FFI-rapport 2012/00490

Design and Implementation of an Augmented Reality System for VBS2

Per-Idar Evensen

Norwegian Defence Research Establishment (FFI)

29 March 2012

FFI-rapport 2012/00490

115602

P: ISBN 978-82-464-2058-5
E: ISBN 978-82-464-2059-2

Keywords

Augmented Reality

VBS2

Spillbasert simulator

Modellering og simulering

Approved by

Halvor Ajer	Project Manager
Johnny Bardal	Director

English summary

This report describes the implementation of an Augmented Reality (AR) system for Virtual Battlespace 2 (VBS2). The AR system is developed using VBS2Fusion, which is a C++ based application programming interface (API) for VBS2. VBS2Fusion has functionality that makes it possible to implement graphical overlays in VBS2.

The AR system is designed for use in combat vehicles like Infantry Fighting Vehicles (IFV) and Main Battle Tanks (MBT), and gives the commander, gunner and driver information in the form of graphical symbols in their sights and periscopes. The system works together with an experimental Battlefield Management System (BMS) developed at FFI, and visualizes information from the BMS.

The AR system was developed in order to be able to experiment with the design of AR functionality in combat vehicles in a simulated environment, and to evaluate the benefit of this kind of functionality in combat. The system was used in a large experiment in FFIs Battle Lab in November 2011.

Sammendrag

Denne rapporten beskriver implementeringen av et Augmented Reality (AR) system for Virtual Battlespace 2 (VBS2). AR-systemet er utviklet ved hjelp av VBS2Fusion, som er et C++-basert programmeringsgrensesnitt (API) mot VBS2. VBS2Fusion har funksjonalitet som gjør det mulig å implementere grafiske overlegg i VBS2.

AR-systemet er laget for bruk i militære kjøretøy som stormpanservogner (IFV) og stridsvogner (MBT), og gir vognkommandør, skytter og vognfører informasjon i form av grafiske symboler i sine siktebilder og periskop. Systemet virker sammen med et eksperimentelt Battlefield Management System (BMS) utviklet ved FFI, og visualiserer informasjon fra BMS-et.

AR-systemet ble utviklet for å kunne eksperimentere med utforming av AR-funksjonalitet i stridkjøretøy i et simulert miljø, og å evaluere nytten av slik funksjonalitet i strid. Systemet ble brukt i et stort eksperiment i FFIs Battle Lab i november 2011.

Contents

1 Introduction

The work with designing and implementing an Augmented Reality (AR) system for Virtual Battlespace 2 (VBS2) has been done under FFI-project 1156, “Technologies for Military Vehicles”. The simulated AR system was used in a large simulator experiment in FFIs Battle Lab in November 2011, where six professional Infantry Fighting Vehicle (IFV) crews from the Telemark Battalion (TMBN) participated for one week. The experiment consisted of a number of relevant scenarios for combat vehicles played in VBS2.

The purpose of this experiment was to evaluate the usefulness of AR functionality in combination with a Battlefield Management System (BMS) in combat vehicles. The observations we did in this experiment, and the feedback we got from the users, will be used as input to the ongoing process of designing a real AR system for combat vehicles. The results from this experiment are described in [1]. The details of how the experiment was conducted are described in [2].

At FFI we first started to experiment with simulated AR in an in-house developed combat vehicle simulator called NORBASE (Norwegian Battle Simulator Experiment) in 2006. NORBASE was based on the commercial game Unreal Tournament 2004, and was used in an experiment in November 2006 [3]. Figure 1.1 shows images from the AR system in NORBASE. The AR objects in NORBASE where drawn directly into the scene, and not as a graphical overlay as we now do in it VBS2.

Later NORBASE was replaced by VBS2, and in 2009 an attempt was made to develop an AR system for VBS2 for an experiment in May 2009 [4]. At that time VBS2 had very limited functionality for drawing graphical overlays, and there were no easy way to map between world coordinates and screen coordinates using the VBS2 scripting language. The AR system therefore became very primitive, and did not have all the desired fuctionality. It was first when VBS2Fusion version 2.0 was released at the end of 2010 that it became possible to implement an AR system for VBS2 which included all the functionality that we wanted to test out. VBS2Fusion is a C++ based Application Programming Interface (API) for VBS2. From version 2.0 it has functionality that makes it possible to develop graphical overlays in VBS2.

Our simulated AR system is designed for use in combat vehicles like Infantry Fighting Vehicles (IFV) and Main Battle Tanks (MBT), and gives the commander, gunner and driver information in the form of graphical symbols in their sights and periscopes. The system works together with an experimental Battlefield Management System (BMS), also developed at FFI, and it visualizes information like Blue Force Tracking (BFT) and observations received from the BMS. The vehicle crew can thus keep their eyes constantly on the battlefield in critical situations, and at the same time get important information from the BMS.

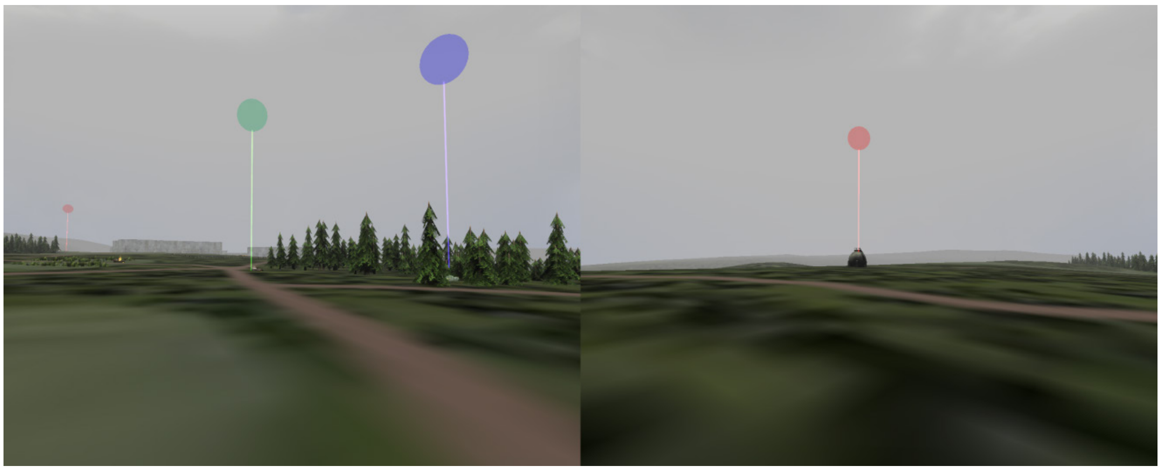


Figure 1.1 The AR system in NORBASE from the early experiments with simulated AR in 2006.

This report first gives a general description of Augmented Reality in Chapter 2. Here we look at the underlying technology, and some applications of AR systems. Next, in Chapter 3, we describe the training and simulation toolkit VBS2, and the C++ based API VBS2Fusion. In Chapter 4 we describe and illustrate the functionality in our simulated AR system for combat vehicles. Finally, the details of the implementation of the AR system are described in Chapter 5.

The AR system described in this report was developed and tested with version 1.5 of VBS2 and version 2.5.2 of VBS2Fusion. This report describes the AR system and how it was implemented at the time this report was published.

2 Augmented Reality

Augmented Reality (AR) is a technology where virtual, computer generated data are added to the data we perceive from the real world, in real time. This gives the user an augmented perception of the reality. Mainly AR means adding virtual objects in the form of computer graphics to visual data from the real world. The virtual objects typically add information that helps the user perform real-world tasks.

The phrase “Augmented Reality” is supposed to have been coined by Thomas P. Caudell in 1990, while working at Boing helping workers to assemble cables into aircrafts, but the concept has been around before this. More general information about AR can be found in [5] and [6]. In these papers, AR is defined as a system that has the following three properties:

1. Combines real and virtual objects in a real environment.
2. Runs interactively, and in real time.
3. Registers (aligns) real and virtual objects with each other.

AR is not necessarily limited to our sense of sight, and can potentially apply to other senses like hearing, touch and smell. Some AR applications also require removing real objects from the perceived environment, in addition to adding virtual objects. A key measure of AR systems is how realistically they integrate the augmentations with the real world.

Examples of areas where AR has been applied are medical visualization, radiation and hazard visualization, assembly, maintenance and repair, visualization within architecture and construction, navigation in aircraft and cars, and entertainment. Conceptually, anything not detectable by human senses but detectable by sensors might be transduced into something that a user can perceive in an AR system. Figure 2.1 shows some examples of AR applications. In Section 2.2 we take a closer look at military applications of AR.

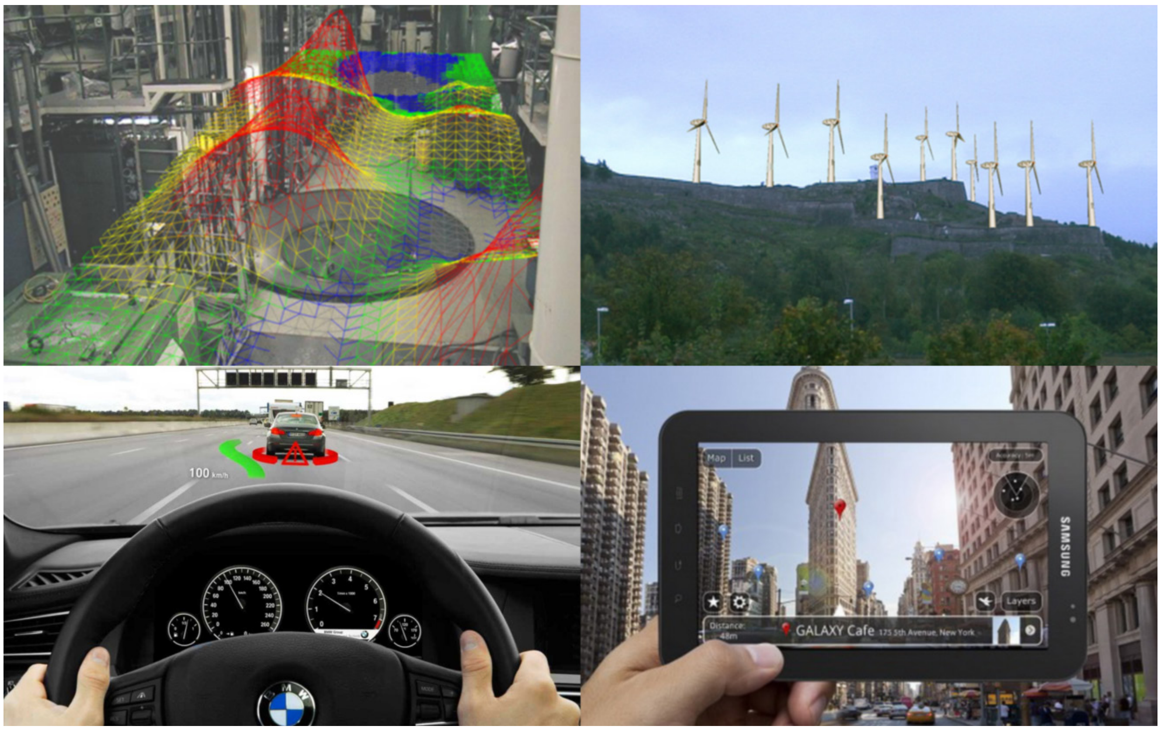


Figure 2.1 Examples of AR applications: visualization of radiation (AR-Lab), visualization of windmills (AR-Lab), HUD in cars (BMW) and smartphone apps (Samsung).

2.1 Technology

An AR system consists of the following components:

- A display device that shows the real world together with the virtual objects. This can be a monitor, a Head-Mounted Display (HMD) or a special optical see-through HMD. If a monitor or a standard closed-view HMD is used, a video camera is needed to capture the real scene. An optical see-through HMD is partially transmissive, so that the user can look directly through it and see the real world. It can also display images from a computer.

