by using the Google speech engine. Because the Google API engine uses WaveNet voices generated by human voices, it strongly mimics the human voice [14]. To make AVA come across as even more human like, the system introduces itself when the car is started.

4 Results

4.1 Conceptual design

This section systematically presents the data flow of the first version of AVA. Essentially, AVA must be programmed to process the object detection and eye-tracker data along with the location, type, velocity and gaze angle of each object, called track data. Finally, it must generate a message from this data. This data flow is visualised as a block diagram in figure 6. This structure is maintained throughout all iterations. For the extensive code, see appendix E. In the subsections is also referred to lines of code where the functions can be found.

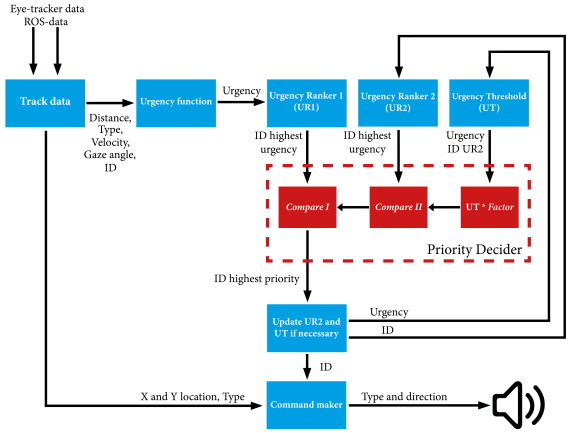


Figure 6: Block diagram of data flow

4.1.1 Urgency function

Once an object has been detected, it is stored together with its track data. The object is then given an urgency value by the urgency function (appendix E line 71-164), shown in equation 1. An urgency is necessary to label objects with, to sort objects on their relevance, as mentioned in requirement (i). The function consists of: a type-urgency reflecting the vulnerability of the object in traffic; a velocity-urgency proportionate to the absolute velocity of the object and a distance-urgency inversely proportionate with the relative distance of the object to the car multiplied by factors a, b and c respectively, figure 7. These factors are determined by looking at traffic situations. It is a possibility to include the gaze-angle in the urgency, section 5.3.2.

$$U = a \cdot Typeurgency + b \cdot VworldAbs + c \cdot DistanceUrgency \quad (1)$$

Multiplication factors	a	b	c
Value	3	4	60

Figure 7: Urgency function values

4.1.2 Urgency rankers

The object and its urgency are then stored in the designated urgency ranker (UR). UR1 features objects not yet mentioned by AVA. When an object is mentioned it is sent to UR2 and the urgency at that time is stored in the urgency threshold (UT). Objects that disappear from the view of the camera and thus are irrelevant, are deleted at this point.

4.1.3 Priority decider

Subsequently, the priority decider compares the urgencies from UR1, UR2 and UT (appendix E line 273-300). Hereby, it prioritises the object with the highest urgency. An object in UR2 will only be mentioned if it exceeds its previous maximum urgency in UT and this urgency is higher than the one from UR1. This means that new objects will be mentioned first and already mentioned objects will only be mentioned again if they become more urgent, e.g. by a sudden change of speed.

4.1.4 Command maker

The object with the highest priority is sent to the command maker along with its track data (appendix E 354-365). From this track data its direction is calculated, using the angles in figure 8 (appendix E line 234-260). Its type and direction are turned into text (appendix E line 367-392) and the message is then ready to be pronounced by AVA (appendix E line 394-424). The frequency of messages can be regulated by the frequency of the triggering of priority decider with a maximum of 10 Hz, the frequency at which the object detection system gathers data.

Angle to object	Direction command
$\alpha <$ -20°	Left
-20° $< \alpha <$ 20°	In front
20° $< \alpha$	Right

Figure 8: Table of direction angles and their commands

4.2 Validation of conceptual design

This iteration is tested in the first test run. The driven route can be found in appendix A.
(i) Only 30 seconds of the route are seen as relevant, because this was the only part with parked vehicles. Out of 11 messages 5 concern parked cars, as can be seen in figure 14. These vehicles are not active traffic participants and thus irrelevant. Furthermore, some objects mentioned are so distant that the driver is not able to distinguish them. With no other major deficiencies, it can be concluded that the block diagram in figure 6 works as desired.
(ii) Road users that have already passed before they can be seen by the driver, are still mentioned.
(iv) The voice is not experienced as comprehensible. The words are not properly articulated and rushed.
(vii) Currently eSpeak is implemented. The voice of the AVA is experienced as robotic and as having no intonation.
Requirements (iii), (v) and (vi) are not addressed in this concept, but later on in the results.

4.3 Iteration 1

This iteration focuses on improving the functionality of the co-driver. The following subsections will explain different solutions to the main problems: (1) filtering out parked vehicles and streaks of oncoming traffic, (2) filtering out too distant objects, (3) providing information in a more timely manner to prevent passed objects from being mentioned and (4) the unintelligibility of the system as noticed in test run 1. The solutions correspond to requirement (ii), (i), (ii) and (iv) respectively.

4.3.1 (i) Stationary vehicles

In the conceptual design parked vehicles did receive a lower urgency from the urgency function 1 because of their low velocity, but due to there being no other traffic around and erroneous detection of velocities, they were still mentioned.
Stationary vehicles are often lined up and this can be used in a solution, additional to the speed detection system. Objects detected with a absolute velocity $\leq 3m/s$, will be inserted in a streaklist, a list consisting of the coordinates of all parked cars. If the following detected object has a maximum distance Δx of 1 meter and Δy of 3.5 meter to the previous object, figure 9, this object will not be mentioned.
In a similar way, streaks of oncoming traffic can be

filtered. Objects detected with a negative relative velocity and a maximum Δx of 0.5 meter and Δy of 2.5 meter from the first oncoming vehicle, are not mentioned (appendix E line 119 - 146).

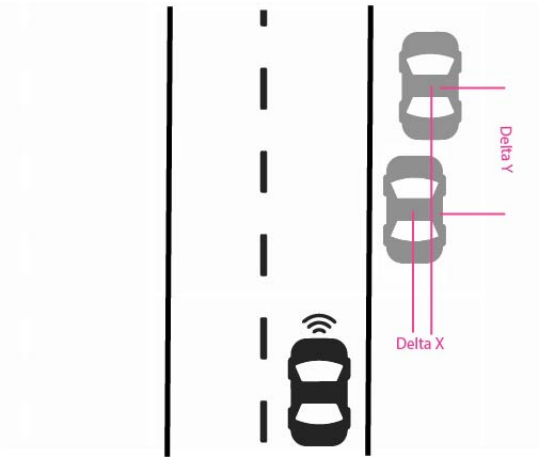


Figure 9: A visualisation of the streaklist

4.3.2 (i) Very distant objects

Objects that are indistinguishable for the driver, are filtered out by implementing a speech box. A speech box is defined as an area wherein a message about the object may be generated. The maximum horizontal distance is set to 15 meters on both sides of the car. The maximum vertical distance in front of the car is velocity dependent and set at $8 \cdot vehicleSpeed$.

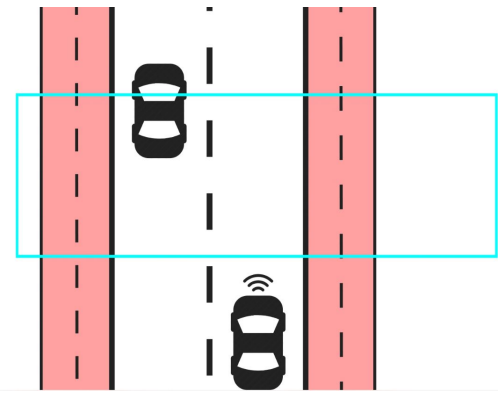


Figure 10: A visualisation of the detection distance

4.3.3 (ii) Passed objects

To present messages in a timely manner, an imaginary line has been placed in front of the car. The distance of this line is placed at $1 \cdot vehicleSpeed$, as is explained in section 2.4.2. Only if objects are situated in front of this line, AVA may generate a message. Combined with section 4.3.2, this gives a speech box as shown in figure 10

4.3.4 (iv) Intelligible messages

Due to the voice being unintelligible during test run 1, the choice is made to implement Google API as a speech engine. This differs from the initial choice of using eSpeak, as explained in section 3.

4.4 Validation of iteration 1

Iteration 1 is tested in the second test run. The route that is used can be found in appendix A. Only 10 minutes of representative footage is evaluated. There were harsh lighting conditions during the test run, complicating the object detection.
(i) As can be seen in figure 14, 19 out of 71 messages concern stationary cars. It can also be seen that there are 8 false positives in the 10 minute data analysis. The message are experienced as more relevant and with no changes in the traffic situation AVA is silent. However, the user driving the test run still experiences it as many messages.
(ii) Out of the 19 times AVA reports a person, that person has already passed 3 of those times. Other messages are in a timely manner and the distance to the object at the moment of a command is large enough to react in time.
(iii) AVA mentions vehicles on the other side of the crash barrier. For a human eye these vehicles are clearly irrelevant, however AVA does not see that.
(iv) AVAs voice is experienced as comprehensible, because of clear articulation and intonation.
(v) The amount of irrelevant messages causes the user to stop listening to AVA, which is naturally undesirable.
(vii) The user reports that the new voice of AVA is more pleasant to listen to.
Requirement (vi) is not addressed in this first iteration.

4.5 Iteration 2

In this iteration, human logic and the wishes of people will be implemented. The next subsections will provide the following solutions: (1) The mimicking of human logic, (2) mentioning objects that abruptly appear in vision of the cameras, (3)

a speech box depending on the vehicle speed, (4) a speech box specially for pedestrians and (5) an adaption to the stationary vehicles streak. The solutions will correspond to requirements (iii), (iii), (iii), (ii) and (i), respectively.

4.5.1 (iii) Human logic

As stated in requirement (iii), AVA is required to mimic human logic in its choices. To get a well-founded idea of logical human choices, a questionnaire will be introduced where participants are asked to determine the order in which they would like to have the traffic objects to be mentioned. The questionnaire exists of 6 pictures of traffic situations encountered in test run 2. The object indicated with the highest priority by the participant is compared to the object that AVA mentioned during the test run. The percentage of participants that choose the same object as most urgent as AVA is called the hitscore. It will then be aimed to make these the same object. See appendix B for the situations, exact results and the full questionnaire.

Question 1-3

Questions 1 through 3 had a hit score of 91%, 74%, 83% respectively. Accordingly, it can be concluded that for these situations AVA already resembles human logic.

Question 4

This situation has a hit rate of 9%. Cyclists on the oncoming intersection are indicated as most important, but because these are located behind other objects, they can not be detected by AVA. This makes them count as a false negative of the detection system.

