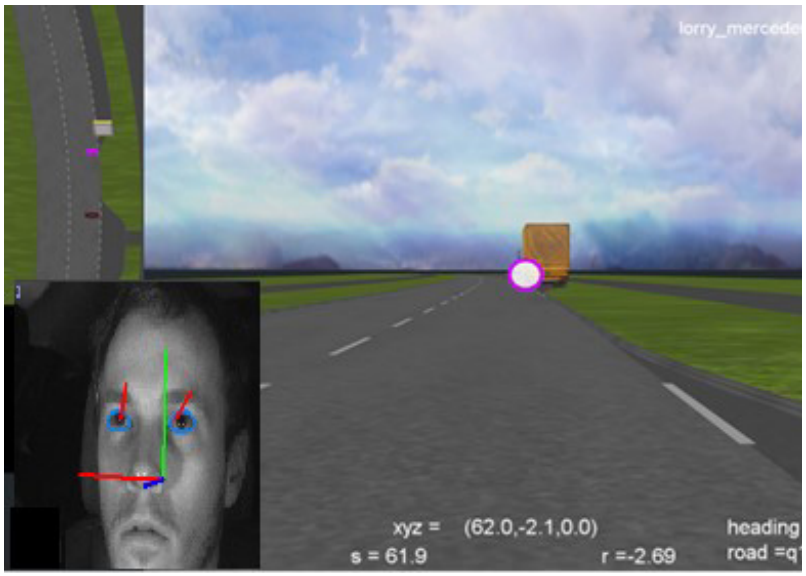
Linking gaze tracking with a simulated world



Authors

Andreas Jansson, VTI
Christer Ahlström, VTI

Driving Simulation Centre

www.vipsimulation.se

ViP PM 2017-1

Linking gaze tracking with a simulated world

Authors

Andreas Jansson, VTI

Christer Ahlström, VTI

www.vipsimulation.se

Cover picture: VTI. Screen shot of simulation graphics, and SmartEye camera picture of driver.
Reg. No., VTI: 2012/0392-26
Printed in Sweden by VTI, Linköping 2017

Preface

The *VIMSI* project was a collaboration project between the Swedish National Road and Transport Research Institute (VTI), SmartEye, the Swedish Road Marking Association and the Swedish Transport Administration within the competence centre *ViP Driving Simulation Centre* (www.vipsimulation.se). The project was financed by the ViP centre, i.e. by the ViP partners and the Swedish Governmental Agency for Innovation Systems (VINNOVA).

Initially the project had two goals. The first goal was to develop a software capable of effectively combining the eye tracking information from the SmartEye system with VTI's driving simulator data in order to achieve an analysis tool for studying drivers' eye movements while driving a simulator. Another requirement was to make it possible for the test leader to watch what objects the driver was looking at in real time. The second goal was to use the developed software to evaluate road markings. Unfortunately, due to delays and the first part being more challenging than expected we never got to the second part. Thus, for budget reasons, we had to omit the road marking evaluations.

Back in 2012 when this project was initiated, the automatic mapping of gaze data to static and dynamic objects in the simulation was a novel idea. Since then, commercial solutions such as Noldus DriveLab[1], have been launched. Such solutions offer the same functionality as we have tried to develop in this project. We look forward to the advancements in visual behaviour research that will follow in the wake of these new tools.

This report presents the undertaken work from a technical point of view, including a brief discussion as well as suggestions for future work to improve the developed software. The technical parts describe the development of the VIMSI application as well as the additional work needed on the graphics engine to present the desired information.

Participants from VTI were Laban Källgren, Björn Blissing, Christer Ahlström, Carina Fors and Andreas Jansson.

Participants from SmartEye were Per Sörner and Kathrin Scheil.

Participant from the Swedish Road Marking Association was Göran Nilsson.

Participant from the Swedish Transport Administration was Ruggero Ceci.

Linköping, May 2017

Andreas Jansson

[1] http://www.noldus.com/innovationworks/products/drivelab/

Quality review

Peer review was performed on 7 June 2017 by Erik Olsson, VTI and on 15 June 2017 by Tania Dukic Willstrand, VTI. Andreas Jansson has made alterations to the final manuscript of the report. The ViP Director Lena Nilsson examined and approved the report for publication on 4 September 2017.

Table of contents

Abbreviations

AABB	Arbitrary Axle-oriented Bounding Box.
CORE	Commander of Road or Rail Environments. VTI's simulation core, responsible for the data traffic, adding actors and logging data.
IPC	Inter-process communication.
OBB	Oriented Bounding Box.
QT	Cross-platform application for developing application software.
UDP	User data protocol.
VIMSI	Fully linked visual behaviour measurements in a simulated world. VIMSI is a name, not an acronym.
VISIR	Visual Simulation of Road or Railway. Name of the graphics engine used at VTI.
XML	Extensible Markup Language (XML) is a markup language to encode documents in a format which is both human-readable and machine-readable.

List of figures

List of algorithms

Linking gaze tracking data to a simulated world

by Andreas Jansson[1] and Christer Ahlström[1]

[1] Swedish National Road and Transport Research Institute (VTI)

Executive summary

The main focus of this study was to develop a software able to link eye tracking data to simulator data, making it possible to automatically detect what the driver is looking at in the simulated world. This was achieved by merging data from a SmartEye system with data from the simulator. Thereby real-time visualisation of where the driver is looking is facilitated, and what the driver is targeting can be shown to the test leader to trigger events in the scenarios, etc. This also facilitates automatic gaze annotations that can be used in subsequent analyses when studying visual behaviour.