- A tracking system for accurately tracking the user's viewing orientation and position. The system needs this information to calculate the position and orientation of the virtual objects. Possible technologies for a tracking system are digital cameras or other optical sensors, GPS receivers and Inertial Measurement Units (IMU).
- A virtual scene generator to render the virtual objects at the correct positions. This is usually a computer with AR software.

In addition an AR system might include devices for interaction with the system. Figure 2.2 illustrates the concept behind an AR system. To get really good AR systems, where the virtual objects appear so realistic that they are virtually indistinguishable from the real environment, there is need for further development and new technology for both display devices and tracking systems.

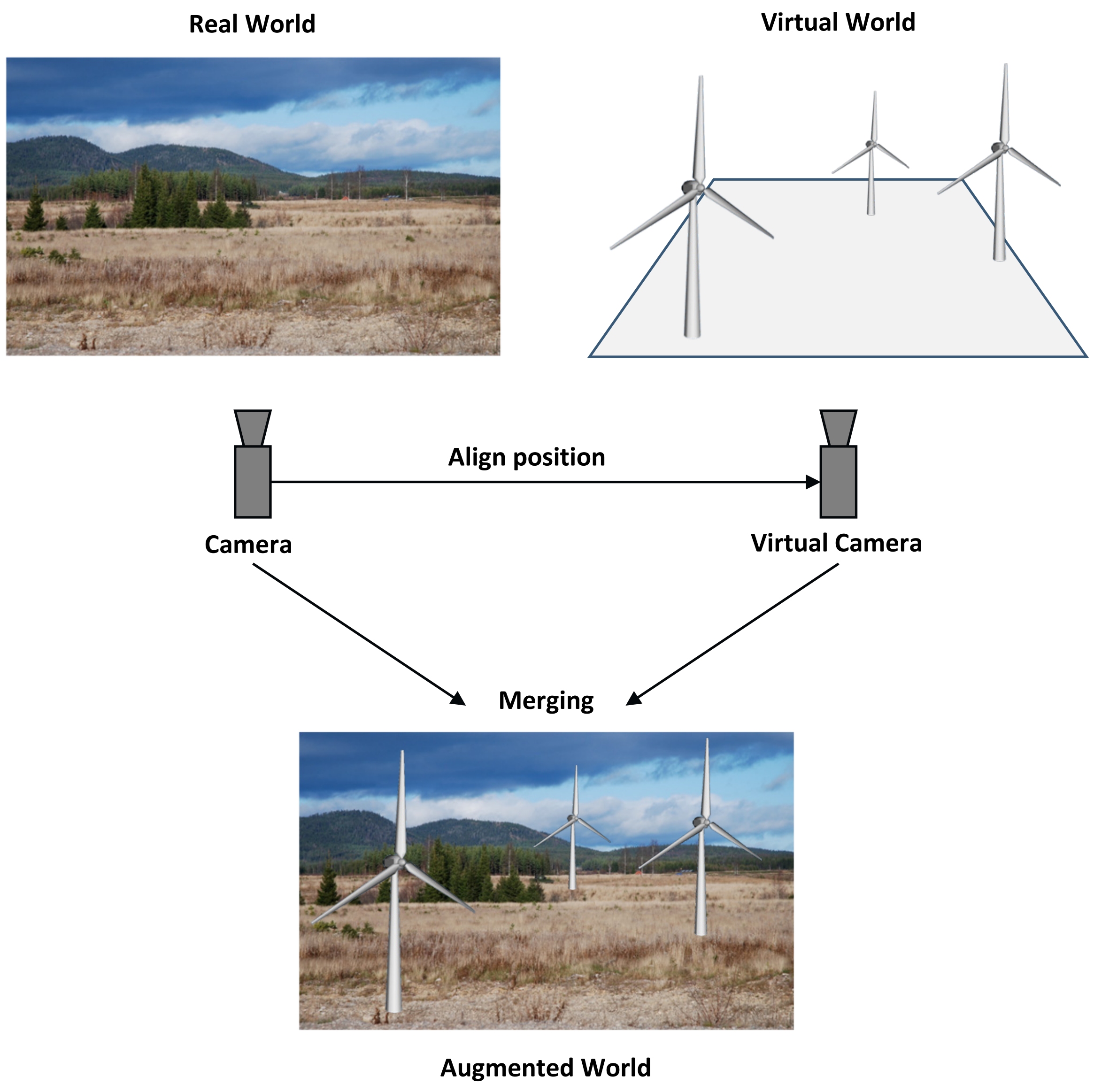


Figure 2.2 The concept behind an AR system.

2.2 Military applications

In military aircraft and helicopters pilots have been using Head-Up Displays (HUD) and Helmet-Mounted Sights (HMS) with AR for several years. These systems provide the pilot with navigation and flight information, and for some systems ground or air targets can be marked with graphics.

AR systems that display tactical battlefield information are now being developed for combat vehicles and ground soldiers. Typically these systems will be used to increase situational awareness through visualization of Blue Force Tracking (BFT) data and points of interest like observations and targets. In the future, sensor platforms and rapid information sharing over networks will probably make it possible to develop AR systems that can provide close to real-time red force tracking on the battlefield.

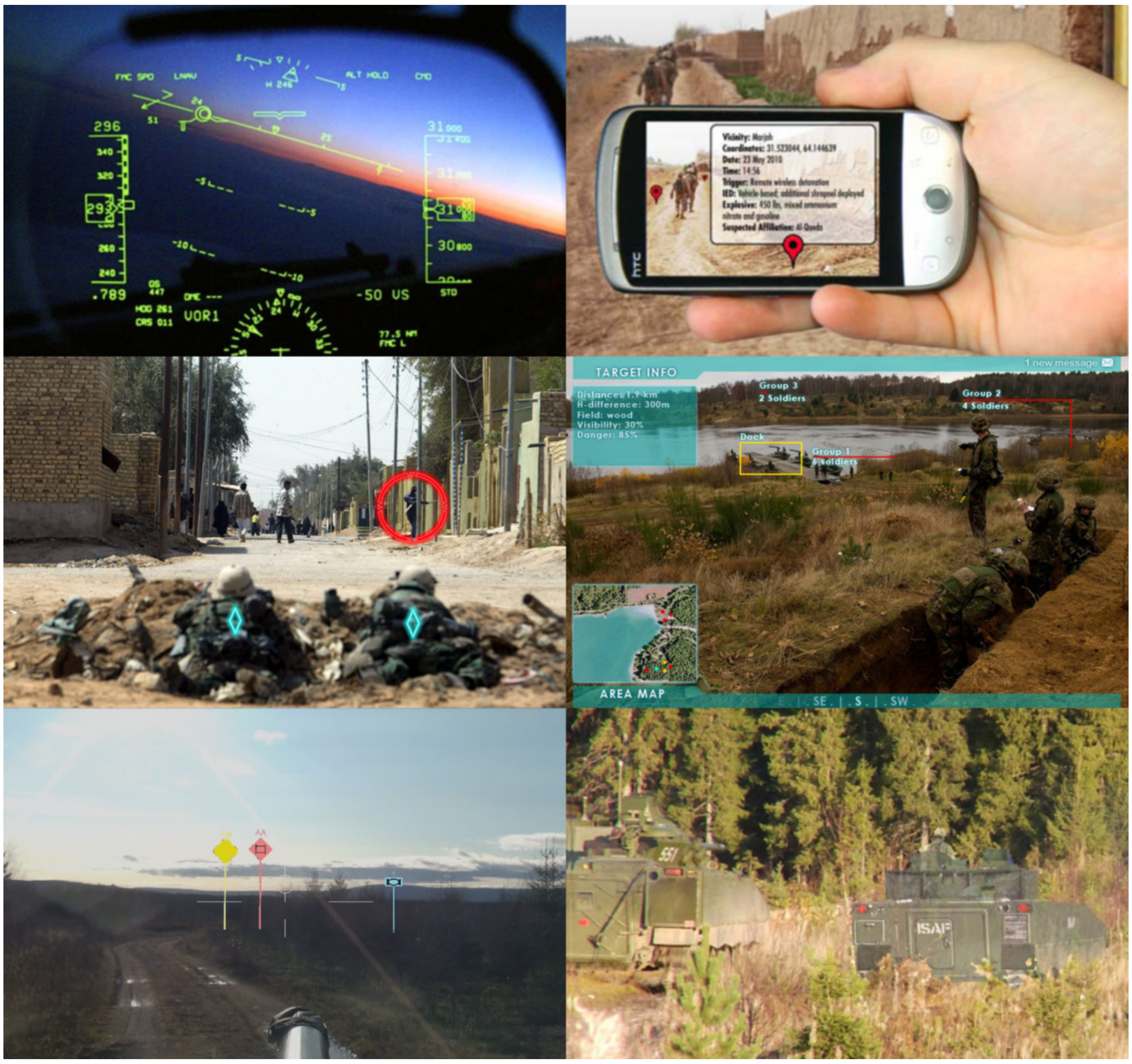


Figure 2.3 Examples of military applications of AR: HUD in aircraft, IED Threat Locator [7], ground soldier systems (Microvision and Contactum), combat vehicle system, and exercises with real and virtual units.

AR is also being used within military training, but so far mostly for experimentation purposes. Here it has been used to visualize virtual enemies in the real world. It has also been used to combine live and simulator based training, where real soldiers and vehicles can train together with virtual forces operated from simulators in a two-way real-time interaction between live and virtual units. Tools for Computer Generated Forces (CGF) can also be used to control the virtual forces. In such applications one of the biggest challenges of both vehicle mounted and wearable AR systems is the need for accurate spatial tracking. The position and orientation of all participants must be accurately tracked over the whole training field. Even with good spatial tracking the occlusion (hiding) of virtual objects that are positioned completely or partially behind live objects is very hard to visualize correctly. Figure 2.3 shows images from some examples of military applications of AR.

2.3 Simulated AR

In computer games it is common to insert additional graphical objects into the virtual scene, and the player is often equipped with a simulated HUD with an AR system. This can be used for visual directions to a location, information about objects and highlighting items in the virtual world. Figure 2.4 shows examples of AR in computer games. The computer game Frontlines: Fuel of War from Kaos Studios uses a simulated HUD with an AR system to display tactical battlefield information, like blue force and red force tracking and waypoints. In the computer game Heavy Rain from Quantic Dream, the player can use special glasses with an AR system that aids in the crime scene investigation.

Our simulated AR system for VBS2 adds graphical objects to the virtual scene in VBS2. The system is designed to look like real AR systems, and the graphical objects are drawn in a two-dimensional graphical overlay and not directly into the three-dimensional virtual scene. We also simulate the end-to-end system delay that is present in real AR systems and causes registration errors when motion occurs.

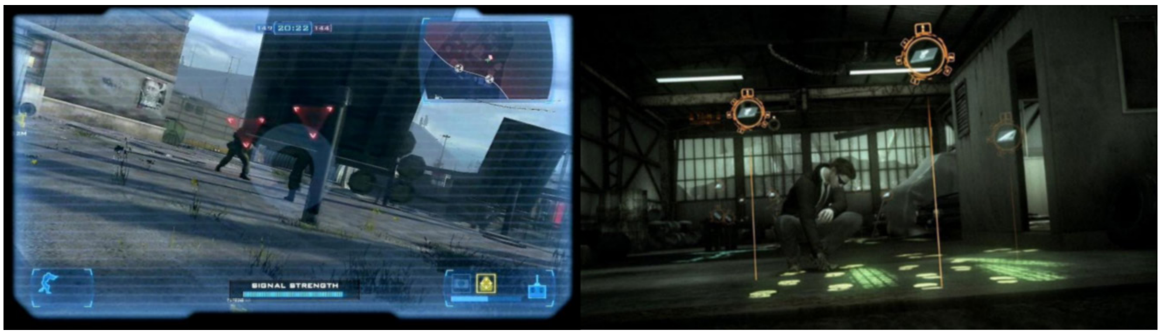


Figure 2.4 Examples of AR in computer games: Frontlines: Fuel of War (Kaos Studios) and Heavy Rain (Quantic Dream).