Question 5

This situation has a hit rate of 4%. A car that suddenly passes on the right is considered most important. However, AVA does not mention this car.

In conclusion, in 3 out of 5 situations, AVA already equals with human expectation. Out of the 2 remaining situations, one is a false negative. The outcome of question 5 calls for a change regarding the logic of AVA. A solutions for the implementation of this logic is explained in the following subsection.

4.5.2 (iii) Rapidly appearing objects

The reason AVA did not mention the fast passing car in the situations of question 5 in section 4.5.1 is as follows. The Prius had a large speed at the time, so that means the red car was behind the line (solution for requirement (ii)) as seen in figure 10 and a message was not generated. This is solved by implementing a so-called First Detection label (appendix E line 599-601). This label is given to an object when it is first detected in the area between the $1 \cdot vehicleSpeed$ line and the car, giving it the exception of being mentioned behind this line.

4.5.3 (iii) High speed detection

To solve the problem of messages about vehicles on the opposite side of the road, a speed dependent speechbox is introduced (appendix E line 187-191). So far only a constant speechbox was implemented as discussed in section 4.3.2. It is determined that, when travelling at a speed of 65 km/h or more, information is only relevant one lane to the left and the right from the car, because these road users are part of the drivers direct traffic. The standard lane width in the Netherlands is 3,5 meters [2], so the speechbox has a horizontal distance of 5,25 meters on both sides of the car. The initial vertical distance is maintained. The speechbox will then resemble figure 11.

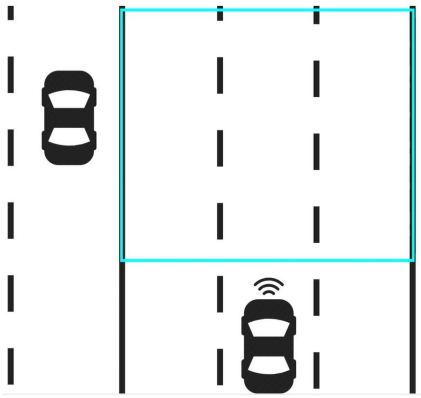


Figure 11: A visualisation of the speed dependent speechbox

4.5.4 (ii) Person speechbox

As mentioned in section 4.4, especially "person" detections are mentioned too late. This is caused by the relatively low speed of pedestrians in comparison to other road users such as cars, so a special person speechbox is implemented (appendix E line 177-185). This speechbox has the same format as the earlier mentioned general speechboxes, including an absolute x-distance of 15 meters on both sides of

the car, but its detection now ranges in y-distance from $2 \cdot vehicleSpeed$ to $6 \cdot vehicleSpeed$.

4.5.5 (i) Improvement on stationary vehicles

Some parked cars are still mentioned. The comparison between the stationary object and the streaklist can not properly function, because objects are randomly detected and not necessarily consecutively. This is why the solution for stationary vehicles as mentioned in section 4.3.1, is updated by comparing a stationary object with a streak of 30 other objects, instead of one other object.

4.6 Validation of iteration 2

This iteration is tested in a third test run. The route that is used can be found in appendix A. The same 10 minutes of representative footage are evaluated as in 4.4.
(i) A total of 18 out of 54 mentioned objects, concern parked cars, as can be seen in figure 14. This is a decrease compared to iteration 1. There are 6 false positives mentions. In situations where there is little traffic, the few present objects are mentioned even though they are irrelevant. Overall, AVA has the tendency to give few messages and these messages concern irrelevant objects.
(ii) Out of 10 messages concerning a person, 4 are delivered when that person has already passed. This is a degradation with respect to iteration 1.
(iii) AVA no longer talks about cars on the opposite side of the road at high speeds. AVA does mention suddenly appearing object, such as cars merging onto the highway.
(vi) In the next subsection, the results of the questionnaire will be provided.
Requirements (iv), (v) and (vii) are left unchanged with respect to iteration 1 and thus are unlisted. Furthermore, it can be concluded that the speed dependent speechbox, section 4.5.3, and first detection, section 4.5.2, are functioning successfully.

Questionnaire results
In this section the results from the questionnaire, as discussed in 2.4.3, are reviewed. Elaborate results can be found in appendix C.1.
Question 1-3 & 6
With a hitscore of 98%, 89%, 72% and 91% respectively, AVA performs sufficiently in these situations. Thereby, the logic of the speed dependent speechbox and first detection function are validated.
Question 4
This situation has a hitscore of 0 %. The car is mentioned before the person about to cross, although the person is rated higher than the car by participants. The car receives a high urgency due to its large speed. Likewise, the stationary person receives a low urgency. This means that, according to human logic, the speed has too large a factor in the current urgency function, see equation 1. Moreover, the person is not valued enough in urgency as an object, especially with respect to the car.
Question 5
This situation has a hitscore of 2 %. The parked car with the opening door, is marked as most important to mention. However, AVA does not see a parked car as relevant due to its programming, see section 4.3.1, and mentions the person on the sidewalk. The importance of the car is created by the opening of the door, something the object detection is not able to detect. Thus, in this case the deviation is caused by a form of human logic that can not yet be implemented in AVA.

The results from question 1 to 6 are visualised in figure 12. The figure visualises the similarity in order of urgency between AVA and the participants in every situation. Each colour represents an object that AVA mentioned and the magnitude is dependent on the percentage of participants that agrees with the objects place in the ranking. The total of mentions is visualised in the graph.

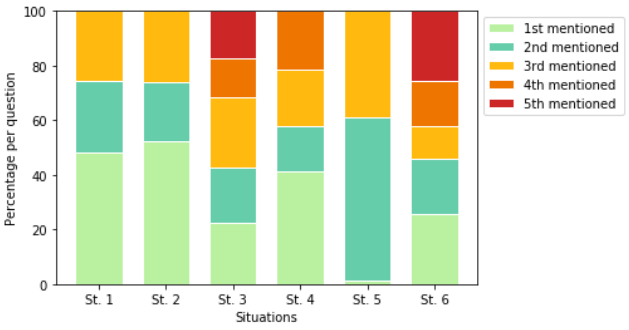


Figure 12: Percentage resemblance scored per question

5 Final design

5.1 Final iteration

After test run 3, a few changes need to be made to present AVA as required and desired. This section will provide the last set of implemented solutions.

To prevent the system from mentioning irrelevant objects when nothing else is happening in traffic, as cited in requirement (i), a general iteratively based threshold for the urgency is added to the code (appendix E line 174).

The values of Δx and Δy for the distance between two parked vehicles, are changed from 1 and 3.5 to a maximum value of 0.8 and 5 respectively. When a streak of cars is parked, these values are a better approximation of the real life distances. It is chosen to remove the filtering of oncoming traffic completely, due to the uncertainty in the relative velocity of objects as mentioned in 4.3.1 refraining it from functioning properly and its small contribution to the system.

When looking at the results of the human logic questionnaire, see appendix C.1, it can be noticed that people are not mentioned as much as participants would like them to. Three solutions are implemented to solve this problem: (1) The person speech box from 4.5.4 which was initially speed dependent, is now independent of the speed. (2) The TypeUrgency of a person is increased from 4 to 4.9 (appendix E line 77). This is kept lower than 5, the value of a bicycle, to partially solve the problem of mentioning a cyclist as a person, which can be confusing for the driver. (3) The a, b and c factors of function 1 are optimised, so that the products of the different urgencies are now in the same order of magnitude. The figure in 7 then changes to:

Multiplication factors	a	b	c
Original value	3	4	60
New value	8	2	60

Figure 13: Updated urgency function values

5.2 Validation of final design

The changes made between the second iteration and the final design are tested via a last reiteration as mentioned in section 2.3.3. The relevant requirements will be evaluated.

(i) A total of 5 out of 55 mentioned objects, concern parked cars. There are 3 false positives mentioned. This is an improvement with respect to the previous iterations as can be seen in figure 14. Currently, 71% of AVAs messages are relevant.

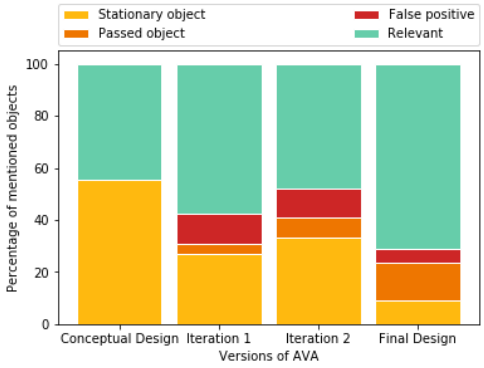


Figure 14: Classification of objects mentioned in part of test run

(ii) AVA gives 8 messages about road users that have already passed. In figure 15, a graph can be seen of the distance from the object at the time AVA mentions it and the vehicle speed at that time and thus the time the car would take to reach that object without interference. As explained in 2.4.2, this time needs to be larger than or equal to 1 for the mention to be timely. Currently, 29% of AVAs mentions can be considered timely.

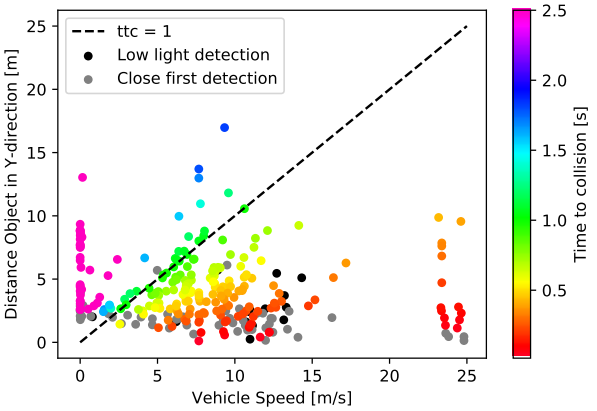


Figure 15: Scatterplot of the y-distance of an object versus the vehicle speed at the time the object is mentioned

(vi) This requirement is validated with a questionnaire discussed in 2.4.4. This section

will discuss the results. See appendix C.2 for the situation, exact results and the full questionnaire.

Question 1 & 6

From these results it can be concluded that when two objects are in the same direction and neither have a particularly high urgency, it is not clear which object is meant by AVA. It is assumed that people expect an object with a high urgency

Question 2

On a road with different lanes, participants view the car in the same lane as in front. This is not necessarily the case for AVA and could cause confusion.

Question 3-5

When one object is noticeably more urgent than the other objects in the situation, it is assumed that object is being mentioned disregarding the direction. In conclusion, it is not always clear which object is indicated by AVA.

Requirements (iii), (iv), (v) and (vii) have been left unaltered with respect to iteration 2, 4.6.

