FFI-RAPPORT

17/00879

Modelling and implementation of a generic active protection system for entities in Virtual Battlespace (VBS)

–

Per-Idar Evensen

Modelling and implementation of a generic active protection system for entities in Virtual Battlespace (VBS)

Per-Idar Evensen

Norwegian Defence Research Establishment (FFI) 6 April 2017

Keywords
Modellering og simulering
Aktive beskyttelsessystemer
Simuleringsmodeller
Simuleringsverktøy
Virtual Battlespace (VBS)

FFI-rapport
FFI-RAPPORT 17/00879

Prosjektnummer
1401

ISBN
P: 978-82-464-2898-7
E: 978-82-464-2899-4

Approved by
Halvor Ajer, *Research Manager*
Jon E. Skjervold, *Director*

Summary

Active protection systems (APSs) for combat vehicles have been under development for several decades. During the last few years this technology has matured, and several systems are currently being fielded. Examples of such systems are the Russian-made Arena, Raytheon's Quick Kill, as well as Iron Fist and Trophy used by the Israel Defence Forces (IDF).

In order to equip entities in our combat simulations with active protection systems, we have implemented a model of a generic hard-kill APS for entities in the simulation tool Virtual Battlespace (VBS). An APS using hard-kill measures generally means that the incoming projectiles are physically intercepted and destroyed or degraded.

The modelled APS performs the same sequence of actions as a real APS. It detects, classifies and tracks, and intercepts incoming projectiles. Our generic APS model can easily be configured and calibrated to simulate most existing, and possible near-future, hard-kill APSs.

The APS model has been implemented using the VBS scripting language. It can be used by both virtual and constructive entities in VBS. For virtual entities a graphical user interface (GUI) showing the status of the APS has been designed and implemented. This GUI has been implemented in VBSFusion, which is a C++-based application programming interface (API) for VBS.

Sammendrag

Aktive beskyttelsessystemer (Active protection systems (APSs)) for stridskjøretøy har vært under utvikling i flere tiår. I løpet av de siste årene har denne teknologien blitt mer moden, og flere systemer er nå i operativ bruk. Eksempler på slike systemer er russiske Arena, Quick Kill fra Raytheon, samt Iron Fist og Trophy som er i bruk i det israelske forsvaret.

For å kunne utstyre entitetene i våre stridssimuleringer med aktive beskyttelsessystemer, har vi implementert en modell av et generisk, “hard-kill” APS for entiteter i simuleringsverktøyet Virtual Battlespace (VBS). At et aktivt beskyttelsessystem anvender motmidler klassifisert som “hard-kill”, betyr at de innkomne prosjektilene fysisk avskjæres og ødelegges eller degraderes.

Vår APS-modell utfører den samme sekvensen av handlinger som et reelt APS. Den detekterer, klassifiserer og tracker, og avskjærer innkomne prosjektiler. Vår generiske modell kan enkelt konfigureres og kalibreres til å simulere de fleste eksisterende, og mulige framtidige, “hard-kill” aktive beskyttelsessystemer.

APS-modellen har blitt implementert i det innebygde skriptspråket i VBS (VBS scripting language). Den kan brukes av både “virtual” og “constructive” entiteter i VBS. For “virtual” entiteter har vi utviklet et grafisk brukergrensesnitt som viser statusen til det aktive beskyttelsessystemet. Det grafiske brukergrensesnittet har blitt implementert i VBSFusion, som er et C++-basert programmeringsgrensesnitt for VBS.

Content

1 Introduction

Active protection systems (APSs) for combat vehicles have been under development for several decades. During the last few years this technology has matured, and several systems are currently being fielded. Examples of active protection systems are the Russian-made Arena, Raytheon's Quick Kill, as well as Iron Fist and Trophy used by the Israel Defence Forces (IDF).

In order to equip entities in our combat simulations [1] with active protection systems, we have implemented a model of a generic *hard-kill* APS for entities in the simulation tool Virtual Battlespace (VBS). An APS using hard-kill measures generally means that the incoming projectiles are physically intercepted and destroyed or degraded. The generic APS model can easily be configured and calibrated to simulate most existing, and possible near-future, hard-kill APSs.

The APS model has been implemented using the VBS scripting language. It can be used by both *virtual* and *constructive* entities in VBS. For virtual entities a graphical user interface (GUI) showing the status of the APS has been designed and implemented. This GUI has been implemented in VBSFusion, which is a C++-based application programming interface (API) for VBS.