3 Virtual Battlespace 2

Virtual Battlespace 2 (VBS2) is a game-based[1], fully interactive, three-dimensional, synthetic environment for use in military training and experimentation. VBS2 is developed by Bohemia Interactive Simulations (BISim), and is used by many military organizations worldwide. At FFI we have been using VBS2 since 2008. Figure 3.1 shows some pictures from VBS2.

Figure 3.1 Pictures from VBS2 (Bohemia Interactive Simulations).

VBS2 VTK (Virtual Training Kit), which is the baseline VBS2 product, encompasses the following components:

- VBS2 Desktop Trainer
- VBS2 Development Suite
- VBS2 HLA/DIS Gateway (LVC Game)

VBS2 Desktop Trainer is the main application which includes the game client, the scenario editor and the module for After Action Review (AAR). In the game client the user controls an avatar (virtual character) that can move around and operate as infantry, drive a vehicle or an aircraft, or operate a weapon or a sensor platform. A large collection of vehicles and weapon platforms for land, sea and air is included in VBS2. In addition about 10 terrain models are included in VBS2.

VBS2 Development Suite contains a set of tools that can be used to create new content for VBS2. Examples of new content are vehicles, weapon platforms and avatars, or new terrain models.

[1] VBS2 is based on the first person shooter game Armed Assault.

VBS2 HLA/DIS Gateway makes it possible to connect VBS2 with other military simulations that support HLA (High Level Architecture) or DIS (Distributed Interactive Simulation).

A typical setup for a VBS2 simulation consists of a dedicated VBS2 server, a number of VBS2 game clients and possibly any other simulators connected via HLA/DIS. This is shown schematically in Figure 3.2. At FFI we mainly use VBS2 for experimentation purposes. Often we implement new weapon systems and new technology for testing and demonstration in a synthetic environment.

VBS2 is under continuous development, and new versions are usually released twice a year. We have used VBS2 version 1.5 in our implementation and testing of the simulated AR system. VBS2 version 1.5 was released in September 2011. A more comprehensive description of VBS2 can be found in [8].

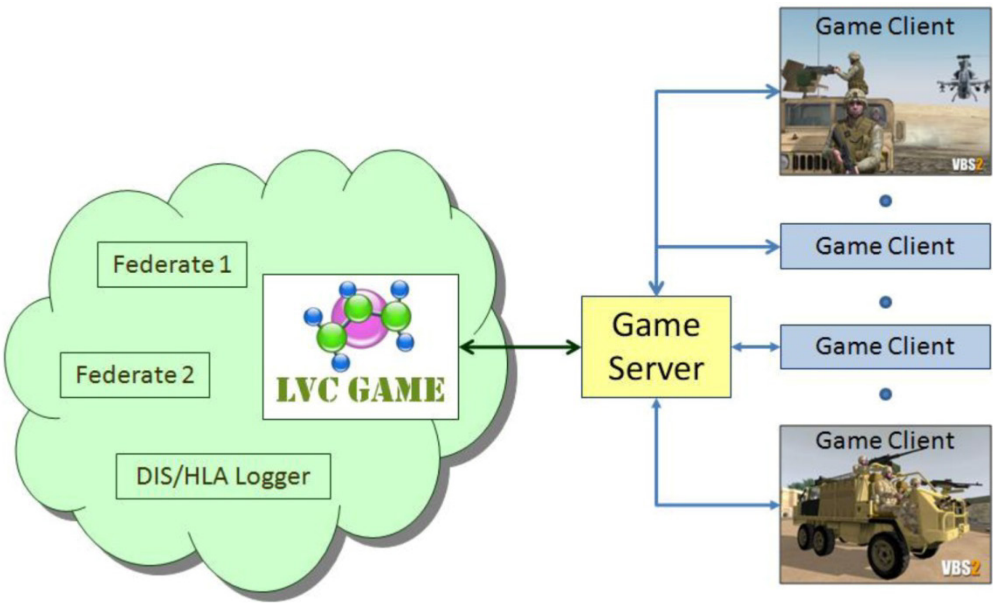


Figure 3.2 Typical setup for a VBS2 simulation (Bohemia Interactive Simulations).

3.1 Scripting in VBS2

VBS2 has a built-in scripting language. A scripting language (or interpreted language) is a programming language that is read and executed by another computer program called an interpreter, rather than being compiled to machine code and executed directly on the processor. An advantage with scripting languages compared to compiled languages is that it is usually faster to make changes to a script, since the script only has to be reread by the interpreter. The major disadvantage is that a script runs much slower than a program that has been compiled to machine code. For instance is VBS2Fusion (version 2.0), which is a C++ based API for VBS2, over 100 times faster in execution than the VBS2 scripting language [8]. VBS2Fusion is described in detail in Section 3.2.

The VBS2 scripting language has a syntax and control structure that is similar to C++ and Java. It contains more than 1 500 scripting commands, and a function library which contains a number of pre-made utility functions. Documentation for the VBS2 scripting language and the function library can be found in the Bohemia Interactive Simulations Wiki [9]. The VBS2 scripting language is intended for programing events in a scenario or additional functionality for vehicles and weapon platforms.

3.2 VBS2Fusion

VBS2Fusion is a C++ based API for VBS2 developed by SimCentric Technologies. VBS2Fusion is not included in VBS2 VTK, but is a VBS2 module that has to be purchased separately. Version 1 of VBS2Fusion provided a C++ interface towards VBS2, but the VBS2 scripting layer was still used to communicate with the VBS2 core. It was first when VBS2Fusion version 2 was released in October 2010 that VBS2 finally got a complete, object oriented, C++ based programming interface with direct access to the VBS2 core. The new version of VBS2Fusion also contains more functionality than what is available through the VBS2 scripting language, for example the ability to draw graphical primitives and text into the VBS2 window. This functionality is used in the implementation of our simulated AR system.

VBS2Fusion consists of a collection of data and utility classes. The data classes are used to store and maintain data fetched from the VBS2 engine, whereas the utility classes are used to access the VBS2 engine directly. The intention of this design concept is to minimize the need for directly accessing the VBS2 engine, to avoid problems with lower performance of the VBS2 program. The components of the VBS2Fusion framework are shown schematically in Figure 3.3. A solution developed with VBS2Fusion is compiled as a plugin Dynamic Link Library (DLL) that is used by the VBS2 engine. More information about VBS2Fusion can be found in the VBS2Fusion Product Manual [10], which is available online.

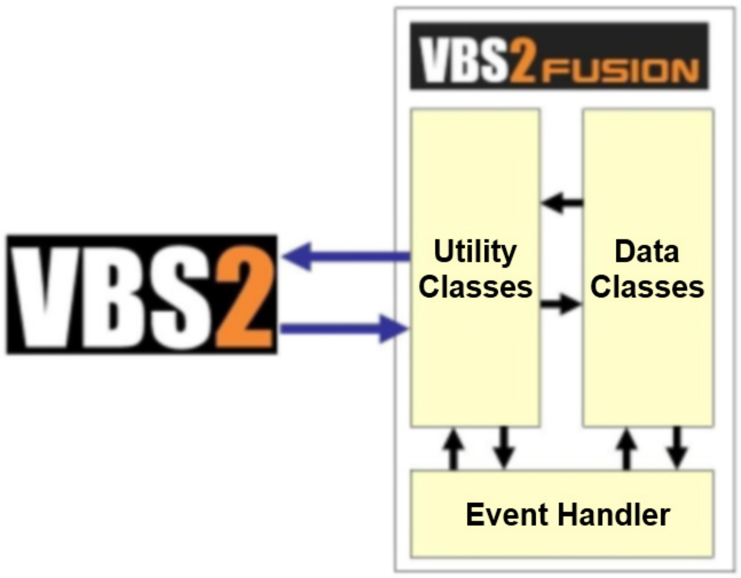


Figure 3.3 The VBS2Fusion framework (Bohemia Interactive Simulations).

4 Functionality in the AR system

Our simulated AR system is developed for use in combat vehicles like Infantry Fighting Vehicles (IFV) and Main Battle Tanks (MBT). It gives the commander, gunner and driver information in the form of graphical symbols in their sights and periscopes. The system works together with an experimental Battlefield Management System (BMS) [11] developed at FFI, and visualizes information like Blue Force Tracking (BFT) and observations received from the BMS.

4.1 Starting the system

Before the AR system can be activated it has to be connected to our experimental BMS. We have added the item "Connect to BMS" to the action menu for the commander of our vehicles in VBS2. This action creates a combo box, where the user can select which BMS client to connect to. This is shown in Figure 4.1. When a vehicle is connected to a BMS client, it sends the following data through an IP (Internet Protocol) connection:

- Vehicle position
- Vehicle direction
- Turret direction
- Laser Range Finder (LRF) direction

Updated data are sent every second. We have developed a VBS2Fusion plugin that sends data from the VBS2 client to the BMS. The connections between the VBS2 clients and the BMS clients are described further in Section 5.1.

After a vehicle has been connected to a BMS client, the AR system can be activated by selecting "Activate AR" from the action menu. In a vehicle the commander, gunner and driver can activate their own AR system independently. The AR system is deactivated by selecting "Deactivate AR" from the action menu, and the commander can terminate the connection to the BMS by selecting "Disconnect from BMS". When the vehicle is connected to a BMS client, a message indicating which BMS client the vehicle is connected to is shown in the lower left corner of the screen. The message "AR activated" is shown at the same place when the AR system is active.

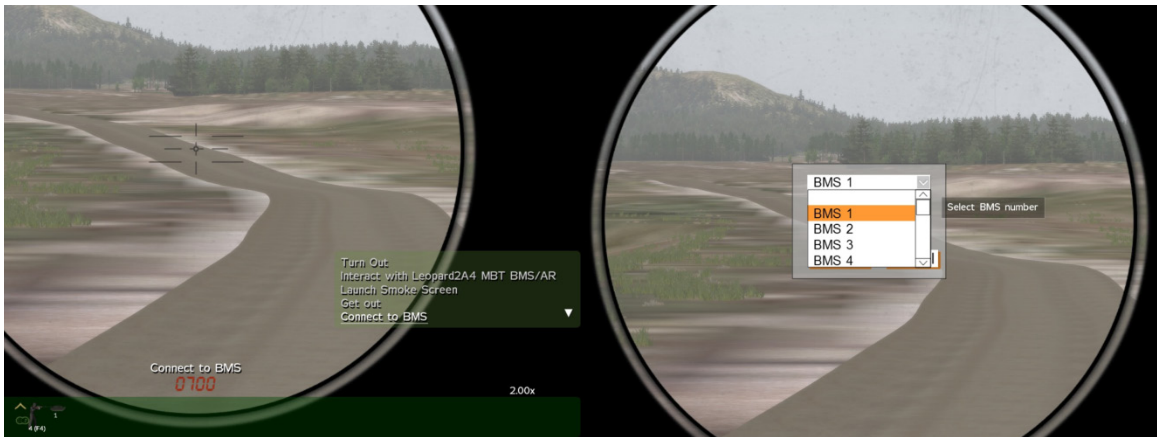


Figure 4.1 Connecting to a BMS client from the commander's view in VBS2.