5.3 Additional option: eye-tracker

In this research, the possibility of implementing eye-tracking is explored as an additional option for a co-driver system. This possibility arises from requirement (i), and explores whether the system would provide more relevant messages, if the gaze of the driver is taken into account.

5.3.1 Design choices regarding gaze

The Smart Eye eye-tracker provides the angle of the gaze of the driver and the cameras determine the angle of an object relative to the car. The angles of these two vectors are compared to define the gaze angle. The stereo vision cameras in the Toyota Prius cover a total angle of approximately 60° and combined with the range of the eye-tracker the maximum gaze angle of a driver can reach the value of 110°. Furthermore, it is assumed that a person has a field of view of 60° and a central vision of 5° [7].

5.3.2 Implementing gaze

It is assumed that when an object has been in the drivers central vision, it is less urgent for AVA to mention this object. This is why the gaze angle is added to the urgency function as in 4.1.1. The urgency function then resembles equation 2. The factor d has a value of 0.5, in order for the product to be in the same order of magnitude as the products of the original function.

$$U = a \cdot Typeurgency + b \cdot VworldAbs \\ +c \cdot DistanceUrgency + d \cdot gazeAngle \quad (2)$$

When an object has been in the drivers central vision, it will be placed in UR2 (appendix E line 150-154) as seen in figure 6, following an identical path to objects that have been mentioned as explained in section 4.1.2 and 4.1.3. This is done to prioritise unseen objects above objects that have already been seen.

5.3.3 Validation of gaze implementation

This feature is not tested through a test run, but by means of a reiteration as mentioned in 2.3.3. The coded system is first replayed with the original function and later on the code with the eye-tracking function. Because the eye-tracking factor has now been added to the urgency function, it will be larger than the urgencies from the original function. Consequently, the threshold as mentioned in section 5.1 has been temporarily removed for fair comparison.

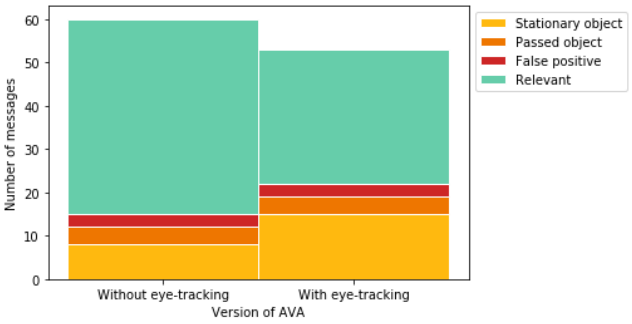


Figure 16: Classification of messages from AVA during test run

As can be seen in the graph above, AVA gives less messages when using the eye-tracker and the percentage of relevant messages is lower. An explanation for this is that driver tends to look at the most relevant objects, so these will not be mentioned. Implementing an eye-tracking option would have to be combined with a threshold, otherwise the information is irrelevant.

5.4 Demonstration

In this paper and the accompanying demonstration video, a prototype of the Automated Vehicle Assistant concept is presented with the intention of improving the situational awareness of the driver for a successful manual takeover in partially autonomous driving. The video can be accessed at *https://vimeo.com/380507599*.

6 Discussion

In this section, both potential shortcomings and possible improvements are discussed along with long term design ideas that can be implemented or investigated in future designs and research.

The current urgency function applies the absolute velocities of objects measured in the world frame. A more accurate speed-urgency could be generated using their relative velocity. This would favour objects approached by the car at high speed, over objects driving along at the same speed, which is currently not the case. Objects approaching the car perpendicularly could also be favoured over objects leaving the car in that direction. However, due to relative velocities being unreliable during the design of AVA, they are not implemented in the current urgency function.

The data processing is complicated by double and erroneous detections. This can be caused by overexposed low light situations or errors in the image tracking. The current system using two cameras with stereo vision is not capable of detecting objects behind the car, under bad light conditions (low or back light) or at night. Future designs can focus on implementing a LUX LIDAR [10] sensor on the roof of the car. LIDAR makes a 360° scan of the surroundings, and use this data combined with the camera data, as input for the system. Moreover, LIDAR contains a light source so that the system is able to detect objects at night or scenarios with low or back light conditions. Lastly, LIDAR would also be able to give the objects a correct relative velocity.

Although the urgency function functions properly when applied to standard situations, AVA falls short of human logic in anomalous situations such as someone suddenly opening the doors of a vehicle parked on the side of the road. To identify and anticipate on these unusual situations, an advanced image recognition system could be used.

According to the results of the recognition questionnaire in 5.2, it seems that the direction is disregarded more frequently when an object is more urgent than others. On the other hand, when two road users have a similar urgency, it is unclear which road user is mentioned by AVA. A solution can be to adapt the style of the messages to the difference in urgency, i.e. when the difference in urgency is large, a short message is enough. For situations with a small difference in urgency, a detailed message must be sent. According to Fleming [5] in such a situation details about the road user and landmarks must be added to a message to make it clear for a driver.

Results from questionnaire 2 show differences in interpreting the direction given by AVA, as presented in 8. For example, AVA classifies a direction in the left plane and gives it direction "left", but a driver can interpret it as "in front". A solution to solve this misunderstanding is to implement a gaze trainer in the code. After the gaze of the driver is calibrated in the system, the gaze trainer can let the driver experience the different detection zones. For example by turning the head slowly to the left until the gaze trainer will say, "at this angle, objects are mentioned as left."

The effectiveness of the eye-tracking is decreased by bad lighting conditions. In this study problems were encountered with low lighting that over exposed the driver's face, so that the eye tracking cameras could no longer distinguish the drivers eyes. Because of the way the code is structured, it will stop running when it does not get eye tracking input. This occurred during the third test run. To compare both human logic that follows the second questionnaire and the commands of AVA, the individual urgencies of objects in the questionnaire are manually calculated with function 1. These values could differ from the actual values.

6.1 Further recommendations

Feedback plays an important role in car-driver interactions it gives context for understanding, acceptance of a driver and keeps the driver more attentive [11]. By letting the electronic co-driver ask questions about the traffic, an the driver answering

them, a driver can be less distracted.

In this study, a logic speech rate was implemented in AVA. This speech rate was based on trial and error and by examining literature as provided in section 3.3. Future research can focus on providing the optimal speech rate for electronic vehicle assistants by doing experiments on participants with different speech rates and their experiences.

Using this speech-based prototype AVA, extensive research with participants could be done to determine the amount of acceptance the driver has towards the system. In appendix D, a detailed elaboration of such a research can be found.

7 Conclusion

In section 5.2, the final prototype for speech-based assistance in partially automated driving, AVA, is validated one last time. In the following picture the essence of what the system can do, is visualized.

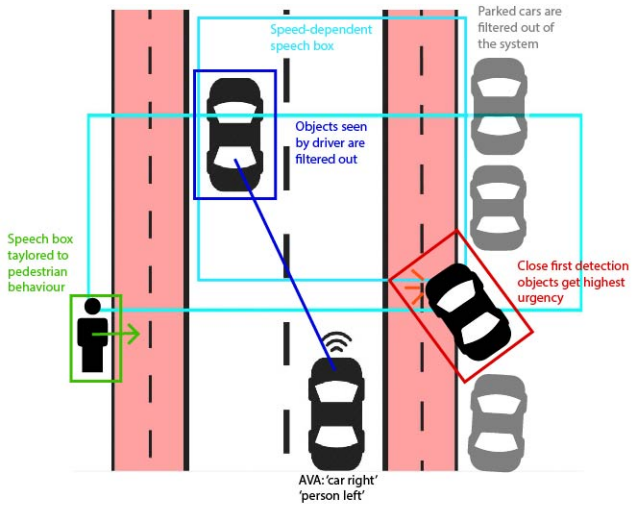


Figure 17: Simplified overview of capabilities of AVA

Most of the irrelevancies along the road are filtered out, even with the additional use of an eye-tracker. Objects that form an acute danger in traffic, are given priority. As it is proven that appropriate feedback reduces the time needed for a successful takeover, it can be concluded that AVA provides a clear, human like, solution to situational awareness improvement to provide a better possibility of manual interference. Until the future where driving is fully automated, it is important to keep our eyes on the road ahead.

7.1 Acknowledgements

Our gratitude goes out to the Intelligent Vehicle department of the Delft University of Technology, for thinking along and for the use of the Toyota Prius. In addition, we would like to thank Ir. J. Stapel and Dr. Ir. J. de Winter for their time, cooperation and involvement in this project.

References

[1] Bazilinskyy, P., Petermeijer, S.M., Petrovych, V., Dodou, D. & de Winter, J.C.F. (2018). Take-over requests in highly automated driving: A crowdsourcing survey on auditory, vibrotactile, and visual displays. *Transportation Research Part F: Traffic Psychology and Behaviour, 56*, 82-98.

[2] Dienst Beheer Infrastructuur (2012). *Handboek ontwerpcriteria wegen: Versie 4.0*. Provincie Zuid-Holland.

[3] Eriksson, A. & Stanton, N. (2017). The chatty co-driver: A linguistics approach applying lessons learnt from aviation incidents. *Safety Science, 99*(A), 94-101.

[4] eSpeak (2019). *eSpeak text to speech.* Consulted on 14 October 2019, from http:\\espeak.sourceforge.net.

[5] Fleming, J., Green, P. & Katz, S. (1998). *Driver performance and memory for traffic messages: Effects of the number of messages, audio quality, and relevance* (UMTRI Technical Report 98-22). Michigan: University of Michigan Transportation Research Institute

[6] Funke, G., Greenlee, E., Carter, M., Dukes, A., Brown, R. & Menke, L. (2016). Which eye tracker is right for your research? Performance evaluation of several cost variant eye trackers. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 60*(1), 1240-1244.

[7] The Government of the Hong Kong Special Administrative Region of the People's Republic of China: Environmental Protection Department (2017). *Proposed comprehensive residential and commercial development atop Siu Ho Wan Depot*(Environmental Impact Assessment Report 252/2017), Appendix 11.1: Prediction of visual impact based on field of view.

[8] Griffiths, R. (1992). Speech rate and listening comprehension: Further evidence of the relationship. *Teachers of English to Speakers of Other Languages Quarterly, 26*(2), 385-390

[9] Heckman, M., Steinhardt, N., Orth, D., Bolder, B., Dunn, M. & Kolossa,D. (2019, September). CORA, a prototype for a cooperative, speech-based on-demand intersection assistant. *Presented at the AutomotiveUI Conference, Utrecht, the Netherlands.*

[10] Himmelsbach, M., Müller, A., Lüttel, T. & Wünsche, H.J. (2008). LIDAR-based 3D object perception. *Presented at 1st International Conference on Application and Theory of Automation in Command and Control Systems, Barcelona, Spain.*

[11] Koo, J., Kwac, J., Ju, W., Steiner, M., Leifer, L. & Nass, C. (2015). Why did my car just do that? Explaining semi-autonomous driving actions to improve driver understanding, trust, and performance. *International Journal on Interactive Design and Manufacturing, 9*(4), 269-275.