The developed software, VIMSI, is responsible for collecting eye tracking data from SmartEye, filter and aggregate this data with data from the simulation and then send processed data to the graphics to visualize the result. The software was tested in one of VTI's driving simulators. This initial testing of VIMSI showed that the software is capable of visualising what the driver is looking at in real time. The software also makes it possible to log data from the test drives which can be useful when studying driver behaviour. To improve the functionality of VIMSI, it is necessary to consider using UDP instead of IPC when directing data from the VIMSI software to the graphics engine VISIR. This will enable the use of another software, ScenarioReplay, developed at VTI for replaying test drives. A proper configuration and calibration of the SmartEye cameras is necessary to achieve high quality of the data from the SmartEye Pro software, which is a precondition for VIMSI.

1. Introduction

Analysing visual behaviour is an increasingly important part of traffic safety research. Application areas include glance behaviour analyses in relation to secondary task engagements such as texting and interacting with navigation systems (Kircher et al., 2014), assessment of medical (Kasneci et al., 2014) or temporary (Rogé et al., 2002) visual deficits, crash causality investigations (Victor et al., 2015), and conspicuity of road equipment (Dukic et al., 2013). Eye tracking clearly plays an important part in such investigations. An issue with eye tracking has always been the manual, daunting and very time-consuming task of encoding what the driver actually is looking at. If this process could be automated, more advanced glance behaviour analyses could be performed for a fraction of the cost.

For many years, it has been possible to create a world model of the vehicle cockpit and map the driver's gaze direction to the object in the vehicle that s/he is looking at (Figure 1). A natural extension of this functionality would be to map the gaze direction to objects in the driving environment. Such objects are for example other vehicles, vulnerable road users, traffic signs and road markings. In real world applications, we simply have to wait until the field of machine vision is mature enough to recognize everything of interest in the road scene. In driving simulators, working knowledge about the surrounding traffic environment is easily available, why such a functionality is possible to implement already today.

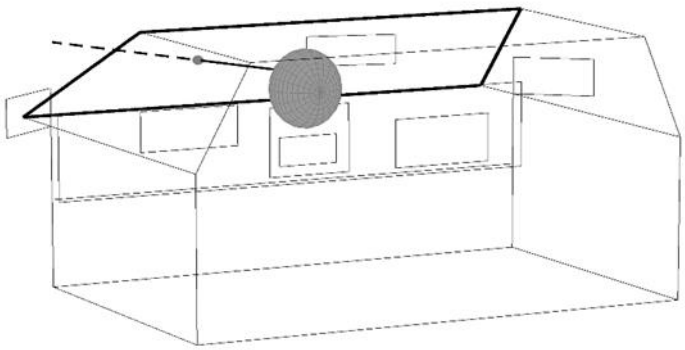


Figure 1. Mapping of the driver's gaze to an object (the windshield) in the car.

A prototype able to visualize the driver's eye movements and gaze direction was implemented by Nordenrot (2009) in the advanced driving simulator Sim III[2] at VTI (see Figure 2). The work by Nordenrot also highlighted a few problems that are hard to solve, for example eye tracking quality, the time offset that might occur in real time between the eye tracking and the visual system, and how to compensate for independent movements between the car cockpit and the projection screen.

In the ViP project "VisualEyes – Head- and eye behaviour measurement and visualisation in simulators" (Ahlström et al., 2010) the intention was to develop a tool (*eyeVis*) for real-time visualisation of eye gaze behaviour in the scenario (for test leader) and in the simulator vehicle (test driver) at the same time. An underlying assumption was that the developed tool should be easily adoptable to any simulator equipped with an eye tracker. Unfortunately, this generalization was very cumbersome to combine with a practically useful tool, why the project was not able to fulfil the real-time demand.

[2] https://www.vti.se/en/research-areas/vtis-driving-simulators/

More recently, the company Noldus has presented DriveLab[3]. DriveLab integrates a fixed-base driving simulator with a SmartEye system and allows for automatic gaze annotation of static objects within the vehicle as well as static and dynamic objects in the simulation.