4.2 AR objects

All the AR objects in the AR system have the same structure, and consist of the following five components:

1. A symbol that shows the affiliation and type of the AR object. We have used symbols from the MIL-STD-2525C standard for military map marking symbols [12]. The AR system receives data about the affiliation and type of an AR object from the BMS. Table 4.1 shows the selection of symbols based on affiliation and type that is available in the AR system. The symbol is always drawn at a specified height in meters above the actual position of the AR object. The default value of this height is 25 meters.
2. A unique text string that represents the ID of the AR object. The ID is drawn above the symbol. The AR system receives this ID from the BMS.
3. A number that shows the distance in meters from the vehicle to the AR object. The distance is drawn on the right side below the symbol.
4. A dot that represents the actual position of the AR object. This dot is drawn with white color if the vehicle has Line of Sight (LOS) to the position of the AR object; otherwise it is drawn with red color.
5. A line that connects the dot and the symbol. The line is drawn with the same color as the symbol, according to the affiliation of the AR object.

Figure 4.2 shows an AR object with its five components. The alpha (transparency) value for the symbol and the line between the dot and the symbol can be adjusted. The default value is 192 on a scale where the value 0 is fully transparent and the value 255 is fully opaque.

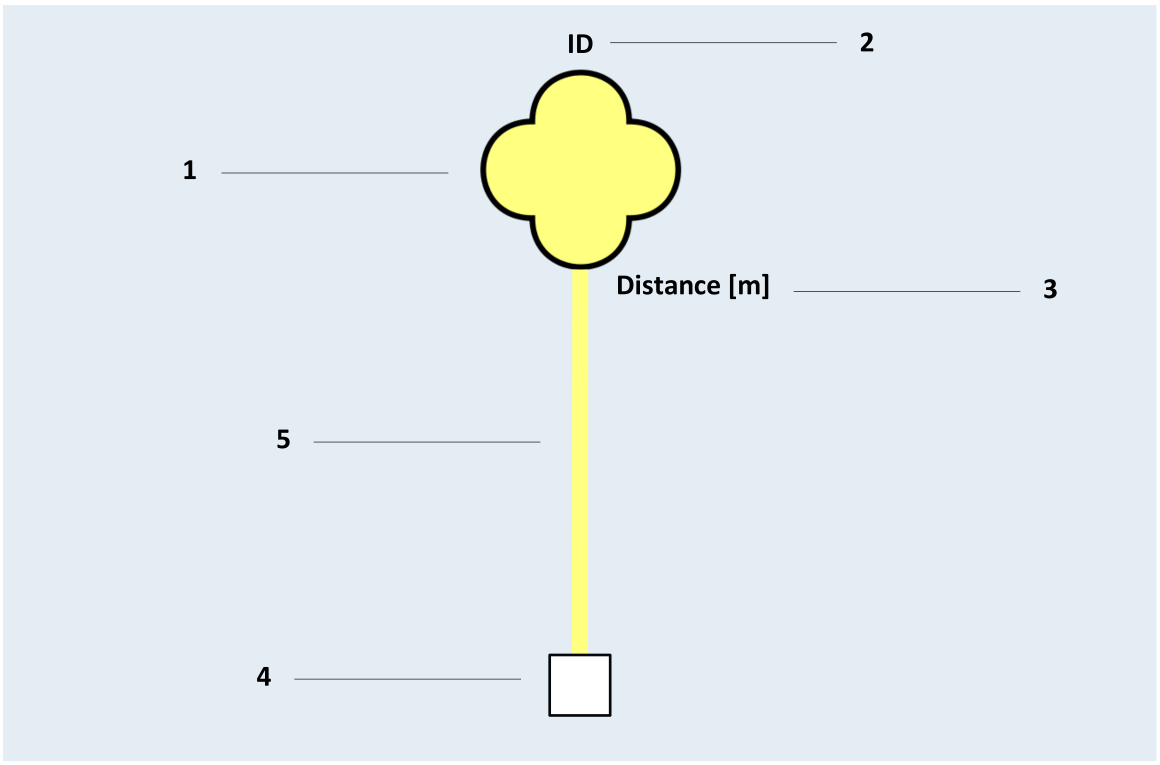


Figure 4.2 The five components of an AR object.

To avoid too much cluttering, the AR system has a minimum distance and a maximum distance for when the AR objects are shown. The default values are 15 meters for the minimum distance, and 6 000 meters for the maximum distance. The default maximum distance reflects the values we are using for the maximum view distance and maximum object draw distance in VBS2.

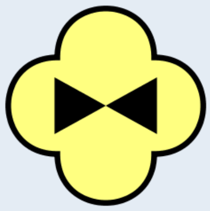
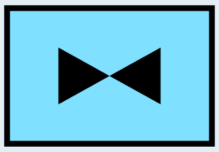
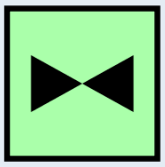
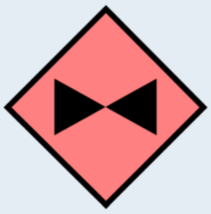
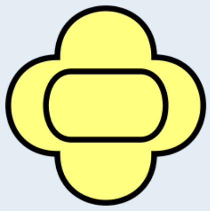
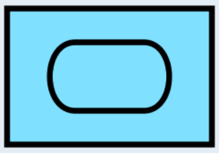
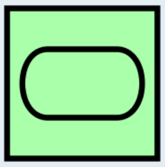
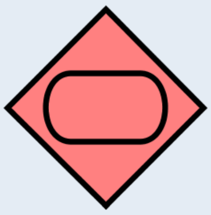
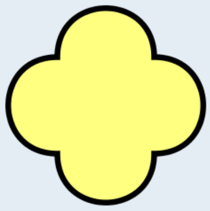
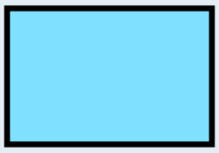
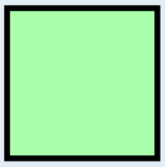
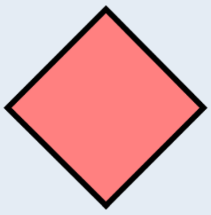
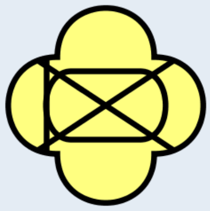
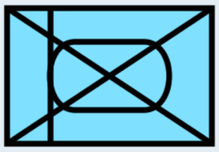
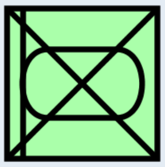
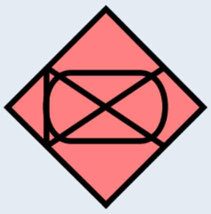
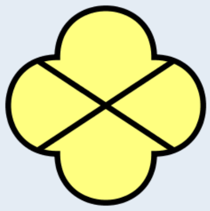
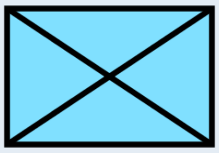
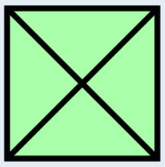
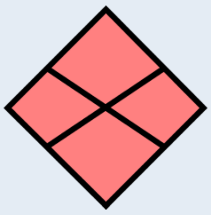
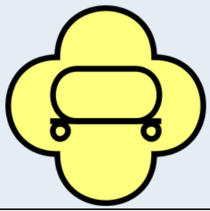
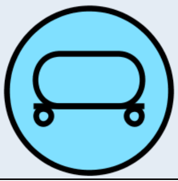
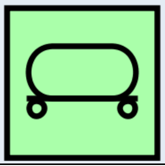
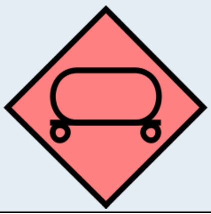
	Unknown	**Friend**	**Neutral**	**Hostile**
Air				
Armor				
Ground				
IFV				
Infantry				
Vehicle				

Table 4.1 The selection of MIL-STD-2525C symbols available in the AR system.

4.3 Blue Force Tracking

The BMS provides Blue Force Tracking (BFT) of all the vehicles that are connected to a BMS client. The BFT data are sent to the AR system where they are visualized with AR objects. Figure 4.3 shows an example with BFT symbols drawn on the BMS screen (to the left), and the virtual scene viewed through the vehicle's sight with AR objects marking the blue forces (to the right). The information is shown from the perspective of the vehicle with ID 1-1, which is looking at two friendly vehicles with IDs 1-2 and 1-3. The first number in the BFT IDs represents the platoon, while the second number represents the unit number within that platoon. On the BMS screen the blue dots marks the position of the vehicles, the short blue lines marks the direction of the vehicles, and the pairs of two long blue lines in a "V"-shape marks the viewing sector of the gunners. In addition there is a short orange line on the vehicle with ID 1-1 that marks the direction of the Laser Range Finder (LRF), which is also the viewing direction of the commander. The BFT information in our system is usually updated once per second, which is probably the best rate one can hope for in a real system today.

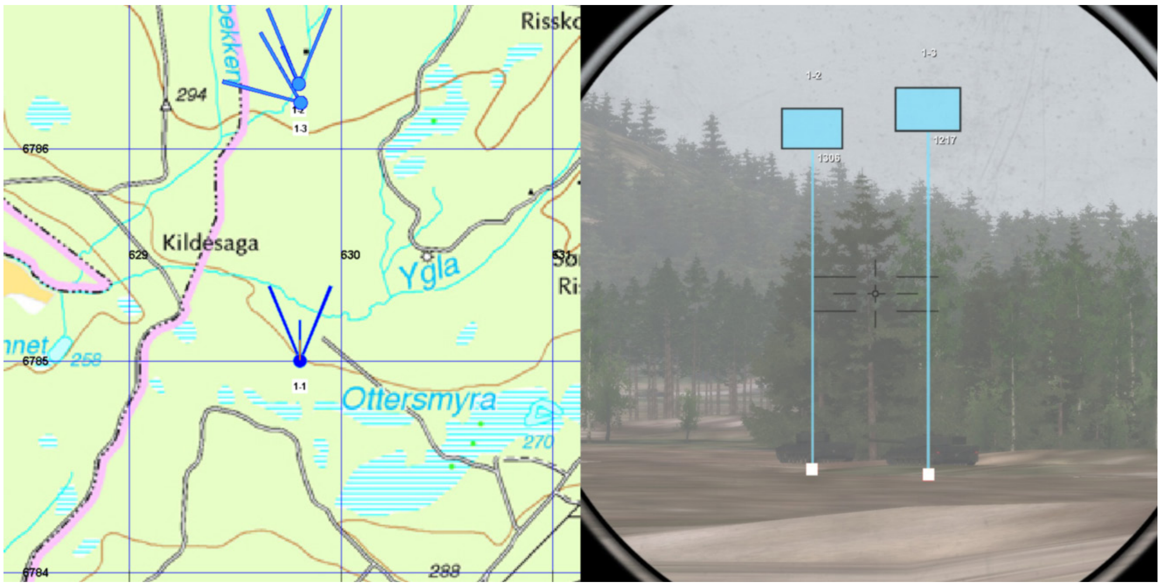


Figure 4.3 Blue Force Tracking on the BMS screen (to the left) and through the vehicle commander's sight with the AR system (to the right).

4.4 Observations

It is possible to mark positions of interest in the BMS by adding observations. Observations can be given an affiliation and a type. The data for the observations are sent to the AR system where they are visualized with AR objects in the same manner as the BFT data. The affiliation and type of the observations can be changed, and the observations can be moved or deleted via the user interface for the BMS. As we will describe in the next section, we have also integrated a Laser Range Finder (LRF) with the BMS that can be used for interaction.

Figure 4.4 shows images from two situations where the BMS screen images with BFT and observations are shown to the left, and the corresponding images from the sight with the AR system are shown to the right. The information is shown from the perspective of the vehicle with ID 1-1. The observations in our experimental BMS have unique IDs consisting of two letters, where the first letter correspond to the creator and the second letter represents the sequence number for that creator. The ID BA thus means that this observation was created by BMS 2 (or BMS B), and it is the first observation created from this BMS.