[12] van der Laan, J.D., Heino, A. & de Waard, D. (1997). A simple procedure for the assessment of acceptance of advanced transport telematics. *Transportation Research Part C: Emerging Technologies, 5*(1), 1-10.

[13] McGehee, D.V., Mazzae, E.N., Scott Baldwin, G.H. (2000). Driver reaction time in crash avoidance research: Validation of a driving simulator study on a test track. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 44*(20), 320-323.

[14] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A. & Kavukcuoglu, K. (2016). WaveNet: A generative model for raw audio. *ArXiv.org:1609.03499v2.*

[15] Stapel, J.C.J. (2019). Collect vehicleData.py. *Available in the marked regions of Appendix E*, lines 434-810 as in appendix E.

[16] Vanholme, B., Gruyer, D., Lusettie, B., Glaser, S. & Mammar, S. (2013). Highly automated driving on highways based on legal safety. *IEEE Transactions on Intelligent Transportation Systems, 14*(1), 333-347.

[17] Walker, F., Wang, J., Marten, M.H. & Verwey, W.B. (2019). Gaze behaviour and electrodermal activity: Objective measures of drivers' trust in automated vehicles." *Transportation Research Part F: Traffic Psychology and Behaviour, 64*, 401-412.

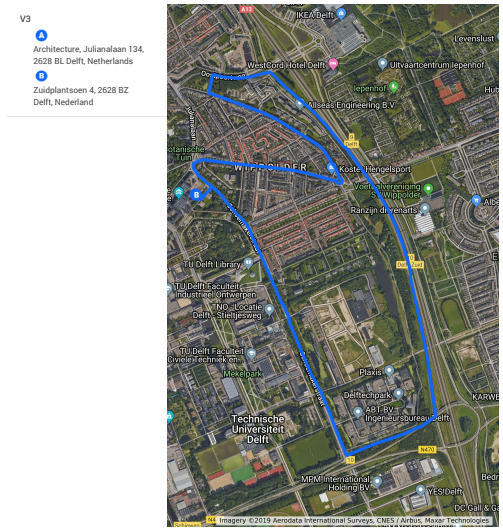
Appendices

A Routes

A.1 Test run 1

This route was driven during the first test run as mentioned in section 4.2.

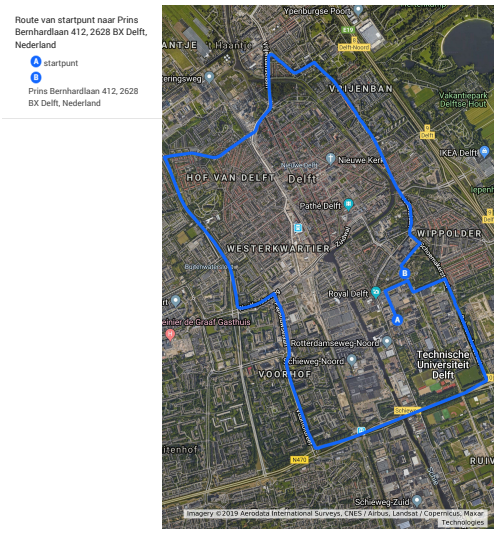
Route 1: November 12, 10 minutes



A.2 Test run 2

This route was driven during the second test run as mentioned in section 4.4.

Route 2: November 22, 23 minutes



A.3 Test run 3

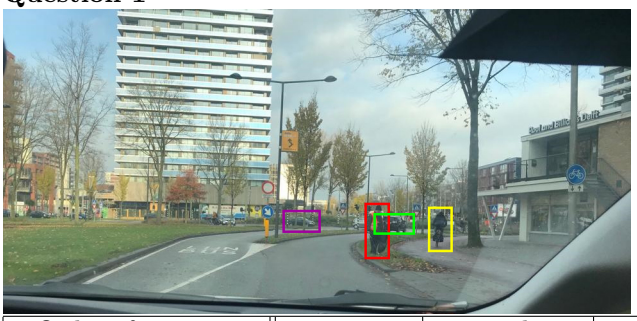
This route was driven during the third test run as mentioned in section 4.6.

Route 3: November 29, 26 minutes

B Questionnaire human logic

This appendix consists of the questions in the questionnaire discussed in section 2.4.3 and section 4.5.1. The following introduction was included:
Your co-driver is sitting next to you in the car and you have to listen to what they tell you about the surroundings. To make it as meaningful as possible, you want the co-driver to tell you the most important things first. Participants are asked the question: *"In which order would you like to hear something about the traffic in this situation?"*, for each of the next situations.

Question 1



Order of mention	1st	2nd	3rd	4th
Questionnaire result	Red(91%)	Green(57%)	Purple(43%)	Yellow(35%)
Current AVA	Green	Red	Yellow	Purple

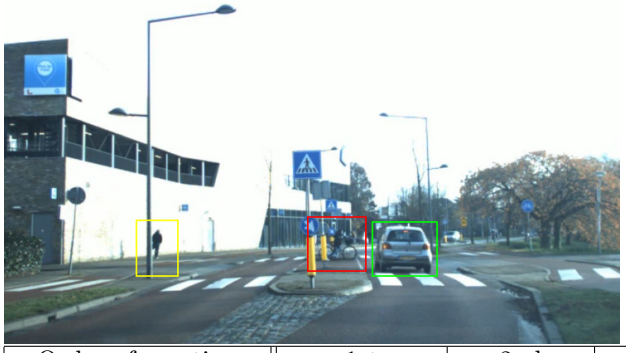
The purple and yellow traffic object were interchangeable, 43% would like to hear Yellow third and 30% would like to hear Purple last. The green object was mentioned by long before the situation occurred. AVA is accurate in this situation.

Question 2

Order of mention	1st	2nd	3rd
Questionnaire result	Red(74%)	Yellow(61%)	Green(61%)
Current AVA	Green	Red	Yellow

Green was mentioned 10 seconds earlier. AVA is accurate in this situation.

Question 3



Order of mention	1st	2nd	3rd
Questionnaire result	Green(83%)	Red(83%)	Yellow(96%)
Current AVA	Green	Yellow	-

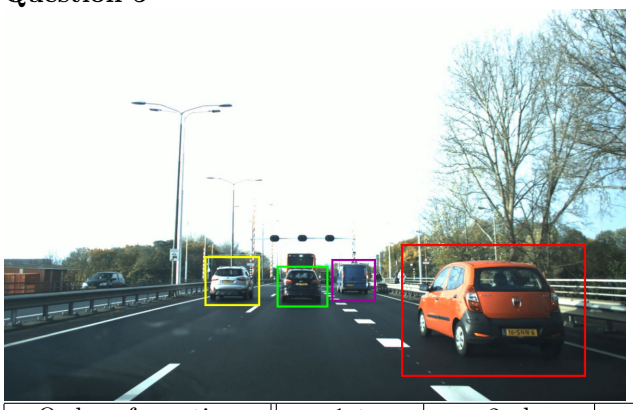
Red was not mentioned by AVA, because it could not be detected. AVA is accurate in this situation.

Question 4

Order of mention	1st	2nd	3rd	4th
Questionnaire result	Yellow(48%)	Purple(52%)	Red(61%)	Green(78%)
Current AVA	Red	-	-	-

Both the purple and yellow objects were seen as almost equally urgent by the participants. Of the participants, 39% would like to hear Purple first and 30% would like to hear Yellow second. AVA is not yet accurate in this situation.

Question 5



Order of mention	1st	2nd	3rd	4th
Questionnaire result	Red(95%)	Green(68%)	Purple(55%)	Yellow(82%)
Current AVA	Green	Yellow	-	-

This question included: "*You want to merge onto the rightmost lane*". One answer was marked as invalid. AVA is not yet accurate in this situation.

C Questionnaire validation requirement (iii) and (vi)

Requirement (iii) and desirable (iv) are validated in the same questionnaire. This questionnaire has the following introduction:
This survey is about an electronic co-driver, a system in your car that talks about the environment you are driving in. Imagine that you are driving a car and you have to listen to what the system tells you about the traffic around you. To make the information as meaningful as possible, you want the system to tell you the most important things first."
Information about the system:

- *The system detects 'person','car', 'cyclist' and 'motorcycle'(sometimes a cyclist is called a 'person').*
- *The system can tell about 'left', 'right' and 'in front'.*

Note: there is no right or wrong answer.

C.1 Relevance questionnaire

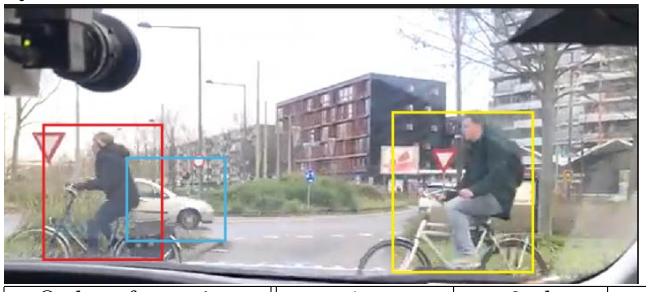
This appendix displays the results from the questionnaire as discussed in section 2.4.3 and 4.6. Participants are asked the following question: "*In what order of importance would you like to hear about the traffic, from your co-driver, in this situation?*".

Question 1

Order of mention	1st	2nd	3rd
Questionnaire result	Red(96%)	Blue(53%)	Yellow(55%)
Current AVA	Red	Yellow	Blue

For this question the additional information was added: "*You want to insert to the left; the white car does as well and the van inserts to the right*". In this result, the difference between the 2nd and 3rd mention was minimal for the questionnaire as well as AVA. Therefore AVA can be seen as accurate in this situation.

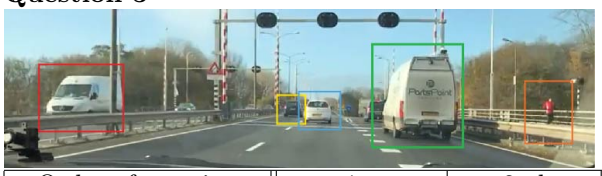
Question 2



Order of mention	1st	2nd	3rd
Questionnaire result	Yellow(89%)	Blue(53%)	Red(55%)
Current AVA	Yellow	Red	Blue

In this result, the difference between the 2nd and 3rd mention was minimal for the questionnaire as well as AVA. Therefore AVA can be seen as accurate in this situation.