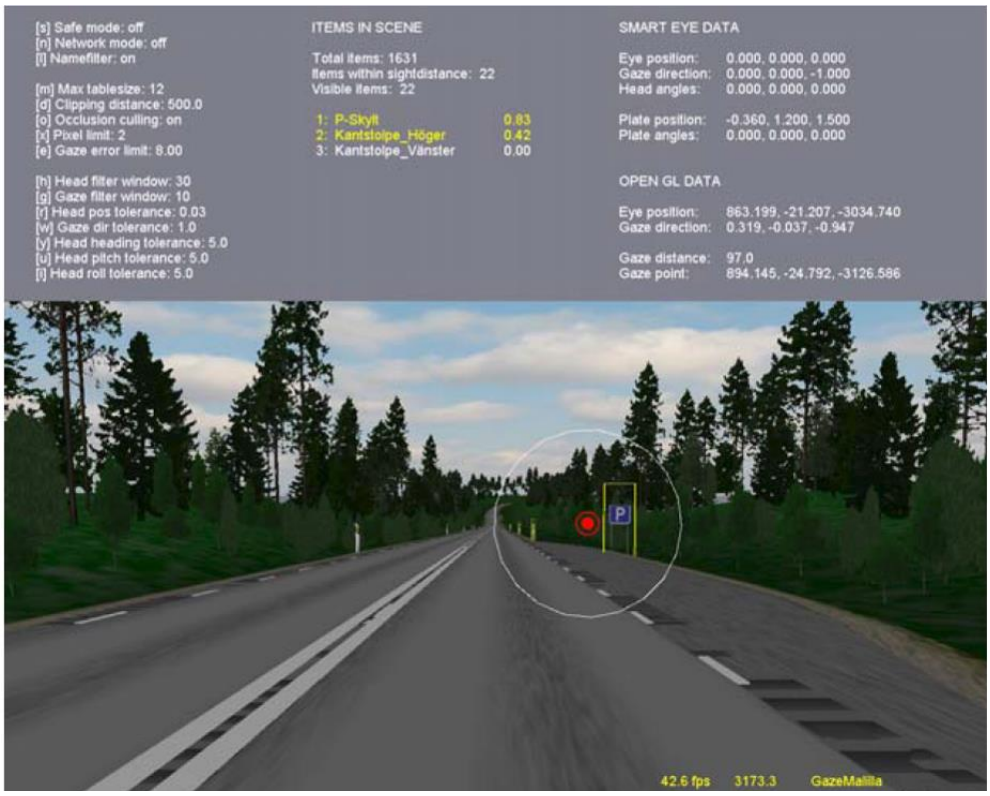


Figure 2. Screen shot from a prototype able to visualize the driver's gaze in the simulator graphics (Nordenrot, 2009).

1.1. Project objectives and goals

The objective of the VIMSI project was to design and implement a visualization tool for eye gaze measurement systems to be used in simulator studies. More specifically, the developed software should:

1. Add information to the simulator log file about the objects that the driver is/has looking/looked at during a driving session.
2. Make it possible to replay the test drive with a marker indicating where the driver was looking.
3. Visualize the driver's gaze target in real time.
4. Be implemented in the advanced driving simulators Sim III[4] and Sim IV[4] at VTI.

[3] http://www.noldus.com/innovationworks/products/drivelab/
[4] https://www.vti.se/en/research-areas/vtis-driving-simulators/

2. Method and realization

There are some crucial demands for the VIMSI application to work successfully. The simulator data from the vehicle and the data from the eye tracker (SmartEye Pro ver. 6.1.4[5], SmartEye AB, Gothenburg, Sweden) must be synchronized and sent in the same network package to the graphics. There has to be an interpolation of the data due to different frequencies for the two systems. This is one task the VIMSI application will take care of (Section 2.1). The graphics engine, VISIR[6], has to be modified as well in order to show the eye tracking data on the screen. An algorithm in VISIR for identifying which object the driver is looking at has also to be implemented (Section 2.2).

The VIMSI system architecture is shown in Figure 3. Four SmartEye cameras monitor the driver and the resulting videos are processed in the SmartEye Pro software to obtain the eye tracking data. This eye tracking data is sent to VIMSI via UDP. The Eye tracking data sends two packets to VIMSI. One containing filtered data and one containing raw data. These packets are unpacked, processed and aggregated into one packet. This combined packet with processed SmartEye data is aggregated with data from the simulation kernel, CORE. The aggregated data, containing simulator data from CORE and SmartEye data, are sent to the VISIR application via IPC. The simulator data is used to place the actors in the graphical environment while the SmartEye data is used for presenting the driver's gaze direction. The gaze direction data is finally passed on to VISIR which shows the results on a screen.

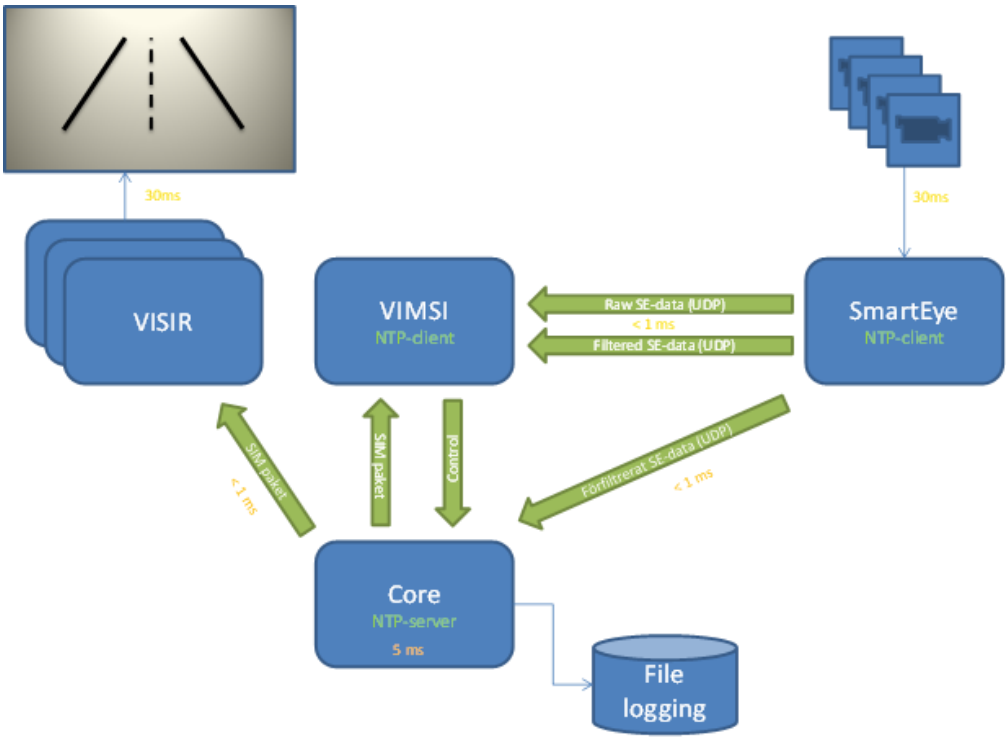


Figure 3. VIMSI system architecture.