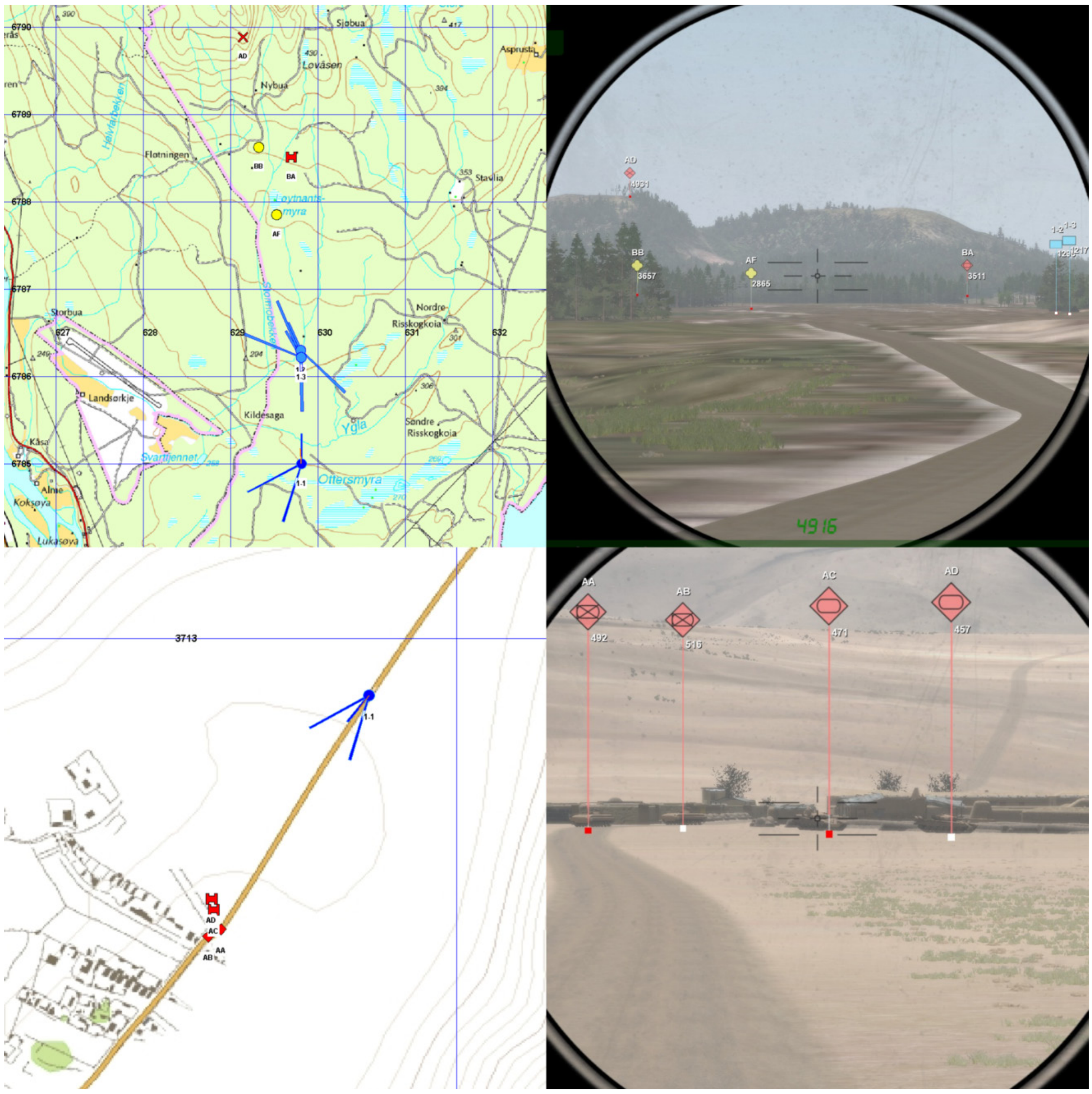


Figure 4.4 Two situations with observations on the BMS screen (to the left) and through the vehicle commander's sight with the AR system (to the right).

4.5 Laser Range Finder

We have also integrated a vehicle mounted Laser Range Finder (LRF) with the BMS. This LRF is operated by the vehicle commander. When the LRF is triggered, the position where it is pointed is sent to the BMS. In a real system this position must be found by using the measured distance and the orientation of the LRF. The position is shown as an orange dot on the BMS screen for a few seconds, and during that period the commander can accept this position as an observation by selecting “Confirm” from the BMS GUI (Graphical User Interface). As default, the observation is added as an unknown ground observation, which afterwards can be changed to the right affiliation and type. Figure 4.5 shows a sequence of four images that illustrates the process of creating an observation with the LRF. Here the car in the commander’s sight is marked as an unknown ground observation.

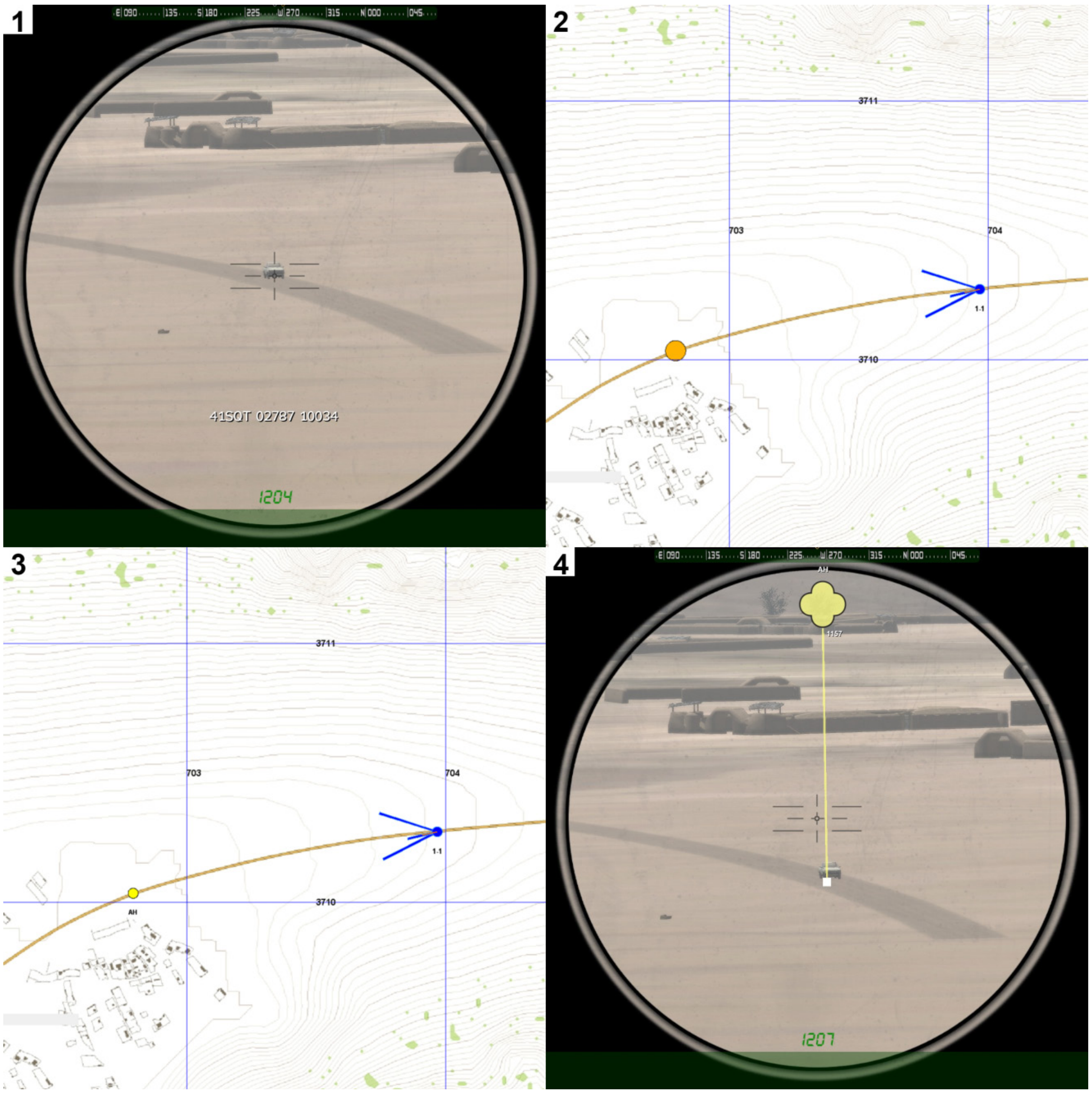


Figure 4.5 A car is marked as an unknown ground observation with the integrated Laser Range Finder (LRF).

4.6 Voice recognition

We have also been experimenting with speech recognition software connected to the BMS, in a way that the commander can say “Confirm!” to have the position from the Laser Range Finder (LRF) accepted as an observation. In addition the commander can specify the affiliation and type through voice commands. For example if the commander points the LRF at an enemy vehicle and says “Confirm enemy vehicle!” an observation with right affiliation and type should be created.

Of course, there could be a lot of problems with integrating a voice recognition system in a noisy environment like a combat vehicle, but it works quite well in simulated vehicles at the lab. The voice recognition system is described in [13].

4.7 Scaling

It is possible to turn on and off whether the AR objects are scaled according to the distance from the vehicle or not. The scaling affects the size of the symbols, the size of the dots, and the width of the lines between the dots and the symbols. The system has a minimum and a maximum scaling factor, to avoid that the symbols become too small or too big. The text parts of the AR objects always have the same size, and the symbols are always at the specified height in meters above the dots that mark the actual positions. Figure 4.6 shows two situations where scaling is turned off in the images to the left, and turned on in the images to the right.