Question 3



Order of mention	1st	2nd	3rd	4th	5th
Questionnaire result	Green(72%)	Blue(66%)	Yellow(83%)	Orange(47%)	Red(55%)
Current AVA	Green	Blue	Yellow	-	-

AVA is accurate in this situation.

Question 4

Order of mention	1st	2nd	3rd	4th
Questionnaire result	Red(98%)	Green(40%)	Yellow(49%)	Purple(51%)
Current AVA	Purple	Green	Red	Yellow

Other clarifying numbers are: 36% of the participants would like Green to be mentioned the last and 36% of the participants would like Yellow to be mentioned second. AVA is not yet accurate.

Question 5

Order of mention	1st	2nd	3rd
Questionnaire result	Blue(98%)	Yellow(62%)	Red(64%)
Current AVA	Yellow	Red & Blue	-

AVA is not yet accurate.

Question 6

Order of mention	1st	2nd	3rd	4th	5th
Questionnaire result	Red(91%)	Yellow(72%)	Green(43%)	Blue(57%)	Purple(91%)
Current AVA	Red	Blue	Yellow	-	-

For clarification, 34% of the participants would like Green to be mentioned as last. Blue and Purple are not mentioned by AVA. AVA is accurate.

C.2 Object recognition

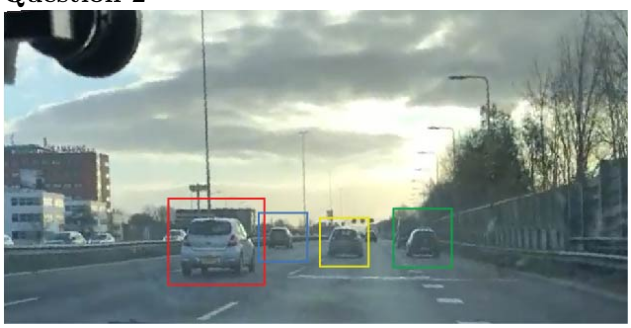
This section displays the results from the questionnaire mentioned in section 2.4.4. The participants were asked: "*Your co-driver warns you with "*[message]"*. What object do you think your co-driver has detected?*".

Question 1

The AVA message was: "car left". To the participants it was not clear which car was indicated.

Object	Yellow	Red
Indicated as mentioned object	55%	45%

Question 2



The AVA message was: "car in front". To the participants it was not clear which car was indicated.

Object	Red	Blue	Yellow	Green
Indicated as mentioned object	4%	0%	96%	0%

Question 3

The AVA message was: "person in front". To the participants it was not clear which person was indicated.

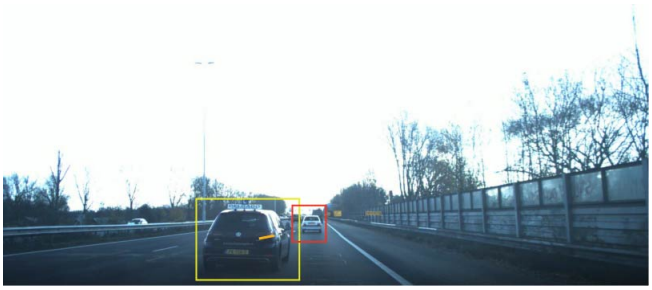
Object	Red	Yellow	Green
Indicated as mentioned object	81%	19%	0%

Question 4

The AVA message was: "person left". To the participants it was not clear which person was indicated.

Object	Red	Green	Yellow
Indicated as mentioned object	19%	45%	36%

Question 5



The AVA message was: "car in front". To the participants it was not clear which car was indicated.

Object	Yellow	Red
Indicated as mentioned object	96%	4%

Question 6

The AVA message was: "person in front". To the participants it was not clear which person was indicated.

Object	Red	Yellow
Indicated as mentioned object	45%	55%

D Experimenting recommendations

In this paper it has been researched what is believed to be a successful version of the chatty co-driver. However a technology can only be implemented if it is adopted by the user [17], in this case the driver. In this appendix a recommendation can be found for the testing of the acceptance and trust of the driver towards the chatty co-driver.

As mentioned in section 6, it is recommended to continue examination on the use of eye tracking data. A recommended format for this examination is to include the three following situations: (1) AVA mentions everything it would do as programmed until this point, (2) AVA gives information on the road users the driver has already seen and (3) AVA gives information on the road users that the driver has not yet seen. In this manner it can be investigated if and if so how the usage of eye tracking data would improve the acceptance of the system.The participants will be asked to drive a route in a car equipped with the chatty co-driver four consecutive times, each with the implementation of a different situation and naturally a baseline.

D.1 Participants

At least 20 participants are needed in order for the experiment to be considered significant. Concerning the given information beforehand, the participants should be told no more than that they are about to cooperate in an experiment concerning an electronic co-driver. This is to prevent the premature forming of opinion and inconsistency of provided information between participants.
Before commencing the experiment the participants will be briefed by the experimenter via a pre-established message read aloud, again to prevent inconsistency. This briefing contains an explanation of the duration and setup of the experiment, still only revealing that the electronic driver will give suggestions about the surrounding traffic. Additionally, it contains the two following regulations. During the experiment non-experimental related communication will be kept to a minimum as to not interfere with the messages from the chatty co-driver and if the participant finds themselves ill at ease, stressed by the system or physically uncomfortable, they should inform the experimenter immediately so the experiment can be (temporarily) terminated.

After the briefing, an informed consent must be presented to them, in which they are informed on the goal, duration, procedure and possible risks of the experiment and the privacy regulations. Possible risks include over-stimulation of the participant and the risks that apply for everyday traffic. This informed consent is to be signed by the participant.

D.2 Route

For consistency, the route should be identical during each iteration of the experiment. It is recommended to use a route within one will encounter multiple different environments and traffic situations and a complete overview can be made of the acceptance of a chatty co-driver in everyday traffic e.g. urban area, highway, main roads and residential area. Each situation will have a duration for 10 minutes, to supply sufficient data to analyse but still eliminating factors like acclimatisation to the system.

D.3 Setup of the experiment

All participants will complete an identical experimentation. It is suggested to structure this experiment as follows:
The participant is to drive four round identical consecutive rounds, beginning with the baseline measurement followed by the three different situation in a randomised order. This is to eliminate the influence a specific order of situation would have on the experiment clear and concise directions will be given by the experimenter.These directions will be identical during each experiment. Following each situation the

participant will be asked to fill out a questionnaire. This is the vanderLaan-questionnaire which has specially been developed for measuring acceptance of telematic systems [12]. The participant is granted a minimum of 5 minutes rest before commencing the next iteration, to eliminate fatigue. The baseline measurement will always be the primary iteration, to gather data without the usage of AVA and to ensure the same level of familiarity with the route between all participants. After that the three situations will be driven, alternating the order with each participant. This is to eliminate factors such as customisation to the system.