In 2006 we implemented a simple model of an APS for entities in an in-house developed combat vehicle simulator called NORBASE (Norwegian Battle Simulator Experiment). NORBASE was based on the commercial game Unreal Tournament 2004 and was used in an experiment in November 2006 [2]. The APS model in NORBASE was implemented by overriding the damage calculations and simply not assigning damage to the vehicle if it was hit in a sector covered by the APS.

The APS model we have implemented for entities in VBS has a higher fidelity and behaves more like a real APS. It detects incoming projectiles and attempts to physically stop the projectiles from hitting the vehicle.

This report has been organized as follows. In Chapter 2 a short introduction to hard-kill APSs is given. Next, in Chapter 3, the simulation tool Virtual Battlespace (VBS) is briefly described. Chapter 4 describes how the APS model has been implemented, and Chapter 5 describes the GUI. Finally, a summary and conclusion can be found in Chapter 6.

The work with implementing the APS model has mainly been done in FFI-project 1401 “Combat systems – manoeuvre II” and FFI-project 1353 “Combat Effectiveness in Land Operations II”.

The APS model described in this report was implemented and tested in version 3.9.2 of VBS and version 3.9.2 of VBSFusion. This report describes how the APS model was implemented at the time this report was published.

2 Active protection systems

Active protection systems (APSs) for combat vehicles have been under development since the 1980's. Such systems are designed to actively prevent incoming projectiles from hitting their target.

2.1 Soft-kill and hard-kill APSs

APSs are usually classified into *soft-kill* and *hard-kill* systems.

- *Soft-kill* systems use different kinds of countermeasures to alter the electromagnetic or acoustic signature of the target and thereby disturb the sensors of the incoming threat so that it misses the target. This can typically be achieved by reducing the signature of the target, or by enhancing the signature of the background in such a way that the target becomes less distinct. Examples of soft-kill countermeasures are jamming, smoke-screens, and decoy flares.

- *Hard-kill* systems physically intercept and degrade or destroy the incoming threat. Examples of hard-kill countermeasures are blasts with or without fragments, small missiles, and multiple explosively formed projectiles (EFPs). Figure 2.1 shows the hard-kill active protection system Quick Kill from Raytheon intercepting and destroying an incoming projectile.

As mentioned in Chapter 1, this report describes the modelling and implementation of a generic hard-kill APS.

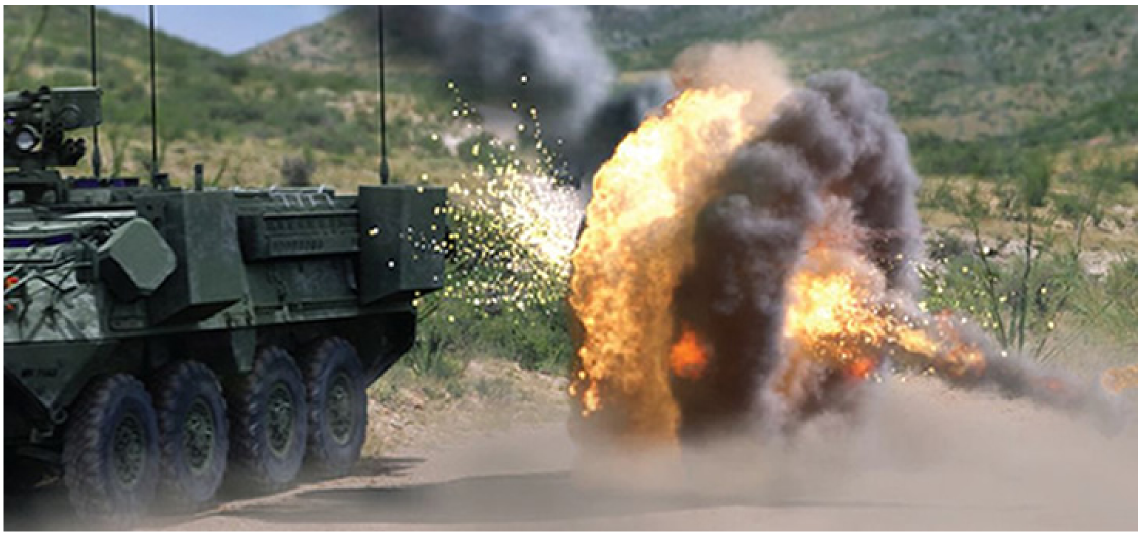


Figure 2.1 The hard-kill active protection system Quick Kill from Raytheon intercepting and destroying an incoming projectile (Raytheon).