[5] http://smarteye.se/research-instruments/se-pro/
[6] VISIR is the graphical engine in the ViP platform.

The VIMSI system design is shown in Figure 4. The large rectangle describes the VIMSI application's design and what tasks it can handle: compensation for head rotations, save data to log files and send data to VISIR are some of VIMSI's tasks. Each block is further described in Sections 2.1 to 2.3.

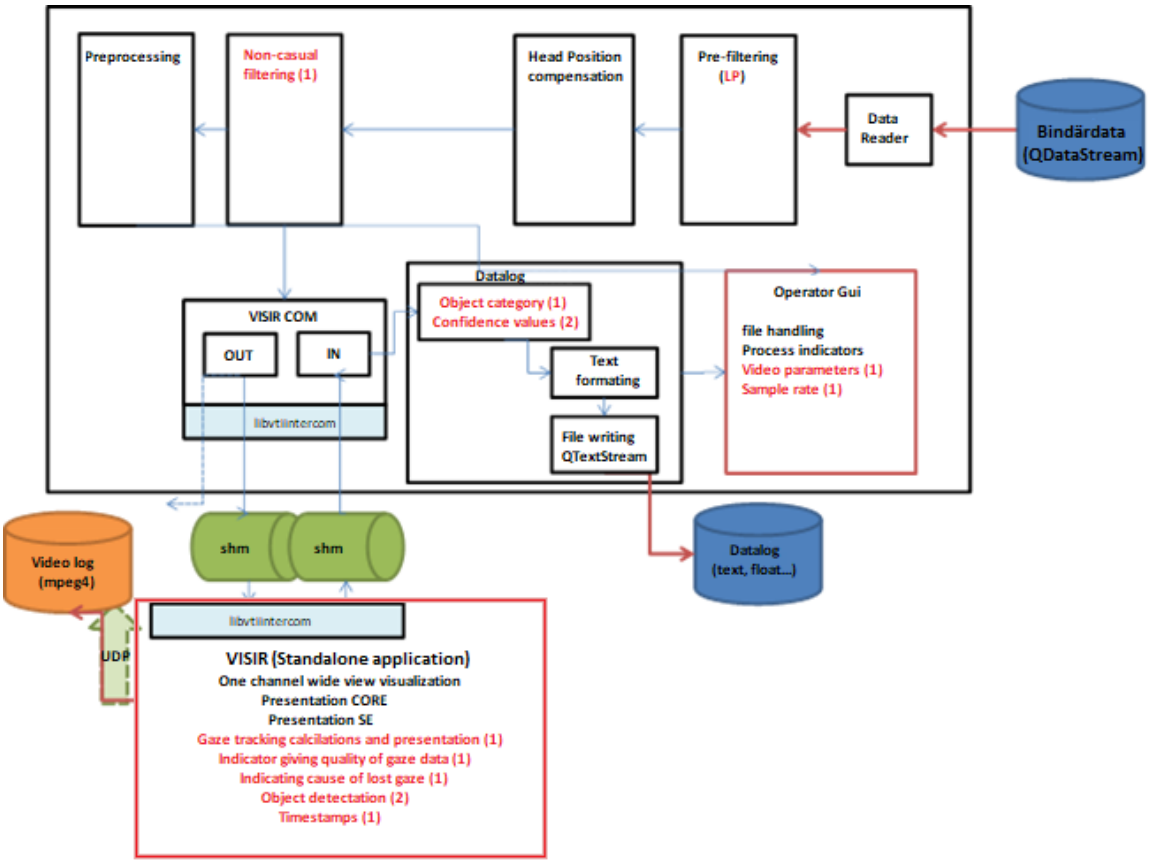


Figure 4. VIMSI system design.

2.1. Synchronization between the CORE and the VIMSI software

Four cameras are positioned in the cabin in a manner aiming to decrease the risks from getting low quality gaze data. The cameras cannot lose sight of the driver's eyes at any time in order to maintain good quality of the data. This can be achieved by containing the driver's field of view as much as possible, regardless of if s/he is looking out of the side windows or in the rear-view mirrors. The quality of the gaze data is one of many measurements logged by the SmartEye software. Gaze directions, head rotations/positions are other examples of data available from the software. Every dataset has a timestamp which is used by VIMSI to aggregate data from the simulation. Data from the simulation is mainly information about the vehicle's position, which is needed in VISIR to graphically update the vehicle's position.

The SmartEye Pro software streams two different data packages to VIMSI. One package contains filtered data and one contains raw data (see Figure 3). The gaze direction data is found in the packet with filtered data and the head position can be found in the packet with raw data. These two data sets are used to create a view direction vector in VIMSI. The view direction vector is a normalized vector of the result from a vector addition between the gaze direction and head position.

Data from the SmartEye system is, together with the simulation data, streamed to the VIMSI application with different frequencies. Data from the simulator (operating at 200 Hz) has to be synchronized with data from the SmartEye system (operating at 60 or 120 Hz). The issue with using both the filtered and the raw SmartEye data is the time delay in the filtered data. This is taken care of by up-sampling (interpolating) and aggregating the SmartEye data into one data packet. The view direction data is filtered with a median filter. The filtering is made in order to smooth out and steady the rapid eye movements from the data. The aggregated, processed SmartEye data is then interpolated and aggregated with the simulator data due to the differences in frequencies. This aggregated data package, containing time stamps, view direction, gaze quality and simulation data is then sent to VISIR.