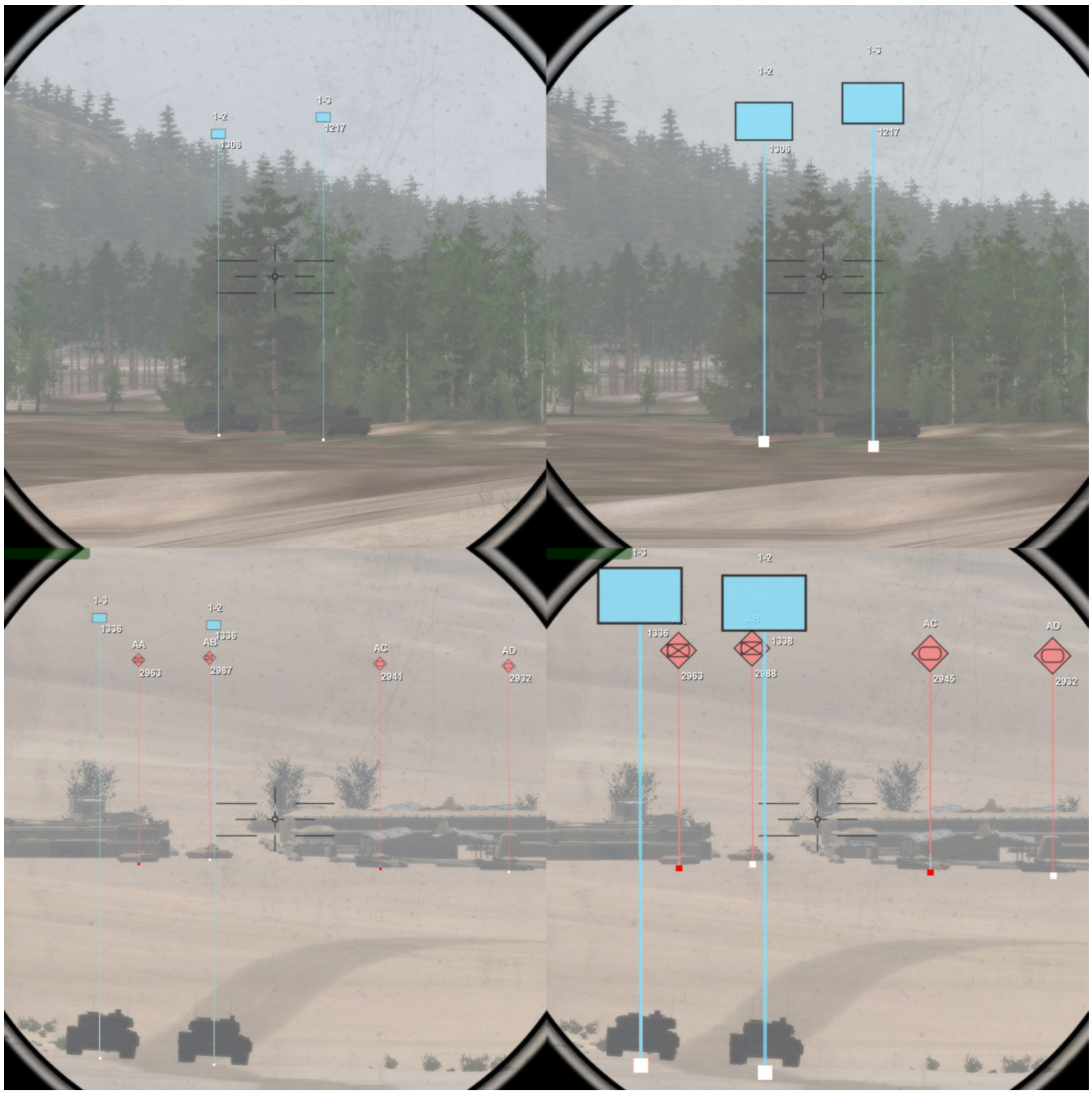


Figure 4.6 AR objects with scaling according to the distance from the vehicle turned off (to the left) and turned on (to the right).

4.8 Occlusion by terrain

It is possible to specify whether the AR objects are to be occluded by the terrain or not. If occlusion is activated and the position of an observation is occluded from the posisiton of the vehicle, the dot is drawn in red color, and the part of the line between the dot and the symbol that is occluded is drawn in white color. This is demonstrated in Figure 4.7, where the two vehicles behind the hill have been marked as enemy ground vehicles. The two vehicles are not visible from the vehicle with the AR system, and this is visualized by the AR objects.

This functionality will be problematic to achieve in a real AR system, and in VBS2 it has the side-effect that AR objects are occluded by buildings and objects as well, since actual Line of Sight (LOS) calculations are being used. Nevertheless, we wanted to be able to experiment with this functionality.

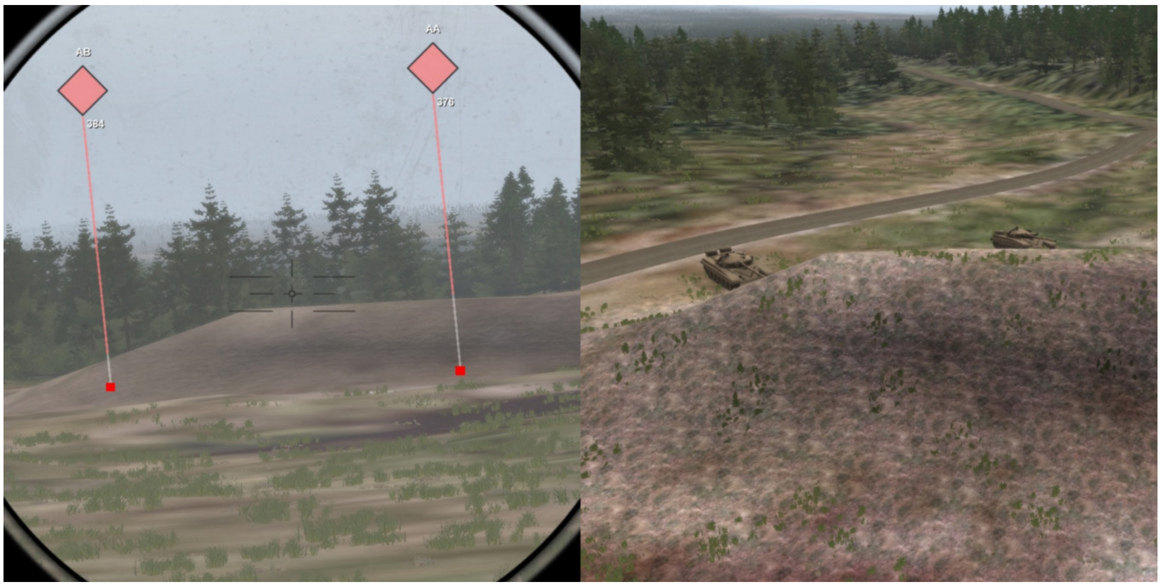


Figure 4.7 AR objects that are positioned on the other side of a hill are visualized with red dots and partially white lines.

4.9 Symbol clamping

The AR system has the possibility to specify whether the symbols and text objects are to be clamped within the upper border of the screen or not. If this functionality is activated the symbols and the text objects will always stay below the upper border of the screen, as long as the position of the AR object itself does not disappear above the upper border of the screen. They will still disappear to the left and right side and to the bottom of the screen though. The effect of this clamping is shown in Figure 4.8, where the image to the left shows AR objects with symbol clamping turned off, and the image to the right shows AR objects with clamped symbols.

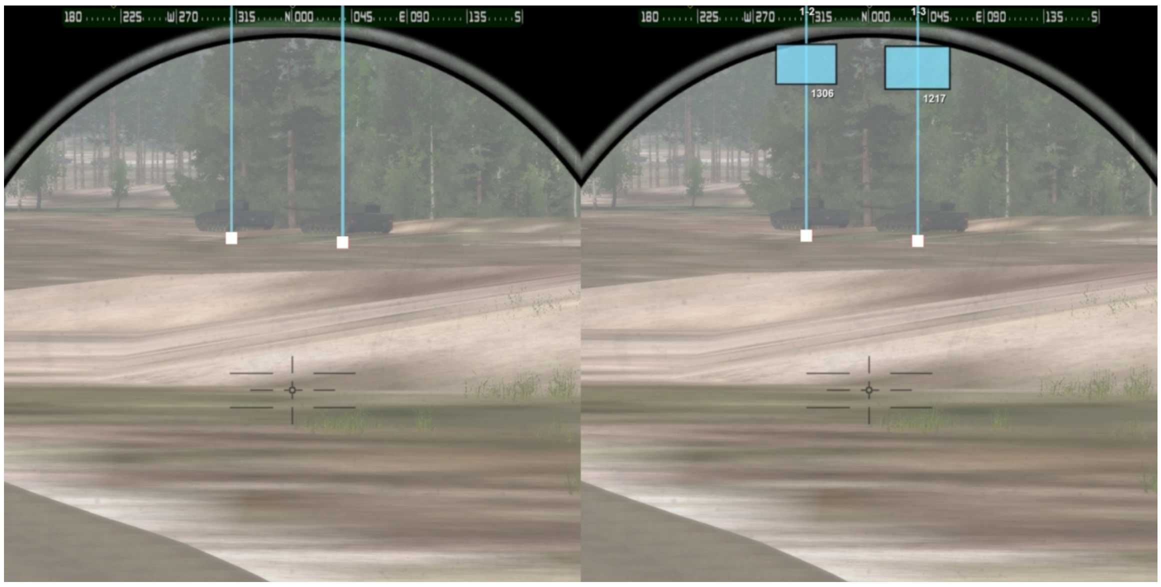


Figure 4.8 AR objects with symbol clamping turned off (to the left), and AR objects with symbol clamping turned on (to the right).

4.10 Simulated system delay

In a real AR system the end-to-end system delay can be defined as the time difference between the moment that the tracking system measures the position and orientation of the viewpoint to the moment when the generated AR graphics corresponding to that position and orientation appear in the displays. End-to-end system delays cause registration errors when motion occurs, and the objects drawn by the AR system are left at their old screen positions during this delay.

We wanted to represent this effect in our simulated AR system to experiment with how long end-to-end system delays that can be accepted by the users. This also makes the simulated AR system appear more realistic in terms of available technology. This delay can be adjusted, and the default value is 0.1 seconds. Figure 4.9 illustrates the visual effect of the end-to-end system delay when the camera is moved to the left. Here the AR system has not yet been updated in the image to the right, and the AR object remains on the same screen position as in the image to the left.

Decreasing the end-to-end system delay is often very expensive, so this is a cost-benefit problem. In this kind of AR system the end-to-end system delay is probably not that critical, as long as it is below a certain threshold.

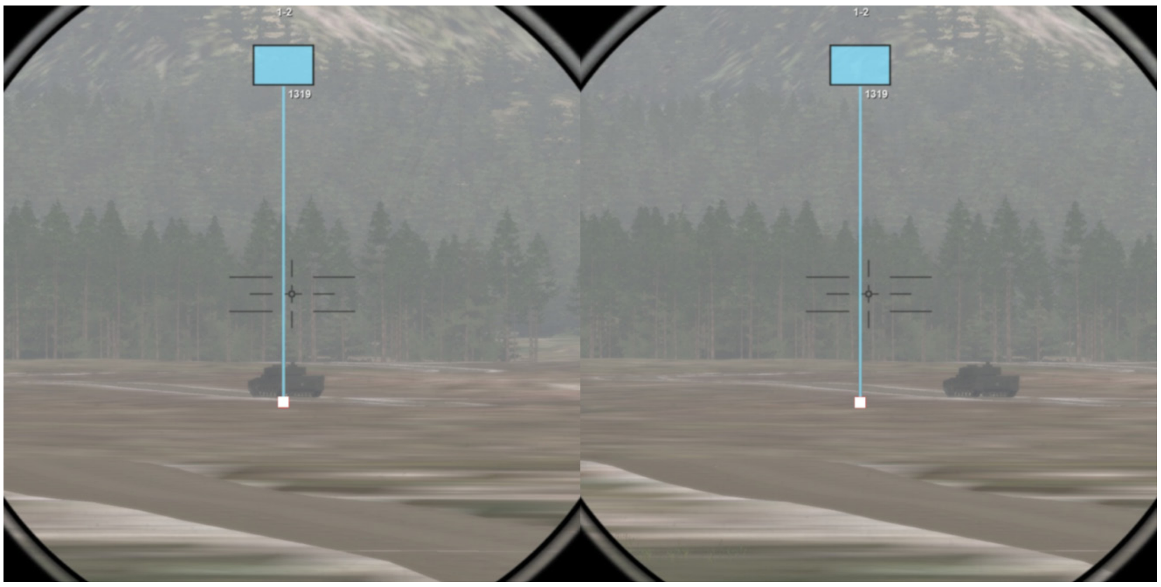


Figure 4.9 The visual effect of the end-to-end system delay.

4.11 Configuration file

When the AR system is activated it reads a configuration file that can be used to define values for a set of parameters for the AR system. The configuration file is called “arconfig.txt”, and can be found in the folder “VBS2_HOME\pluginsfusion”, where VBS2_HOME is the directory where VBS2 has been installed. Table 4.2 shows the parameters that can be set in the configuration file and their default values. The default values are used for all the parameters if the configuration file is missing, otherwise the default parameters are used for any parameters not defined in the configuration file. The values of the parameters can easily be changed by deactivating the AR system, editing the configuration file, and then activating the AR system again.