E Code

```
from copy import deepcopy
import rospy  # make this script part of ros
import sys
import tf2_ros  # ros reference frame transformations?
import tf2_geometry_msgs  # transformations for custom ros messages?
import numpy as np  # for basic math
import cv2  # OpenCV for image processing
import colorsys
import math
import message_filters  # to synchronise the incoming messages
import cPickle as pickle # to store data
import operator
import thread # to run the main code and speech code in different threads
import os # to save data in a chosen file path
from playsound import playsound # to play a saved sound
from datetime import datetime # current date and time

from collections import Counter
from cv_msgs.msg import GenericTracks, HypothesesStamped
from containers_chatty import trackParam_container, gazeParam_container

from ros_smarteye.msg import EyeTracker
from visualization_msgs.msg import Marker, MarkerArray
from geometry_msgs.msg import PointStamped, Vector3

from sensor_msgs.msg import Image
from std_msgs.msg import ColorRGBA, String
from FixedTransformListener import FixedTransformListener
from nav_msgs.msg import Odometry

class gaze_associator:
    """__init__ is executed once by ROS when the gaze_associator node is created
    """
    def __init__(self, outname='tracked_objs_gaze_3d_marker', output_path_topic='gaze_path_marker',
                 gui_comm_name='message_to_gui'):
        self.path_pub = rospy.Publisher(output_path_topic, MarkerArray, queue_size=1)
        self.pub_to_gui = rospy.Publisher(gui_comm_name, String,
                                          queue_size=1)  # sends message back to gui containing path
                                                         # to saved pickle
        self.all_tracks = {}  # we will accumulate data in a dictionary we can access with
                              # ["unique_track_id"]["age"]
        self.tfBuffer = tf2_ros.Buffer()
        self.listener = FixedTransformListener(self.tfBuffer)
        self.odom_frame = rospy.get_param('~odom_frame', 'odom')  # World frame in which tracks...
                                                                  # ...should be visualized
        self.imu_frame = rospy.get_param('~left_frame',
                                         'left')  # vehicle-fixed frame in  which gaze angles...
                                                                       # ...should be resolved
        self.current_time_stamp = rospy.Time.now()
        self.last_time_stamp = rospy.Time.now()  # to clean history when we go back in time
                                                 # (convenient when replaying rosbag)
        self.countbagCounter = 0  # used for track visualisation
        self.UrgencyRanker = {}  # initial Urgency dictionary, keys are trackid and values are urgency
        self.UrgencyRanker2 = {}  # secondary Urgency dictionary, item is put in here once it...
                                                       # ...is pointed out by the chatty co-driver
        self.UrgencyThreshold = {} # Urgency at time of previous mention is saved here for comparison
        self.vehicleSpeed = 0 # initialise vehicleSpeed
        self.commandlist = ''  # contains the command the codriver will say
        self.time = 0
        self.command = None
        self.commands = {} #commands and ranking are saved here
```

```
62          self.poplist = [] # deleted data is saved here
63          self.odom_cache = []  # buffer of odom messages
64          self.streaklist_X = [] #list of X-coordinates of parked cars
65          self.streaklist_Y = [] # list of Y-coordinates of parked cars
66          self.oncomingtraffic_X = [] # list of X-coordinates of oncoming traffic
67          self.oncomingtraffic_Y = [] # list of Y-coordinates of oncoming traffic
68          self.parkedcarlist = [] # Trackid's of parked cars are saved here
69          self.iteration_data = {} # commands said by the co-driver are saved here
70
71      def Urgency(self, trackid):
72          '''
73          Gives an urgency value to an item based on its speed, distance and type.
74          '''
75          # Determine TypeUrgency
76          trackId = self.all_tracks[trackid].classId
77          if trackId == 'person':
78              TypeUrgency = 4
79          elif trackId == 'car':
80              TypeUrgency = 2
81          elif trackId == 'bus':
82              TypeUrgency = 1
83          elif trackId == 'bicycle':
84              TypeUrgency = 5
85          elif trackId == 'motorbike':
86              TypeUrgency = 5
87          elif trackId == 'truck':
88              TypeUrgency = 1
89          elif trackId == 'horse':
90              TypeUrgency = 5
91          else:
92              TypeUrgency = 0
93
94          # Determine velocity of an object from imu data
95          VworldVec, Vimuvec = self.CalculateVelocityCorrectly(trackid)
96          VworldAbs = math.sqrt(VworldVec[0] * VworldVec[0] + VworldVec[1] * VworldVec[1])
97          VimuAbs = math.sqrt(Vimuvec[0] * Vimuvec[0] + Vimuvec[1] * Vimuvec[1])
98          self.all_tracks[trackid].Vabs = VworldAbs
99
100         # Determine location of an object for current timestamp and 10 timestamps
101         # (1 second since function is called at 10 Hz) ago in imu frame
102         track_content = self.all_tracks[trackid].track_content[-1]
103         distance_x_new_imu = track_content.objectLocation_imu.point.x
104         distance_y_new_imu = track_content.objectLocation_imu.point.y
105         trackage = self.all_tracks[trackid].age
106         CloseFirstDetection = self.all_tracks[trackid].CloseFirstDetection
107         gazeAngle = track_content.gazeAngle
108         centralBool = self.all_tracks[trackid].beenIn_central
109
110         # Determine relative distance of an object and create DistanceUrgency
111         Distance = math.sqrt(distance_x_new_imu ** 2 + distance_y_new_imu ** 2)
112         if Distance <= 0.1:  # to avoid ZeroDivisionError, with some margin
113             DistanceUrgency = 10
114         else:
115             DistanceUrgency = Distance ** -1  # the closer an object the higher the urgency
116
117         if distance_y_new_imu < 0:  # filters out objects behind the car by giving it low urgency
118             U = 0
119         elif VworldAbs < 3.0 and trackId != 'person' and VworldVec[-1] != 0.0: # filters out slow...
120                                                                                #... objects except for persons
121             U = 0
122             # append the stationary object to the parked car lists
123             self.streaklist_X.append(distance_x_new_imu)
124             self.streaklist_Y.append(distance_y_new_imu)
125             self.parkedcarlist.append(trackid)
126
```

```
127         try:
128             delta_yvec = []
129             delta_yvec2 = []
130             delta_xvec = []
131             # Calculate the differences in x- and y-coordinates for the 30 last appended objects
132             for i in range(min(30, len(self.streaklist_Y))):
133                 delta_yvec.append(distance_y_new_imu - self.streaklist_Y[-i])
134                 delta_xvec.append(distance_x_new_imu - self.streaklist_X[-i])
135             for value in delta_xvec:
136                 if abs(value) <= 0.8:
137                     k = delta_xvec.index(value)
138                     delta_yvec2.append(delta_yvec[k])
139             delta_y = min(delta_yvec2)
140             # If the object is close to the previous parked car it is identified as parked
141             if abs(delta_y) <= 5:
142                 U = 0
143                 if trackid not in parkedcarlist:
144                     self.parkedcarlist.append(trackid)
145                     self.streaklist_X.append(distance_x_new_imu)
146                     self.streaklist_Y.append(distance_y_new_imu)
147             else:
148                 U = self.Detectionbox(distance_x_new_imu, distance_y_new_imu, TypeUrgency, VworldAbs
149                     DistanceUrgency, trackage, CloseFirstDetection, trackId, gazeAngle)
150                 if centralBool == True:
151                     if trackid in self.UrgencyRanker.keys():
152                         temp = self.UrgencyRanker.pop(trackid)
153                         self.UrgencyRanker2.update({trackid : temp})
154                         self.UrgencyThreshold.update({trackid : temp})
155         except:
156             U = self.Detectionbox(distance_x_new_imu, distance_y_new_imu, TypeUrgency, VworldAbs,
157                 DistanceUrgency, trackage, CloseFirstDetection, trackId, gazeAngle)
158             if centralBool == True:
159                 if trackid in self.UrgencyRanker.keys():
160                     temp = self.UrgencyRanker.pop(trackid)
161                     self.UrgencyRanker2.update({trackid : temp})
162                     self.UrgencyThreshold.update({trackid : temp})
163 
164         return U
165 
166     def Detectionbox(self, distance_x_new_imu, distance_y_new_imu, TypeUrgency, VworldAbs,
167                      DistanceUrgency, trackage, CloseFirstDetection, trackId, gazeAngle):
168         '''
169         Calculates the correct urgency for an id based on whether or not it is in the detection box
170         '''
171         # Define factors for urgencyfunction
172         a = 3
173         b = 4
174         c = 60
175         d = 0.4
176 
177         if trackId == 'person': # Filter out persons outside the person speech box, unless the...
178                                                                 # ...first detection is < 1 meter.
179             if distance_y_new_imu <= 1 or distance_y_new_imu >= 15 or abs(distance_x_new_imu) >= 5:
180                 if CloseFirstDetection == True:
181                     U = a * TypeUrgency + b * VworldAbs + c * DistanceUrgency + d * gazeAngle
182                 else:
183                     U = 0
184             else:
185                 U = a * TypeUrgency + b * VworldAbs + c * DistanceUrgency + d * gazeAngle
186         else: # Apply the normal speech box to other objects
187             if self.vehicleSpeed >= 18: # Filter out objects more than one lane away
188                 if abs(distance_x_new_imu) >= 5.25 or distance_y_new_imu >= 40: # a lane is 3.5 m
189                     U = 0
190                 else:
191                     U = a * TypeUrgency + b * VworldAbs + c * DistanceUrgency + d * gazeAngle
```

```
            elif self.vehicleSpeed >= 12.5 and self.vehicleSpeed < 18: # Filter out objects not on...
                                                                       # ...or directly beside the road
                if abs(distance_x_new_imu) >= 8 or distance_y_new_imu >= 30:
                    U = 0
                else:
                    U = a * TypeUrgency + b * VworldAbs + c * DistanceUrgency + d * gazeAngle
            else:
                if abs(distance_x_new_imu) >= 10 or distance_y_new_imu >= 30:
                    U = 0
                else:
                    U = a * TypeUrgency + b * VworldAbs + c * DistanceUrgency + d * gazeAngle

        return U

    def CalculateVelocityCorrectly(self, trackId):
        '''
        Calculates the velocity of an object
        '''
        nmeas = len(self.all_tracks[trackId].track_content)
        Vodom= [0.0, 0.0]
        Vimu = [0.0, 0.0] # setting to 0 instead of None, since both are handled in the same way
        if nmeas > 1:
            # calculate the velocity in world frame (absolute velocity)
            pos_new_odom = self.all_tracks[trackId].track_content[-1].objectLocation_world
            pos_old_odom = self.all_tracks[trackId].track_content[0].objectLocation_world
            dt = (pos_new_odom.header.stamp - pos_old_odom.header.stamp).to_sec()
            dx = pos_new_odom.point.x - pos_old_odom.point.x
            dy = pos_new_odom.point.y - pos_old_odom.point.y
            if dt>0:
                Vodom = [dx/dt, dy/dt]

            # calculate the velocity in imu frame (relative velocity)
            pos_new_odom = self.all_tracks[trackId].track_content[-1].objectLocation_imu
            pos_old_odom = self.all_tracks[trackId].track_content[0].objectLocation_imu
            dt = (pos_new_odom.header.stamp - pos_old_odom.header.stamp).to_sec()
            dx = pos_new_odom.point.x - pos_old_odom.point.x
            dy = pos_new_odom.point.y - pos_old_odom.point.y
            if dt>0:
                Vimu = [dx/dt, dy/dt]

        return Vodom, Vimu

    def directionEngine(self, trackid):
        '''
        Determines the direction of an object relative to the driver
        '''
        # Pick the latest x- and y-coordinates of an object
        try:
            x = self.all_tracks[trackid].track_content[-1].objectLocation_imu.point.x
            y = self.all_tracks[trackid].track_content[-1].objectLocation_imu.point.y
            #Calculates the angle from the object towards the driver
            try:
                angle = np.arctan(x / y) * 180 / math.pi

                if angle >= 20:
                    message = '_right'