The choice of using inter-process communication (IPC) instead of user data protocol (UDP) for sending data to VISIR was done because the IPC interface was already implemented.

2.2. Gaze mapping between VISIR and the VIMSI software

The items loaded into the graphics scenario (i.e. parked cars, road signs etc.) are saved in a list together with the road/environment they belong to. Each object located on the same road as the driver's current position is checked in a loop with an algorithm.

The algorithm behind finding which object the driver is looking at is a ray–object intersection algorithm. The ray is in this case based on the view direction from the driver, and its origin is located at the driver's eye position. A bounding box is created for each object for the application to do a ray intersection.

A bounding box is the theoretical minimal three-dimensional box needed to encapsulate an object. An arbitrary axle-oriented bounding box (AABB) is often used thanks to its simplicity, but the more advanced oriented bounding box (OBB) was implemented since high precision within the algorithm was desired. Figure 5 shows the comparison between the two bounding boxes in a two-dimensional presentation.

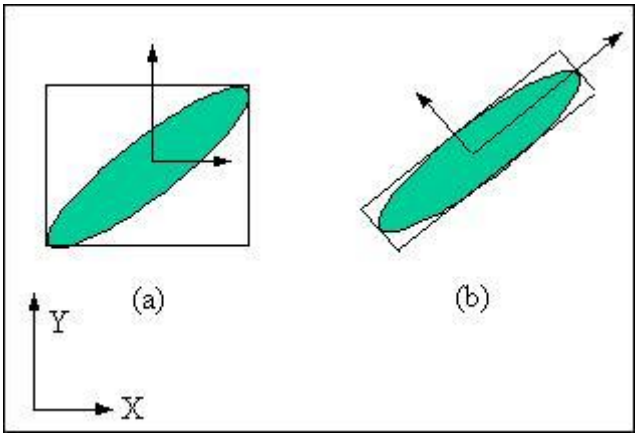


Figure 5. Two-dimensional comparison between AABB and OBB (Basdogan & Ho, 2001). The AABB (a) is a box sufficient to cover the object and the box is aligned with the X and Y axis. The OBB (b) is setup with another vector space to minimize the "dead space" inside the box.

The goal is to examine if the ray intersects with the current object's bounding box. The procedure is repeated for the next object if no intersection was found.

A ray which contains the points **p,** is defined as

$$\boldsymbol{r} = \boldsymbol{o} + t\boldsymbol{d} \qquad \forall t \geq 0 \qquad (1)$$

where **o** is where the ray starts and **d** is the ray's direction. The objects, which are defined as boxes, are set up by six planes which will be checked for intersection. A plane is defined by taking a point **p** together with its normal *n*. The goal is to find t in (1) so the following equation (2) is satisfied.

$$(\boldsymbol{r} - \boldsymbol{p}) \qquad n=0 \qquad (2)$$

Putting (1) in (2) together with some rewriting gives

$$t = \frac{(\boldsymbol{p}-\boldsymbol{o})\cdot n}{\boldsymbol{d}\cdot n} \qquad (3)$$

When t > 0 the ray is towards the plan and it will eventually intersect. The two other options are t = 0, which means that the ray has its origin in the plane, and t < 0 when the ray is away from the plane and there will not be an intersection.

In this case, where the object is a box instead of a plane, equation (3) is expanded to:

$$t_i = \frac{(\boldsymbol{p}_i-\boldsymbol{o})\cdot n}{\boldsymbol{d}\cdot n} \qquad i=1,2 \qquad (4)$$

The idea with (4) is to check two planes each time in each direction. t_1 will be the value for intersection with the object's closest plane and t_2 with the furthest. The algorithm works as shown in the pseudo code snippet in Algorithm 1 below. The view direction vector is a normalized vector of the sum of the vector containing data of the head position and a vector containing the gaze direction.

```
for x y and z {
    if view direction vector orthogonal in any direction {
       if the eyes position are not in between the objects min and max {
         no intersection
        }
    } else {
        Calculate t1 and t2

        if t1 > t2 {
            swap the values for t1 and t2
        }

        tmin = max(tmin, t1);
        tmax = min(tmax, t2);

        if tmin > tmax {
            no intersection
            break
        }
    }
}
Intersection, return intersection point
```

Algorithm 1. Pseudo code snippet for the object intersection algorithm.

Moving objects in the environment, i.e. pedestrians and moving cars, are actors in the scene and they need to be checked with the same algorithm as above (Algorithm 1), but in another loop. This is

because they are actors from the scenario designed and added by CORE. The static objects mentioned above are coded within the graphics scenario. It is possible for both the loops to find an object intersected by the driver's view. These two objects' positions are therefore compared and the closest one is assumed to be the object the driver is targeting. This target is presented on the screen which can be seen by the test leader and the targeted object's name is shown in the upper right corner of the screen. Figure 9 gives an example on what the test leader can see on his/her screen.

The resulting target based on the driver's view direction together with the current object hit by Algorithm 1 is presented on a monitor in real time. Today the results can only be presented in real time. Estimations done in these calculations are:

1. Objects located further away than 400 meters from the driver are ignored. This distance is set to prevent the algorithm from looping over more objects than necessary.
2. Objects whose sizes are tremendous compared to other objects are ignored. Examples of such objects are modelled apartment complexes which, in the graphics, are over 100 meters long and therefore extremely easy to be targeted by the driver's view direction.

2.3. Data logging

The data logging is handled in an own thread in order to maximize the performance of Algorithm 1. The projects XML file has to be modified in order to log desired data.