Parameter	**Default value**	**Description**
ARMinDist	15 [m]	Minimum distance for AR objects to be drawn
ARMaxDist	6 000 [m]	Maximum distance for AR objects to be drawn
bARScaling	True	If true, the AR objects are scaled (Ch. 4.7)
ARMinScaleFactor	0.1	Minimum scale factor for AR objects
ARMaxScaleFactor	1.0	Maximum scale factor for AR objects
ARDefaultScaleFactor	0.1	Scale factor used when scaling is disabled
ARLineHeight	25 [m]	Height of the line between dot and symbol
ARLineWidth	10	Width of the line between dot and symbol
ARFontSize	20	Font size for the text
ARAlpha	192	Alpha value used for the AR objects
bARTerrainOcclusion	True	If true, the AR objects should be occluded (Ch. 4.8)
bARSymbolClamping	True	If true, the symbols and text are clamped (Ch. 4.9)
ARUpdateInterval	0.1 [s]	Simulated end-to-end system delay (Ch. 4.10)

Table 4.2 The parameters that can be changed by the configuration file for the AR system.

5 Implementation of the AR system

The simulated AR system is implemented in C++ using VBS2Fusion version 2.0. VBS2Fusion version 2.0 has functions that can access the Direct3D interface and draw graphical primitives and text into the VBS2 window. Direct3D is part of Microsoft's DirectX API and is used to generate hardware accelerated, three-dimensional computer graphics. The AR system is compiled as a plugin Dynamic Link Library (DLL) that is used by the VBS2 engine.

5.1 System setup

We have implemented the ability for the commander, the gunner and the driver of a vehicle to use the AR system. In our simulated combat vehicle it is the commander that operates the BMS, but both the gunner and the driver have the ability to watch the BMS screen. The operators and components in our simulated combat vehicle are shown in Figure 5.1.

For each vehicle, the VBS2 client used by the commander must be connected to a BMS client for the AR system to work. We have developed a separate BMS communication plugin, also using VBS2Fusion, which sends vehicle data from the VBS2 clients used by the commanders to the BMS clients. The AR system plugins on the VBS2 clients used by the commanders, the gunners and the drivers of the vehicles with AR functionality receives AR object data from the BMS clients. Figure 5.2 shows the connections between the BMS clients and the VBS2 clients in a system with two combat vehicles with AR systems. Here the BMS communication plugins are marked with yellow color, and the AR system plugins are marked with blue color. The simulated system is designed to resemble the architecture of a real combat vehicle system with BMS and AR.

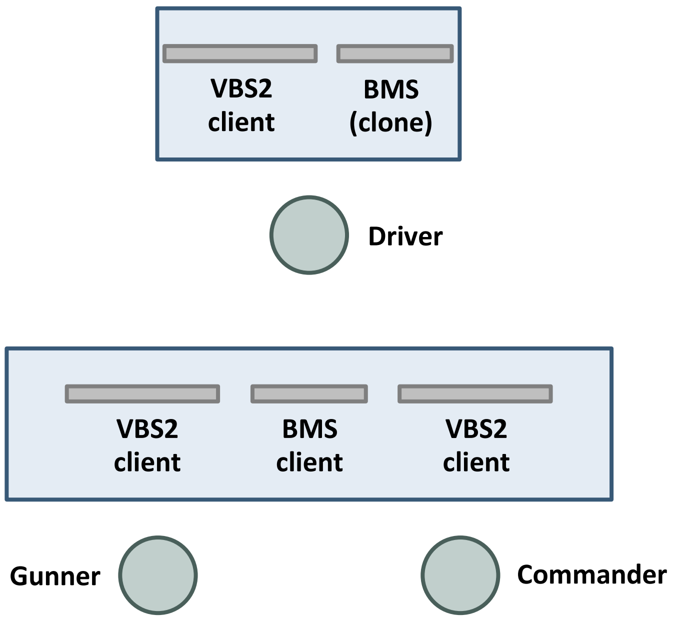


Figure 5.1 Operators and components in our simulated combat vehicle.

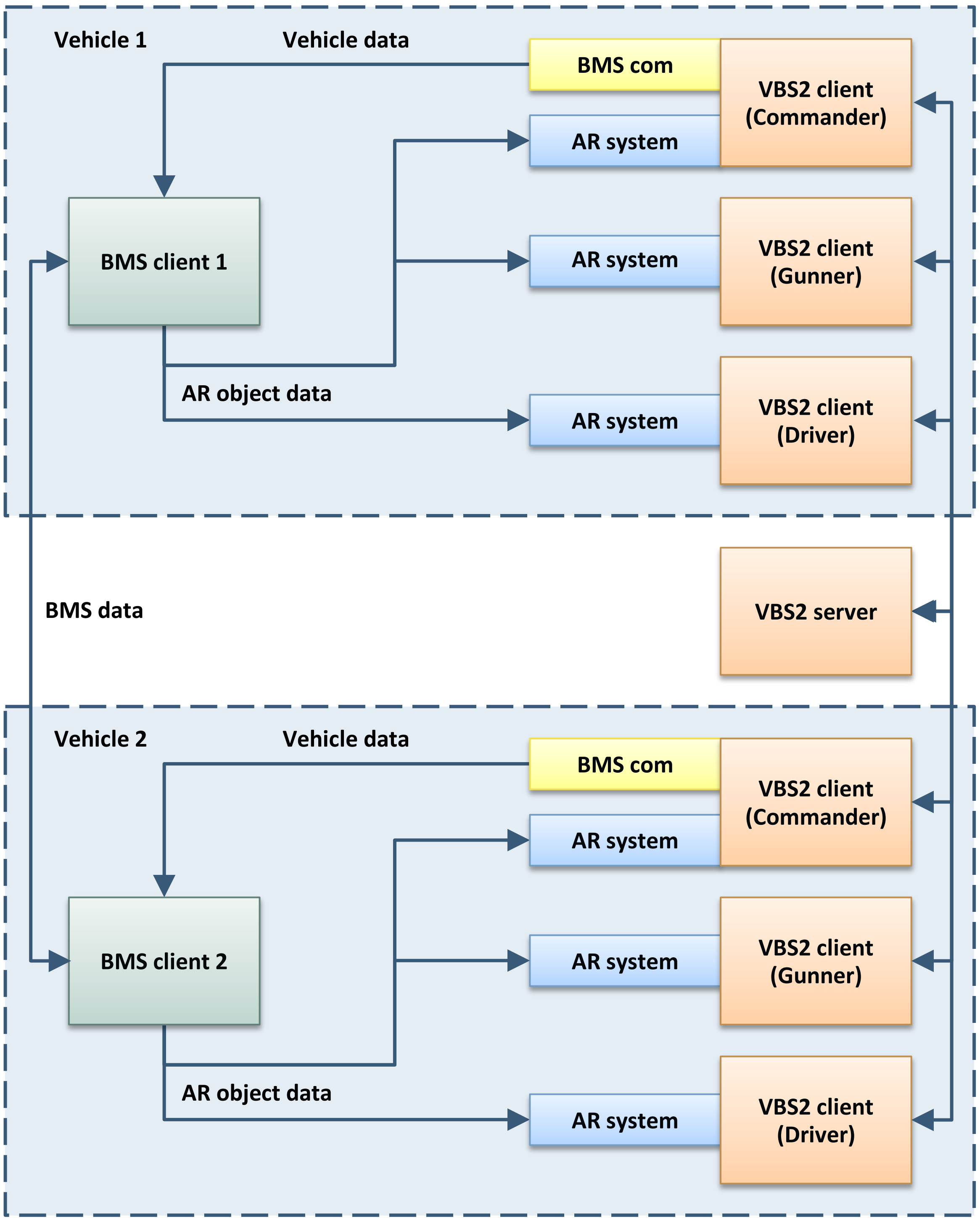


Figure 5.2 Connections between the BMS clients and the VBS2 clients in a system with two combat vehicles with AR systems.

5.2 AR graphics engine

The simulated AR system includes a simple graphics engine that is responsible for drawing the graphical overlay with the AR objects on top of the VBS2 scene. VBS2Fusion has a callback function, OnSimulationStep, which is called by the VBS2 engine in every update loop of VBS2. We use this function to generate the graphics for the AR system in every frame generated by

VBS2. The main structure of the algorithm that generates the graphics for the AR system is outlined in pseudocode in Algorithm 5.1. To simulate the end-to-end system delay the screen coordinates for the AR objects are stored, and only updated at the selected interval (ARUpdateInterval).

The drawing function of the AR graphics engine is outlined in Algorithm 5.2. In this algorithm the AR object scale factor that is used to calculate the size of each AR object when scaling is enabled is calculated according to the formula:

$$\text{AR object scale factor} = \frac{\text{AR line height [pixels]}}{\text{Screen height [pixels]}} \tag{5.1}$$

The AR object scale factor is clamped between the selected values for minimum and maximum scale factor (ARMinScaleFactor and ARMaxScaleFactor respectively).

```
OnSimulationStep() {
    if (ARSystem is activated) {

        if (TimeSinceLastARUpdate > ARUpdateInterval) {
            // Update AR system before drawing.
            ReceiveARObjectDataFromBMS();

            if ((!ARObjectList.Empty) &&
                (Player is in Vehicle) &&
                (Player is turned in)) {
                Sort ARObjectList according to distance;
                Update screen coordinates for the ARObjects;
                DrawARDisplay();
            };
        }
        else {
            // Draw without updating AR system.
            if ((!ARObjectList.Empty) &&
                (Player is in Vehicle) &&
                (Player is turned in)) {
                DrawARDisplay();
            };
        };
    };
};
```

Algorithm 5.1   The main structure of the AR graphics engine.

The drawing functions in the VBS2Fusion API are used for drawing the line and the dot parts of the AR object. For drawing the texture and the text parts, the Direct3D API is used directly due to optimization. At the time of implementation, calling the Direct3D API directly when drawing textures and text was much faster than using the similar functions in the VBS2Fusion API. This issue is further discussed in Section 5.6.

```
DrawARDisplay() {
    // Draw all visible AR objects from back to front.
    for (All objects in ARObjectList) {
        if (ARObject is inside the screen) {
            if ((Distance > ARMinDist) && (Distance < ARMaxDist)) {
                if (bARScaling) {
                    Calculate scale factor; (5.1)
                };

                if (bARSymbolClamping) {
                    Clamp AR symbol to screen height;
                };

                Draw AR line;
                Draw AR dot;
                Draw AR symbol texture;
                Draw AR ID text;
                Draw AR distance text;
            };
        };
    };
};
```

Algorithm 5.2 The main structure of the drawing function from the AR graphics engine.

5.3 BMS communication

Each AR system plugin continuously receives messages from a BMS client. A message from the BMS client is either a Blue Force Tracking (BFT) message or an observation message. BFT messages are sent all the time over UDP (User Datagram Protocol), while the observation messages are sent over TCP (Transmission Control Protocol) when an observation is created, updated or removed. When an AR system plugin connects to a BMS client, it sends a request to receive messages for all current observations, so that it becomes synchronized with the BMS client. The function that receives messages with AR object data from the BMS client is outlined in Algorithm 5.3.

```
ReceiveARObjectDataFromBMS() {
    while (Received new AR object message from BMS) {
        if (Message == Blue Force Tracking message) {
            if (ARObject.ID != Own Vehicle ID) {
                // Ignore AR object for own vehicle.

                if (ARObject with this ID exists in ARObjectList) {
                    Update existing AR object;
                }
                else {
                    Create new ARObject;
                    Insert ARObject in ARObjectList;
                };
            };
        }
        else if (Message == Observation message) {
            if (Message status == NEW) {
                Create new ARObject;
                Insert ARObject in ARObjectList;
            }
            else if (Message status == UPDATE) {
                Update existing AR object;
            }
            else if (Message status == REMOVE) {
                Remove ARObject from ARObjectList;
                Destroy ARObject;
            };
        };
    };
};
```

Algorithm 5.3 The function that receives AR object data from the BMS.