                elif angle < 20 and angle >= -20:
                    message = '_in_front'
                elif angle < -20:
                    message = '_left'
                else:
                    message = ''

                return message
```

```
257             except:
258                 return ''
259         except:
260             return ''
261 
262     def GetItemWithMaxUrgency(self, dictionary):
263         '''
264         Takes in a dictionary and returns the key with the maximum value attached,
265         in our case the most urgent object from an UrgencyRanker
266         '''
267         try:
268             Item_Umax = max(dictionary.iteritems(), key=operator.itemgetter(1))[0]
269             return Item_Umax
270         except: # in this case the dictionary is empty
271             return None
272 
273     def PriorityDecider(self, UrgencyRanker, UrgencyRanker2, UrgencyThreshold):
274         '''
275         Decides which item to send to the speech engine
276         '''
277         # Picks the highest urgency from the Urgency Ranker and Urgency Ranker 2
278         ItemMax1 = self.GetItemWithMaxUrgency(UrgencyRanker)
279         ItemMax2 = self.GetItemWithMaxUrgency(UrgencyRanker2)
280         Factor = 3
281         if ItemMax1 == None:
282             if ItemMax2 != None:
283                 # Only if the urgency in Ranker 2 is 3 times the initial urgency it will be
284                 # sent to the speech engine
285                 if UrgencyRanker2[ItemMax2] >= Factor * UrgencyThreshold[ItemMax2]:
286                     return [ItemMax2, 2]
287                 else:
288                     return None
289             else:
290                 return None
291         elif ItemMax2 == None:
292             return [ItemMax1, 1]
293         elif UrgencyRanker2[ItemMax2] >= Factor * UrgencyThreshold[ItemMax2]:
294             # Only if the urgency in ranker 2 is bigger than in ranker 1 it will be mentioned
295             if UrgencyRanker2[ItemMax2] >= UrgencyRanker[ItemMax1]:
296                 return [ItemMax2, 2]
297             else:
298                 return [ItemMax1, 1]
299         else:
300             return [ItemMax1, 1]
301 
302     def UrgencyRankerFiller(self, track_data):
303         '''
304         The function fills the different urgency rankers at 10Hz.
305         '''
306 
307         # Determine speed of the Prius
308         odom = self.odom_cache.getElemBeforeTime(self.current_time_stamp)
309         if odom is not None:
310             self.vehicleSpeed = self.odom2speed(odom)
311 
312         # Check if the object is detected, calculate the urgency and put it in the correct ranker.
313         for track in track_data.tracks:
314             key = track.id
315             if key in self.all_tracks.keys():
316                 urg = self.Urgency(key)
317                 if key in self.UrgencyRanker2.keys():
318                     self.UrgencyRanker2[key] = urg
319                 else:
320                     self.UrgencyRanker[key] = urg
321 
```

```
322         # Delete old and passed vehicle updates from the urgency ranker
323         for key in self.UrgencyRanker.keys():
324             loc = self.all_tracks[key].track_content[-1].objectLocation_imu
325             y = loc.point.y
326             delay = (self.current_time_stamp - loc.header.stamp).to_sec() # Determines the...
327                                                                           # ...time since the latest update
328             # if an object is 3m behind the car or is not updates for 1s it is deleted
329             if y < -3 or delay>1.0:
330                 self.UrgencyRanker.pop(key)
331             # if the urgency is zero the object will be deleted
332             elif self.UrgencyRanker[key] == 0:
333                 self.UrgencyRanker.pop(key)
334 
335         # idem for UrgencyRanker2:
336         for key in self.UrgencyRanker2.keys():
337             loc = self.all_tracks[key].track_content[-1].objectLocation_imu
338             y = loc.point.y
339             delay = (self.current_time_stamp - loc.header.stamp).to_sec()
340             if y < -3 or delay>1.0:
341                 self.UrgencyRanker2.pop(key)
342                 self.UrgencyThreshold.pop(key)
343             elif self.UrgencyRanker2[key] == 0:
344                 self.UrgencyRanker2.pop(key)
345                 self.UrgencyThreshold.pop(key)
346 
347     def odom2speed(self, odom):
348         '''
349         Converts coordinates in the odom frame to speed
350         '''
351         V = odom.twist.twist.linear
352         return math.sqrt(V.x * V.x + V.y * V.y + V.z * V.z)
353 
354     def CommandMaker(self):
355         '''
356         Decides which trackid/command to sent to the speech engine from the Urgency Rankers
357         by updating the commandlist
358         '''
359         self.command = self.PriorityDecider(self.UrgencyRanker, self.UrgencyRanker2,
360                                             self.UrgencyThreshold)
361         if self.command != None:
362             ID = self.command[0]
363             self.commandlist = ID  # adds command to commandlist
364         else:
365             self.commandlist = ''
366 
367     def Type(self, trackid):
368         '''
369         Translates the class Id's into the word that will be pronounced.
370         '''
371         if trackid != '':
372             trackId = self.all_tracks[trackid].classId
373             if trackId == 'person':
374                 Type = trackId
375             elif trackId == 'car':
376                 Type = trackId
377             elif trackId == 'bus':
378                 Type = trackId
379             elif trackId == 'bicycle':
380                 Type = 'cyclist'
381             elif trackId == 'motorbike':
382                 Type = 'motorcycle'
383             elif trackId == 'truck':
384                 Type = trackId
385             elif trackId == 'horse':
386                 Type = trackId
```

```
387             else:
388                 Type = ''
389         else:
390             Type = ''
391 
392         return Type
393 
394     def Say1(self, Type, Direction, trackid):
395         '''
396         Makes a command that can be pronounced by AVA
397         '''
398         if self.command != None:
399             ID = self.command[0]
400             N = self.command[1]
401             if N == 1: # if an item from UrgencyRanker is chosen
402                 temporaryvalue = self.UrgencyRanker.pop(ID)  # takes the item out of UrgencyRanker
403                 self.UrgencyRanker2.update({ID: temporaryvalue})  # places item in UrgencyRanker 2
404                 self.UrgencyThreshold.update({ID: temporaryvalue})  # makes threshold for item...
405                                                                     # ...to compare current urgency with
406             else: # if an item from UrgencyRanker2 is chosen
407                 self.UrgencyThreshold.update({ID: self.UrgencyRanker2[ID]})  # updates threshold...
408                                                                 # ...because item is sent to speech engine
409 
410         # A list with data about every command is generated so a testrun can be analysed afterwards.
411         y = self.all_tracks[trackid].track_content[-1].objectLocation_imu.point.y
412         ttc_command = y/self.vehicleSpeed # time to collision is calculated from y location only
413         iterlist = [trackid, Type, Direction,
414                     self.all_tracks[trackid].track_content[-1].objectLocation_imu.point.x,
415                     self.all_tracks[trackid].track_content[-1].objectLocation_imu.point.y,
416                     self.vehicleSpeed, ttc_command, self.all_tracks[trackid].age,
417                     self.all_tracks[trackid].CloseFirstDetection]
418 
419         # The data is saved in a library with the time at which the command is spoken
420         self.iteration_data.update({rospy.Time.now():iterlist})
421         if Type != '' and Direction != '':
422             filename = Type + Direction
423             # The command that plays the sound
424             playsound('/home/humanfactors/ros/catkin_ws/src/chatty_codriver/audio/%s.mp3' % filename)
425 
426     def MakeSpeech(self, trackid):
427         '''
428         Takes a command from the commandlist and tells the speech engine to say it
429         '''
430         Type = self.Type(trackid)
431         Direction = self.directionEngine(trackid)
432         # A new thread is made so that the main code keeps running when something is said.
433         thread.start_new_thread(self.Say1, (Type, Direction, trackid))
434 '''
435 The source code below is from Collect_vehicleData.py, Stapel, J.C.J. 2019.
436 '''
437     def get_pick_color(self, id, gazeAngle):
438         """
439         for track visualisation: determines track colour based on track ID (hue)
440         and gaze angle (intensity)
441         """
442         g = gazeAngle
443         maxangle = 90.0
444         if (gazeAngle < 2):
445             g = 0
446         elif (gazeAngle < 7):
447             g = 30
448         elif (gazeAngle < 30):
449             g = 60
450         elif (gazeAngle >= 30):
451             g = 50 + g / 3
```

```
452         if math.isnan(gazeAngle) or gazeAngle > maxangle:
453             g = maxangle
454         colorsys.hsv_to_rgb
455         h = (int(id * 11 * 32) % 361) / 360.0
456         s = (100 - g / maxangle * 100) / 100.0
457         v = (100 - g / maxangle * 50) / 100.0
458         return colorsys.hsv_to_rgb(h, s, v)
459 
460     def fill_gaze_container(self, trackContainer, gaze, Point_world, Point_imu, track):
461         gaze_container = gazeParam_container()
462         gaze_container.FrameNumber = gaze.FrameNumber
463         gaze_container.gazequality = gaze.FilteredGazeDirectionQ
464         gaze_container.objectLocation_world = Point_world
465         gaze_container.objectLocation_imu = Point_imu
466         gaze_container.gazeAngle = self.get_smallest_angle(gaze.GazeOrigin, gaze.FilteredGazeDirection,
467                                                            gaze_container.objectLocation_imu.point)
468         gaze_container.Age = track.age
469         fixation = gaze.Fixation
470         saccade = gaze.Saccade
471 
472         # Determine if objects have been in the Field of View
473         if trackContainer.beenIn_FOV:
474             pass
475         else:
476             if gaze_container.gazeAngle < 30:  # in FOV
477                 trackContainer.beenIn_FOV = True
478             if gaze_container.gazeAngle > 30:  # outside FOV
479                 trackContainer.beenIn_FOV = False
480 
481         # Determine if objects have been seen with central vision
482         if trackContainer.seen:
483             pass
484         else:
485             if gaze_container.gazeAngle < 7:
486                 trackContainer.beenIn_central = True  # object been in central vision
487                 if fixation > 0:
488                     trackContainer.seen = True  # object seen
489                     trackContainer.beenIn_peripheral = False
490                 if saccade > 0:
491                     trackContainer.seen = False
492             else:
493                 trackContainer.beenIn_central = False
494                 trackContainer.seen = False
495 
496         # Define possible peripherally detected objects
497         # objects that never entered the FOV and could have been detected peripherally
498         if not trackContainer.seen:
499             if trackContainer.beenIn_FOV:
500                 trackContainer.beenIn_peripheral = True
501 
502         # Define objects that never entered FOV
503         if not trackContainer.beenOut_FOV:
504             pass
505         else:
506             if gaze_container.gazeAngle > 30:
507                 trackContainer.beenOut_FOV = True
508             elif gaze_container.gazeAngle < 30:
509                 trackContainer.beenOut_FOV = False
510 
511         if gaze_container.gazeAngle < 2:
512             trackContainer.centralTime += 1
513         if gaze_container.gazeAngle < 10:  # useful FOV
514             trackContainer.nearTime += 1
515 
516         return gaze_container
```

```
517     """ Perform transformations
518     """
519
520     def perform_transforms(self, track_data, track):
521         p_in_state_frame = PointStamped()
522         p_in_state_frame.header = deepcopy(track_data.header)
523         p_in_state_frame.point.x = track.state[0]
524         p_in_state_frame.point.y = track.state[1]
525         p_in_state_frame.point.z = 0.0
526
527
528         # initialize object location expressed in imu and map frame
529         p_in_imu_frame = PointStamped()
530         p_in_odom_frame = PointStamped()
531         p_in_imu_frame.header = p_in_state_frame.header
532         p_in_odom_frame.header = p_in_state_frame.header
533
534         # transform position to imu and map(=world) frame,
535         try:
536             tf2imu = self.tfBuffer.lookup_transform(self.imu_frame, track_data.header.frame_id,