This XML file is called and read by VISIR. The file specifies the actors that are going to be included in the graphics, what light setting should be used, how the road network should look like and other graphical options specified for the project. One road network can consist of many modelled roads and VISIR links the roads specified in the XML file together in the same order as they appear. The roads themselves are specified in their own XML files.

The data the user wants to record has to be set in the project's .XML file under the *datalog* bracket (see Figure 6). This tells VISIR to start write a log file whenever VISIR starts. The logging rate and the log files' directory are specified under this datalog bracket. The parameters VISIR wants to read and log from the parameter map are specified under the same bracket. The parameter map works as a look-up table with key-value pairs. A function can report to this parameter map by writing any changes to a specific parameter which then can be read by another function. The basic idea on how it can be used in its simplest form can be described with this example:

We have calculated a value for a variable X in class A which we want to use in another class, class B. Instead of declaring new functions one can tell the function in class A to report any changes of variable X to the parameter map. It is also necessary to tell class B to read variable X from the parameter map. A function in class A writes the value of variable X in the parameter map which is read by a function in class B as soon it notices a change. The parameter map is used because class A and class B are within the same software and, therefore, it is not necessary to use any protocol for data transfer. The parameter map is not a default function and needs to be implemented in order to be used.

The data desired to log is added to the project's XML file in order for VISIR to know what project specific variables it has to work with (see Figure 6). Figure 7 shows a code snippet from the constructor (function initializing the object's member variables) for the class where the desired data is used. Member variables are told to report to the parameters specified in the project's XML file (seen in Figure 6), which they will do whenever there are any changes of their values. Figure 8 shows a code snippet where data is written to the member variables. VISIR has a class for data logging which runs in an own thread. The datalog bracket in Figure 6 initiates this function in VISIR when it starts and VISIR writes the specified parameters to a text file as soon as any changes have occurred.

```
<datalog rate="30" outputPath="C:\\visirdata">
    <parameter name="raytracing_sim_time"       type="unsigned long long"/>
    <parameter name="raytracing_viewd_x"        type="double"/>
    <parameter name="raytracing_viewd_y"        type="double"/>
    <parameter name="raytracing_viewd_z"        type="double"/>
    <parameter name="raytracing_object_name"    type="string"/>
    <parameter name="raytracing_road_id"        type="int"/>
    <parameter name="raytracing_object_pos_x"   type="double"/>
    <parameter name="raytracing_object_pos_y"   type="double"/>
    <parameter name="raytracing_object_pos_z"   type="double"/>
</datalog>
```

Figure 6. Datalog bracket in the project's .XML file. The parameters returned here are simulation time, the view direction vector, the targeted object's name (if any hit), the id for the road the object is positioned on, and the vector for the object's position.

```
class RayTracing : public osg::Referenced {
    public:
        RayTracing() :
            m_sim_time(util::ParameterMap::instance()->report<unsigned long long>("raytracing_sim_time")),
            m_viewd_x(util::ParameterMap::instance()->report<double>("raytracing_viewd_x")),
            m_viewd_y(util::ParameterMap::instance()->report<double>("raytracing_viewd_y")),
            m_viewd_z(util::ParameterMap::instance()->report<double>("raytracing_viewd_z")),
            m_object_name(util::ParameterMap::instance()->report<std::string>("raytracing_object_name")),
            m_road_id(util::ParameterMap::instance()->report<int>("raytracing_road_id")),
            m_object_pos_x(util::ParameterMap::instance()->report<double>("raytracing_object_pos_x")),
            m_object_pos_y(util::ParameterMap::instance()->report<double>("raytracing_object_pos_y")),
            m_object_pos_z(util::ParameterMap::instance()->report<double>("raytracing_object_pos_z"))
        {
```

Figure 7. Code sample showing initialization of the reported member variables.

```
    if (sim_time - m_previous_timestep > 0) {
        m_previous_timestep = sim_time;
        m_sim_time = sim_time;
        m_viewd_x = m_viewdirection.x();
        m_viewd_y = m_viewdirection.y();
        m_viewd_z = m_viewdirection.z();
        m_object_name = m_roadObject->name();
        m_road_id = m_roadObject->roadId();
        m_object_pos_x = m_closestRoadObjectPos.x();
        m_object_pos_y = m_closestRoadObjectPos.y();
        m_object_pos_z = m_closestRoadObjectPos.z();
    }
```

Figure 8. The data the user wants to log is written to the member variables in Figure 6.

The data logged, which can be seen Figure 10 below, are view direction and simulation time. The intersection point and the object's name are saved to the log if the algorithm detects an intersection between the view direction and an object. It is easy to add more data to the log if needed.

3. Output from VIMSI

Figure 9 shows how the graphics will look in real time during a test drive with VIMSI. The marker based on the eye tracking is shown in purple and white. The white area follows the driver's view direction while the purple ring shows the quality of the gaze data from the SmartEye software. The brighter the ring is, the higher the quality of the data from SmartEye. The quality is calculated based on the contrast of the edge between iris and sclera and is normalised. 0 corresponds to the 1st percentile of all collected quality values of the current trip, and 1 corresponds to the 99th percentile.

The screen shot in Figure 9 shows that the driver is currently looking at the parked truck, which is a yellow lorry. The graphical model of the truck is named *lorry_mercedes*. The name will show up in the upper right corner of the screen if the driver's view direction intersects with this truck, as can be seen in Figure 9.