5.4 Terrain occlusion effect

The terrain occlusion effect applied on the line between the AR symbol and the dot is created by testing Line of Sight (LOS) from the vehicle to points along the line. We use a bisection method starting with the end points, to find the lowest visible point on the line within a certain tolerance. This method is outlined in Algorithm 5.4 and illustrated in Figure 5.3. The LOS calculations are done by VBS2Fusion.

```
CheckLOStoARObject() {
    if (!(LOS to lowerPoint on AR line)) {
        // The lower end point on the AR line is not visible.
        lowestVisiblePoint = 0;

        if (LOS to upperPoint on AR line) {
            // The upper end point on the AR line is visible.
            // Find the lowest visible point on the AR line.
            lineHeight = ARLineHeight;

            while (lineHeight > ARTerrainOcclusionTolerance)) {
                if (LOS to middlePoint on line) {
                    upperPoint = middlePoint;
                }
                else {
                    lowerPoint = middlePoint;
                };
                // Update middle point.
                middlePoint = 0.5 * (upperPoint + lowerPoint);

                // Update line height.
                lineHeight = 0.5 * lineHeight;
            };
            // Tolerance reached. Use last middle point as the lowest
            // visible point on the AR line.
            lowestVisiblePoint = middlePoint;
        };
    };
};
```

Algorithm 5.4 The function that checks LOS to the AR object, and finds the lowest visible point on an AR line.

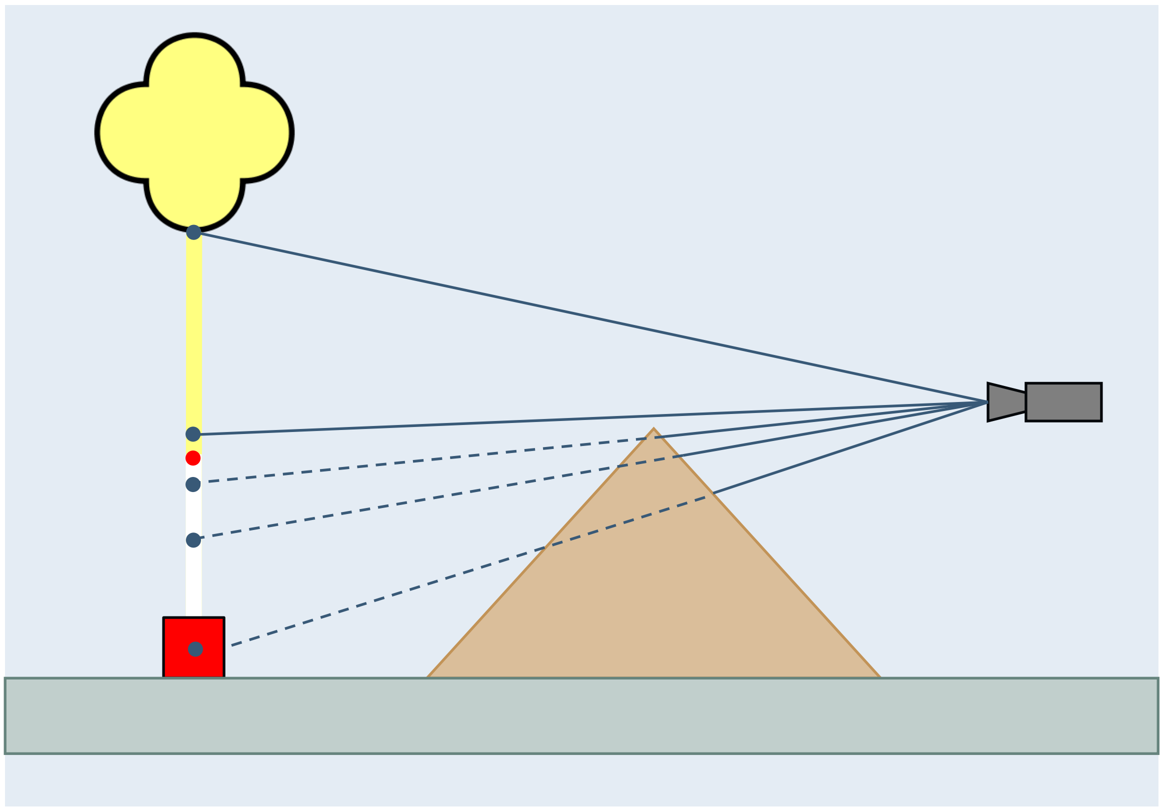


Figure 5.3 Bisection method for the terrain occlusion effect. The red point is the lowest visible point on the AR line found by the method.

5.5 Drawing text with high visibility

The text parts of the AR objects need to be visible on both light and dark background. A simple method for drawing text with high visibility is to draw each character with both black and white color. The black characters are drawn first, and then the white characters are drawn in front of the black characters with a little offset in the left and upward direction. This creates a shadow effect that makes the text readable in front of all background colors. The offset between the black and white characters should be about 5 to 10 percent of the character height. The method is illustrated in Figure 5.4.

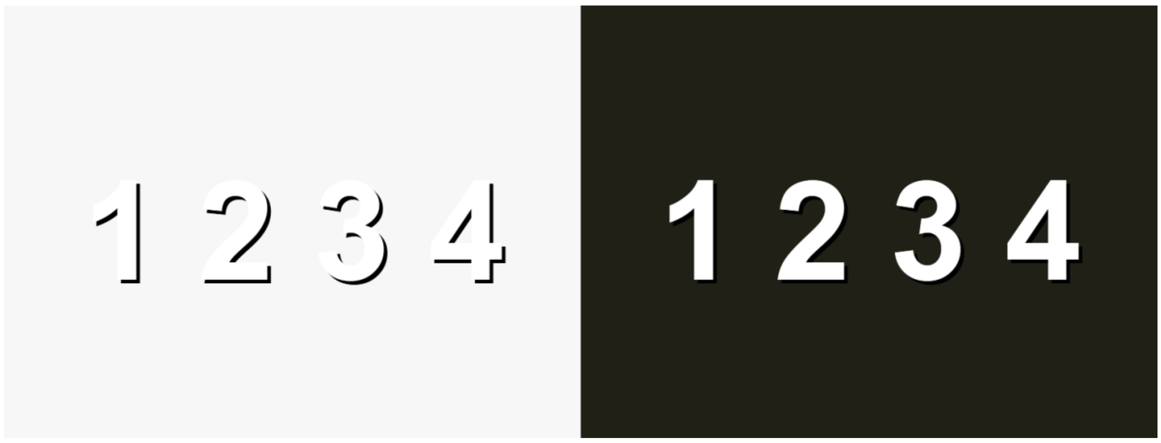


Figure 5.4 Text that is visible on both light and dark background.

5.6 Performance of the system

During the implementation and testing of the AR system we observed performance problems leading to low frame rates when drawing more than about ten AR objects. We found out that this was caused by the functions in the VBS2Fusion API that draw text. This problem disappeared when we used the Direct3D API directly for drawing the text.

To optimize the AR system further we have also used the Direct3D API directly for drawing textures. This has made it possible to cache the textures in memory.

The performance of the final implementation of the AR system is very good, even when the terrain occlusion effect with all the Line of Sight (LOS) calculations is enabled. It can handle more than 50 AR objects visible at the same time without any visible negative effects on the frame rate. Of course, having 50 AR objects visible at the same time will clutter the view, so this is not very practical.

6 Summary

This report has described the functionality in a simulated Augmented Reality (AR) system for VBS2, and how this system has been implemented. The AR system is designed for use in combat vehicles like Infantry Fighting Vehicles (IFV) and Main Battle Tanks (MBT), and gives the commander, gunner and driver information in the form of graphical symbols in their sights and periscopes. It works together with an experimental Battlefield Management System (BMS), and visualizes information like Blue Force Tracking (BFT) and observations received from the BMS. A vehicle mounted Laser Range Finder (LRF) has been integrated with the BMS to be able to select positions from the terrain.

We have used the C++ based application programming interface (API) for VBS2, VBS2Fusion, to implement the AR system. VBS2Fusion has functionality that makes it possible to draw graphical primitives and text into the VBS2 window. The simulated AR system has a simple graphics engine that is responsible for drawing the graphical overlay with the AR objects on top of the VBS2 scene.

The AR system was used in a large simulator experiment in FFIs Battle Lab in November 2011, where six professional Infantry Fighting Vehicle (IFV) crews from the Telemark Battalion participated for one week. During this week the vehicle crews played through a number of relevant scenarios for combat vehicles in VBS2, to evaluate the usefulness of AR functionality in combination with the BMS. The results from this experiment will be used as input to the ongoing process of designing a real AR system for combat vehicles.

References

[1] M. Halsør, “Simuleringsøvelse med VBS2 - Test av BMS og AR”, FFI-rapport 2012/00612, 2012.

[2] M. Halsør, “Praktisk gjennomføring av simulator-eksperiment på FFIs battle-lab”, FFI-notat 2012/00613, 2012.

[3] M. Halsør, S. Martinussen, P. I. Evensen and B. Hugsted, “Uttesting av BMS i syntetisk miljø”, FFI-rapport 2007/00139, 2007.

[4] M. Halsør and S. Martinussen, “Simulatorøvelse for evaluering av BMS, MUAS og AR”, FFI-rapport 2009/01379, Restricted, 2009.

[5] R. T. Azuma, *A Survey of Augmented Reality*, Presence: Teleoperators and Virtual Environments, Vol. 6, 1997.

[6] R. Azume, Y. Baillot, R. Behringer, S. Feiner, S. Julier, B. MacIntyre, *Recent Advances in Augmented Reality*, IEEE Computer Graphics and Applications, Vol. 21, No. 6, 2001.

[7] D. Donovan and J. Cimino, *Augmented Reality as an Emerging Military Training Technology*, I/ITSEC 2010 Paper No. 10305, 2010.

[8] Bohemia Interactive Australia Pty Ltd, “White Paper: VBS2”, Release Version 2.0, 2012, http://armory.bisimulations.com/sites/default/files/file_uploads/VBS2_Whitepaper.pdf.

[9] BISim Wiki, http://resources.bisimulations.com/wiki.

[10] VBS2Fusion Product Manual, http://www.simct.com/online/vbs2fusion_manual.

[11] M. Halsør and B. Hugsted, “Teknisk beskrivelse av eksperiment-BMS”, FFI-rapport 2009/01402, 2009.

[12] United States Department of Defense, “MIL-STD-2525C, Department of Defense Interface Standard: Common Warfighting Symbology”, 2008.

[13] O. M. Strand, D. Olson, B. Hugsted and M. Halsør, “Design av et talegjenkjenningssystem for håndfri input til et BMS med AR-informasjon i siktebildet”, FFI-rapport 2012/00611, 2012.

Abbreviations

AAR	After Action Review
API	Application Programming Interface
AR	Augmented Reality
ASI	Application Scripting Interface
BFT	Blue Force Tracking
BISim	Bohemia Interactive Simulations
BMS	Battlefield Management System
CGF	Computer Generated Forces
DIS	Distributed Interactive Simulation
DLL	Dynamic Link Library
FOV	Field of View
GUI	Graphical User Interface
HDM	Head-Mounted Display
HLA	High-Level Architecture
HMS	Helmet-Mounted Sights
HUD	Head-Up Displays
IFV	Infantry Fighting Vehicle
IMU	Inertial Measurement Unit
IP	Internet Protocol
LOS	Line of Sight
LRF	Laser Range Finder
LVC	Live Virtual Constructive
MBT	Main Battle Tank
TCP	Transmission Control Protocol
TMBN	Telemark Battalion
UDP	User Datagram Protocol
VBS2	Virtual Battlespace 2