537                                                     track_data.header.stamp)
538             p_in_imu_frame = tf2_geometry_msgs.do_transform_point(p_in_state_frame, tf2imu)
539             if self.odom_frame == track_data.header.frame_id:  # if track is already in odom...
540                                                     # ...frame, we don't have to transform it
541                 p_in_odom_frame = p_in_state_frame
542             else:
543                 tf2odom = self.tfBuffer.lookup_transform(self.odom_frame, track_data.header.frame_id,
544                                                          track_data.header.stamp)
545                 p_in_odom_frame = tf2_geometry_msgs.do_transform_point(p_in_state_frame, tf2odom)
546         except (tf2_ros.LookupException, tf2_ros.ConnectivityException, tf2_ros.ExtrapolationException):
547             rospy.loginfo('TF lookup error')
548
549         return p_in_odom_frame, p_in_imu_frame
550
551
552     """ Performs some fancy bookkeeping to populate the trackParam_containers
553     """
554
555     def append_tracks(self, track_data, gazes, ssd):
556         # prepare ssd dictionary for easy lookup
557         ssd_detections = {}
558         for hypothesis in ssd.hypotheses:
559             ssd_detections[hypothesis.id] = hypothesis
560
561         for track in track_data.tracks:
562             if track.id not in self.all_tracks.keys():
563                 self.all_tracks[track.id] = trackParam_container()
564
565             """ update track summary statistics """
566             self.all_tracks[track.id].age = track.age
567
568             """ perform transformations """
569             p_in_odom_frame, p_in_imu_frame = self.perform_transforms(track_data, track)
570
571             """ label trackIDs """
572             if (track.assoc_meas):
573                 measId = track.assoc_meas[0]
574                 if (measId in ssd_detections):
575                     self.all_tracks[track.id].classId = ssd_detections[measId].class_id
576
577             """ add all new gaze data to the track. since we have multiple gazes, we will interpolate
578             object locations over time, if we have a previous detection stored. if not, we will simply
579             use the latest gaze.
580             """
581
```

```
582             if not self.all_tracks[track.id].track_content:  # if track_content is empty, there is...
583                                                                 # ...no point for interpolation
584                 gaze = gazes[-1]
585                 self.all_tracks[track.id].track_content.append(self.fill_gaze_container(
586                                 self.all_tracks[track.id], gaze, p_in_odom_frame, p_in_imu_frame, track))
587             else:
588                 i = 0
589                 for gaze in gazes:
590                     prevPoint_world = self.all_tracks[track.id].track_content[-1].objectLocation_world
591                     prevPoint_imu = self.all_tracks[track.id].track_content[-1].objectLocation_imu
592                     Point_world = self.interpolate_point(prevPoint_world, p_in_odom_frame, gaze.header.stamp)
593                     Point_imu = self.interpolate_point(prevPoint_imu, p_in_imu_frame, gaze.header.stamp)
594                     self.all_tracks[track.id].track_content.append(
595                     self.fill_gaze_container(self.all_tracks[track.id], gaze, Point_world, Point_imu, track))
596                     i += 1
597
598             # Determines if the object is close first detected
599             if self.all_tracks[track.id].age <= 2 and
600                self.all_tracks[track.id].track_content[-1].objectLocation_imu.point.y <= 1:
601                 self.all_tracks[track.id].CloseFirstDetection = True
602
603     """ estimates position at given time with linear interpolation used to up-sample SSD detections
604     for each gaze message
605     """
606
607     def interpolate_point(self, startPointStamped, endPointStamped, atTime):
608         startTime = startPointStamped.header.stamp
609         endTime = endPointStamped.header.stamp
610
611
612         # clip atTime to strict interpolation between the points' time range
613
614         if atTime >= endTime:
615             return endPointStamped
616         elif atTime <= startTime:
617             return startPointStamped
618         # calculate new point
619         newPoint = PointStamped()
620         newPoint.header = deepcopy(startPointStamped.header)
621         newPoint.header.stamp = deepcopy(atTime)
622         a = (atTime - startTime) / (endTime - startTime)
623         newPoint.point.x = (endPointStamped.point.x - startPointStamped.point.x) * a
624                            + startPointStamped.point.x
625         newPoint.point.y = (endPointStamped.point.y - startPointStamped.point.y) * a
626                            + startPointStamped.point.y
627         newPoint.point.z = (endPointStamped.point.z - startPointStamped.point.z) * a
628                            + startPointStamped.point.z
629         return newPoint
630
631     """ calculates angle (in rad or deg) between gaze ray and vector between gaze origin and object.
632         Assumes that all data is provided in the same coordinate frame!
633     """
634
635     def get_smallest_angle(self, gaze_origin, gaze_vector, object_origin,in_deg=True):
636         vec1 = [object_origin.x - gaze_origin.x, object_origin.y - gaze_origin.y,
637                 object_origin.z - gaze_origin.z]
638         vec2 = [gaze_vector.x, gaze_vector.y, gaze_vector.z]
639         # normalise both vectors
640         vec1 = vec1 / np.linalg.norm(vec1)
641         vec2 = vec2 / np.linalg.norm(vec2)
642
643         angle = np.arccos(np.clip(np.dot(vec1, vec2), -1.0, 1.0))
644
645         if math.isnan(angle):
646             angle = math.pi
```

```
647             if in_deg:
648                 angle = np.degrees(angle)
649             return angle
650 
651         """
652             different version of get_smallest_angle. performs same calculation, but between any two angles
653             (not used at the moment)
654         """
655 
656 
657         def get_smallest_angle2(self, vec1, vec2, in_deg=True):
658             vec1 = [vec1.x, vec1.y, vec1.z]
659             vec2 = [vec2.x, vec2.y, vec2.z]
660 
661             # normalise both vectors
662             vec1 = vec1 / np.linalg.norm(vec1)
663             vec2 = vec2 / np.linalg.norm(vec2)
664 
665             # angle in steradian
666             angle = np.arccos(np.clip(np.dot(vec1, vec2), -1.0, 1.0))
667 
668             if math.isnan(angle):
669                 angle = math.pi
670 
671             if in_deg:
672                 angle = np.degrees(angle)
673             return angle
674 
675         """ Returns all gazes since previous execution of synchronizedCallback
676         """
677 
678         def getNewGazesFromCache(self, gaze_cache):
679             try:
680                 gazes = gaze_cache.getInterval(self.last_time_stamp, self.current_time_stamp)
681             except:
682                 rospy.loginfo("delay is %s ", (self.current_time_stamp - self.last_time_stamp).to_sec())
683                 return []
684 
685         """ create line strip marker for each unique track id. This should happen in map frame
686         """
687 
688         def create_line_marker(self, header):
689             msg_id = 0
690             # loop over all unique track id's
691             for track_key, track_value in self.all_tracks.iteritems():
692                 # create a new marker consisting of multiple points/line strips
693                 marker = Marker()
694                 marker.header.stamp = header.stamp
695                 marker.header.seq = header.seq
696                 marker.header.frame_id = self.odom_frame
697                 marker.action = Marker.ADD
698                 marker.lifetime = rospy.Duration(secs=100)
699                 marker.type = Marker.LINE_STRIP
700                 marker.id = msg_id
701                 msg_id = msg_id + 1
702 
703                 # set scale to a visually appealing value
704                 marker.scale.x = 0.3
705                 marker.scale.y = 0.3
706                 marker.scale.z = 0.3
707 
708                 # set orientation to identity
709                 marker.pose.orientation.w = 1
710 
711                 # get all positions from this track to create the line strip
```

```
712             for content in self.all_tracks[track_key].track_content:
713                 r, g, b = self.get_pick_color(track_key, content.gazeAngle)
714                 marker.points.append(content.objectLocation_world.point)
715                 marker.colors.append(ColorRGBA(r, g, b, 1.0))
716             line_markers.markers.append(marker)
717 
718         # Return the entire collection of tracks accumulated up until this point in time
719         return line_markers
720 
721     """this function triggers:"""
722     def synchronizedCallback(self, SSD3D_tracks, gaze, ssd_2d, gaze_cache,
723                              odom_cache):
724         self.current_time_stamp = SSD3D_tracks.header.stamp
725         self.odom_cache = odom_cache
726 
727         self.time += 1
728         if len(gazes) > 0:
729             self.append_tracks(SSD3D_tracks, gazes, ssd_2d)
730             self.UrgencyRankerFiller(SSD3D_tracks)
731             if self.time % 25 == 0:
732                 self.CommandMaker()
733                 if self.commandlist != "":
734                     try:
735                         u = self.UrgencyRanker[self.commandlist]
736                     except:
737                         u = self.UrgencyRanker2[self.commandlist]
738                     if self.vehicleSpeed >= 18:
739                         self.MakeSpeech(self.commandlist)
740                     else:
741                         if u > 25:
742                             self.MakeSpeech(self.commandlist)
743 
744             # create visualization marker of line segments of the separate tracks
745             self.countbagCounter = self.countbagCounter + 1
746             if (self.countbagCounter > 60):
747                 count = 0
748                 print('length Ranker1: {0}, Ranker2:{1} (threshold: {2}), tracks: {3} (containers: {4})'.
749                       format( len(self.UrgencyRanker), len(self.UrgencyRanker2),
750                       len(self.UrgencyThreshold), len(self.all_tracks),count))
751                 self.countbagCounter = 0
752         else:
753             rospy.loginfo("waiting for gazes...")
754         self.last_time_stamp = self.current_time_stamp
755 
756 def main(args):
757     rospy.init_node('gaze_SA_parameterization', anonymous=False)
758     gazeAssociator = gaze_associator()
759 
760     SSD1_path = rospy.get_param('~input_ssd2',
761                                 '/ueye/left/ssd_detected_objs_out')
762     tracker_path = rospy.get_param('~input_tracker', '/ueye/left/ped_track_objs')
763 
764     sd1 = message_filters.Subscriber(SSD1_path, HypothesesStamped)
765     tracker = message_filters.Subscriber(tracker_path, GenericTracks)
766     gaze_sub = message_filters.Subscriber("smarteye", EyeTracker)
767 
768     odom_path = rospy.get_param('~odom_msg',
769                                 '/ukf_localization_odom/odometry/filtered')
770     odom_sub = message_filters.Subscriber(odom_path, Odometry)
771     odom_cache = message_filters.Cache(odom_sub, cache_size=100)
772 
773     gaze_cache = message_filters.Cache(gaze_sub,
774                                        cache_size=200)
775     ts = message_filters.ApproximateTimeSynchronizer([tracker, gaze_sub, sd1], 120,
776                                                      0.0083)
```

```
777     ts.registerCallback(gazeAssociator.synchronizedCallback, gaze_cache, odom_cache)
778 
779     timestamp = gazeAssociator.current_time_stamp
780     time = datetime.now().strftime("%m_%d_%Y_%H_%M_%S")
781     # output the necessary data
782     path = '/home/humanfactors/ros/catkin_ws/src/chatty_codriver/pickles'
783     filename1 ='iteration_data_test' + time
784     filename3 ='parkedcarlist_test' + time
785     filepath1 = os.path.join(path, filename1)
786     filepath3 = os.path.join(path, filename3)
787     # spin() simply keeps python from exiting until this node is stopped
788     try:
789         rospy.spin()
790     except KeyboardInterrupt:
791         print("saving data...")
792 
793     cv2.destroyAllWindows()
794 
795     # Save the data
796     outfile1 = open(filepath1, 'wb')
797     outfile3 = open(filepath3, 'wb')
798     pickle.dump(gazeAssociator.iteration_data, outfile1)
799     outfile1.close()
800     pickle.dump(gazeAssociator.parkedcarlist, outfile3)
801     outfile3.close()
802     print("data saved!")
803 
804 if __name__ == '__main__':
805     # Starts an introduction
806     playsound('/home/humanfactors/ros/catkin_ws/src/chatty_codriver/audio/intro.mp3')
807     main(sys.argv)
808 '''
809 End of the cited source code Collect_vehicleData.py by Stapel, J.C.J, 2019.
810 '''
```