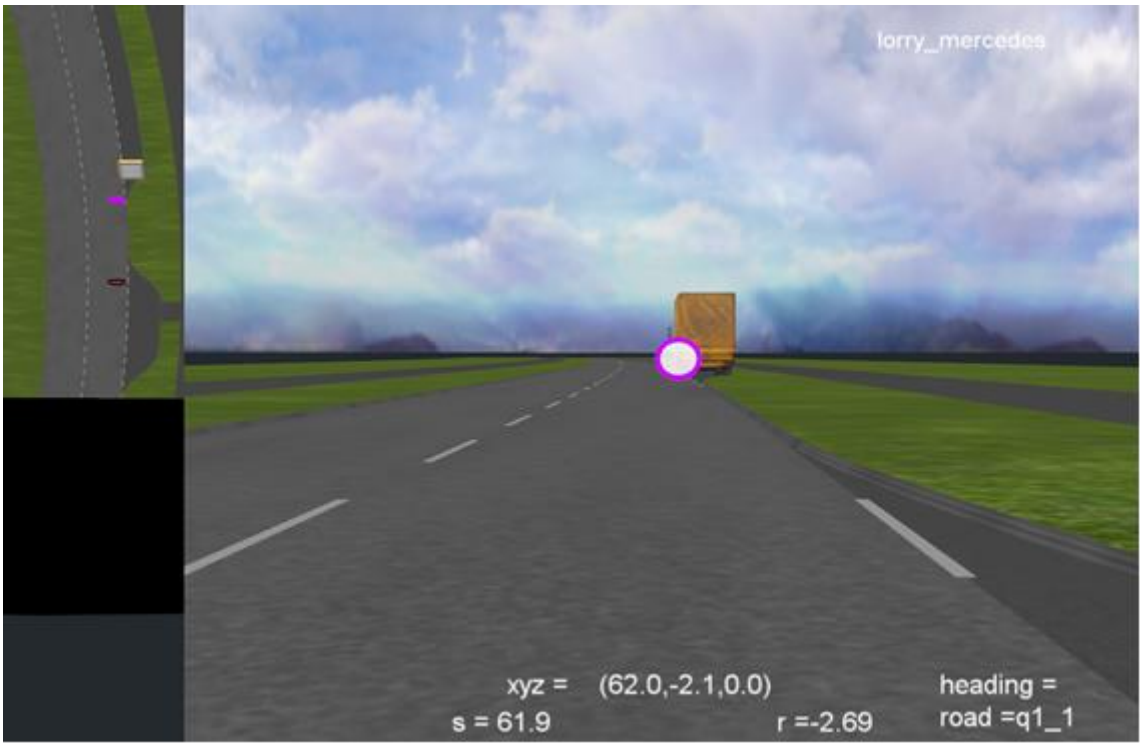


Figure 9. Screen shot of the graphics during a drive with the VIMSI application enabled. The gaze target (white dot) shows that the driver looks at the yellow lorry. The upper left shows the scene from a bird's eye view.

Figure 10 shows a couple of lines from a data log. The data shown are the components of the gaze vector and the object's name if the gaze targets an object.

```
raytracing_sim_time raytracing_gaze_x   raytracing_gaze_y   raytracing_gaze_z   raytracing_object_name

1461700763874   0.998429    0   0.000328795 car_audi_a4
1461700763879   0.977629    -0.185131   0.000585852 lorry_mercedes
1461700763884   0.972949    -0.208161   0.000587901
1461700763889   0.962816    -0.239514   0.000733312
1461700763894   0.966576    -0.223951   0.000732322
1461700763899   0.961664    -0.222813   0.000938029
1461700763904   0.948262    -0.239378   0.00122375
1461700763909   0.926136    -0.313007   0.00123497
1461700763914   0.927024    -0.322931   0.00111858
1461700763919   0.878826    -0.264967   0.00232833
1461700763924   0.895789    0   0.00260805      0   0
1461700763929   0.992001    0   0.000740675 car_audi_a4
1461700763934   0.982816    -0.135763   0.000733817 lorry_mercedes
```

Figure 10. Lines from datalog showing eye tracking data together with targeted objects.

4. Discussion

Test drives were done in the simulator without the motion system activated to control if the object intersection worked satisfactory. The driving environment was a city centre with different amounts of actors. The easiest test is presented in Figure 9 with one single actor, a big truck. More actors and objects, such as traffic signs, vehicles approaching in the opposite lane, parked vehicles and pedestrians, were added in later tests. The outcome from these tests were that the algorithm could detect the objects in the graphical environment. The software also made it possible for the test leader to watch where the driver was looking in real time. It was also possible to log data from the test drives. This means that criteria 1 and 3, specified in Section 1.1 “Project objectives and goals”, are met. The tests were done in Sim III, but they could have been done in Sim IV as well since also Sim IV has SmartEye cameras installed and uses VISIR. Thus, criteria 4 can be met after tests have been performed in Sim IV.

A necessary precondition for VIMSI to work as intended is high quality eye tracking data. Careful camera calibration is therefore of immense importance. A problem in the simulation environment was that even if the calibration procedure was carried out just before the drive, there were still issues with insufficient quality of the data when the driver’s field of view increased, for example when looking out through the right window. This problem arose for more extreme gaze angles where the coverage of the SmartEye cameras is insufficient.

The eye tracking algorithm was directly implemented into VISIR. This solution has the advantage that the test leader does not have to start another software than the VIMSI application in QT when running test drives. The downside is if the test leader wants to change anything in the graphics (such as size and colour of the marker for view direction) or data log. Such a change would have to be done directly in the source code and VISIR would have to be recompiled.

5. Future Work

We intend to incorporate the VIMSI functionality with the ScenarioReplay application, which is used to replay a driving scenario. This means that it will be possible to analyse drivers' test runs after they have been completed. ScenarioReplay was developed during a late phase of the VIMSI project for another project, which is why the UDP was not implemented in the first place. The reason behind the development of ScenarioReplay was to ease the analysis of the drivers' performance in the smaller simulators delivered to customers. ScenarioReplay made it convenient for other users to find answers in test drives that could not be described in reports created based on logged data.

Using IPC, which is a shared memory, means that the results of real-time data must be shown on the same computer as the VIMSI application is running on. The software development was done in a cross-platform application called QT creator. Then it is possible to use QT creator's own defined classes, of which one is QT's UDP class. This approach could make it possible to modify the VIMSI application so it uses UDP instead of IPC. A goal for future work is to implement the VIMSI environment so it uses the UDP protocol. This is needed for the VIMSI application to work together with ScenarioReplay.

The object intersection algorithm could be improved by introducing a function that calculates the probability of what objects the driver is targeting. So instead of returning one targeted object or nothing at all, the function would return a list of, for example, 5 objects with a calculated probability for each object.

A limitation of the current version of VIMSI is that it has not been tested properly. Since the VIMSI software is intended to function in a variety of future projects without additional development, it is necessary to test the developed software in a systematic manner. There are many components that influence the final result. Thus, rigorous testing should be conducted at different levels (unit tests, integration tests, validation tests and system tests). It will also be necessary to test the system with a wide range of participants to see how the system works when eye tracking quality deteriorates. To date, only ad hoc tests have been carried out. Proper quality assurance is still something that needs to be done in future developments of VIMSI.

Finally, there are many difficulties when linking the gaze vector to a simulated object. One of these difficulties is that the simulator's virtual 3D world is projected on a surface relatively close to the test driver, assuming that the driver is sitting in a certain position. If the driver is moving, the mapping between the real 3D world and how the gaze vector continues into the simulated world will change with the 3D position of the driver's eyes. Another difficulty (in Sim III) is that the car is mounted on a vibration table that moves independently of the projection screens. To increase the accuracy of the mapping between gaze direction and simulated objects, these movements should be compensated. These difficulties in linking the gaze vector to a simulated object should be addressed in future work.

References

Ahlström, C., Dukic, T., Ivarsson, E., Kircher, A., Rydbeck, B., & Viström M. (2010). *Performance of a one-camera and a three-camera system.* ViP publication 2010-1. Linköping, Sweden: Swedish National Road and Transport Research Institute (VTI).

Basdogan, C., & Ho, Chih-Hao. (2001) *Principles of Haptic Rendering for Virtual Environments.* Notes from the Basdogan's personal web page on Koç University's web page.

Dukic, T., Ahlström, C., Patten, C., Kettwich, C., & Kircher, K. (2013). Effects of electronic billboards on driver distraction. *Traffic Inj Prev, 14*(5), pp. 469-476. doi: 10.1080/15389588.2012.731546.

Kasneci, E., Sippel, K., Aehling, K., Heister, M., Rosenstiel, W., Schiefer, U., & Papageorgiou, E. (2014). Driving with binocular visual field loss? A study on a supervised on-road parcours with simultaneous eye and head tracking. *PLoS One, 9*(2): e87470. doi: 10.1371/journal.pone.0087470.

Kircher, K., Ahlström, C., Rydström, A., Ljung Aust, M., Ricknäs, D., Almgren, S., & Nåbo, S. (2014). *Secondary Task Workload Test Bench - 2TB - Final Report*. ViP publication 2014-1. Linköping, Sweden: Swedish National Road and Transport Research Institute (VTI).

Nordenrot, M. (2009). *Realtidsvisualisering av ögonrörelser i simulator*. VTI notat 7-2009. Linköping, Sweden: Swedish National Road and Transport Research Institute (VTI) (in Swedish).

Rogé, J., Pébayle, T., Kiehn, L., & Muzet, A. (2002). Alteration of the useful visual field as a function of state of vigilance in simulated car driving. *Transportation Research Part F: Traffic Psychology and Behaviour, 5*(3), pp. 189-200. doi: http://dx.doi.org/10.1016/S1369-8478(02)00017-7.

Victor, T., Dozza, M., Bärgman, J., Boda, C.-N., Engström, J., Flannagan, C., Lee, J.D., & Markkula, G. (2015). *Analysis of Naturalistic Driving Study Data: Safer Glances, Driver Inattention, and Crash Risk*. Transportation Research Board, S2-S08A-RW-1, Washington, D.C.

ViP

Driving Simulation Centre

ViP is a joint initiative for development and application of driving simulator methodology with a focus on the interaction between humans and technology (driver and vehicle and/or traffic environment). ViP aims at unifying the extended but distributed Swedish competence in the field of transport related real-time simulation by building and using a common simulator platform for extended co-operation, competence development and knowledge transfer. Thereby strengthen Swedish competitiveness and support prospective and efficient (costs, lead times) innovation and product development by enabling to explore and assess future vehicle and infrastructure solutions already today.

ViP
Driving Simulation Centre

Centre of Excellence at VTI funded by Vinnova and ViP partners

VTI, Scania, Volvo Trucks, Volvo Cars, Swedish Transport Administration, Dynagraph, Empir, HiQ, SmartEye, Swedish Road Marking Association

www.vipsimulation.se

Olaus Magnus väg 35, SE-581 95 Linköping, Sweden – Phone +46 13 204000