2.2 Neutralizing incoming projectiles

To neutralize an incoming projectile, an APS needs to perform the following sequence of actions:

1. *Detect* the incoming projectile.

2. *Classify* and *track* the incoming projectile and determine if it is necessary to activate countermeasures.

3. If necessary, *activate countermeasures* (soft-kill or hard-kill).

For a hard-kill APS activating countermeasures means physically *intercepting and destroying* the incoming projectile. Figure 2.2 illustrates the sequence of actions performed by a hard-kill system.

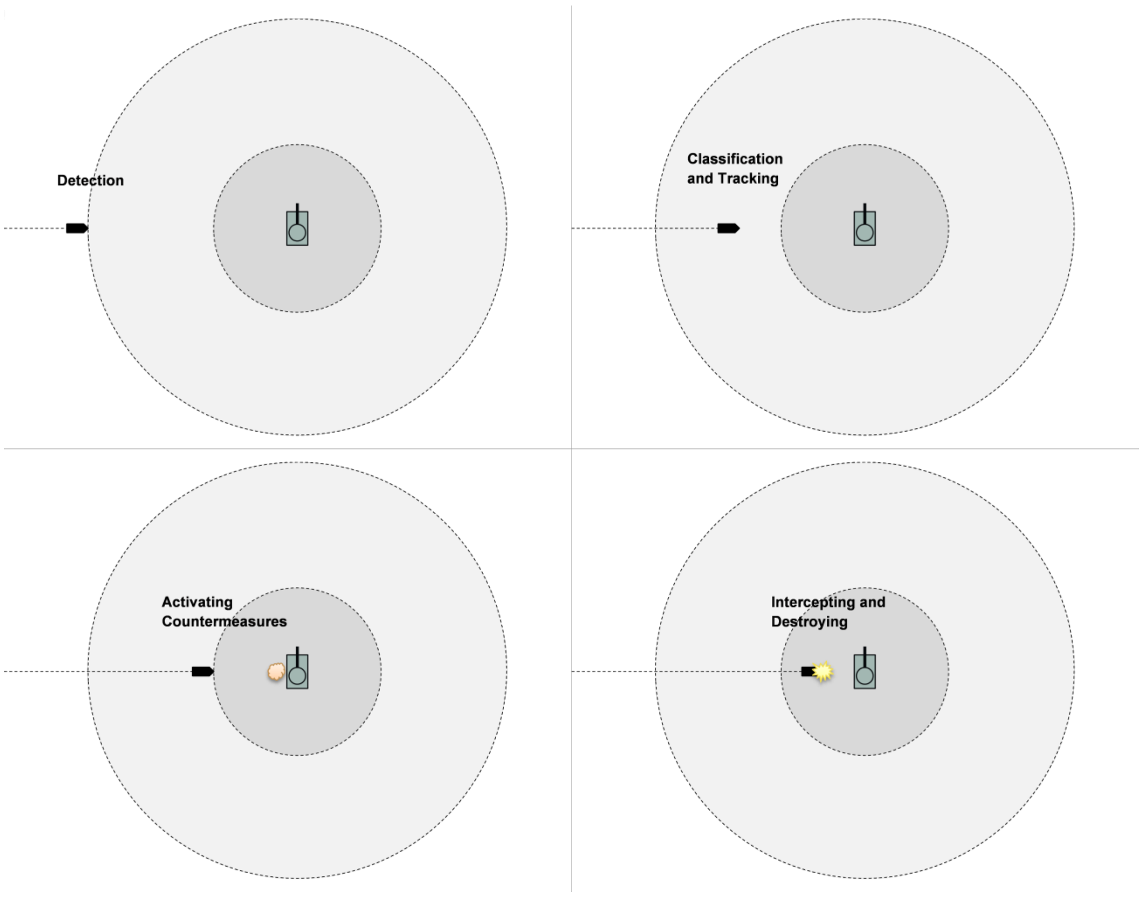

Detection
Classification and Tracking
Activating Countermeasures
Intercepting and Destroying

Figure 2.2 The sequence of actions performed by a hard-kill APS to neutralize an incoming projectile.

A hard-kill APS will thus typically include the following main components:

- A sensor system for detecting, classifying, and tracking the incoming projectile. The sensor system will usually include one or more of the following sensor types: radars, optical sensor, and lidars.

- A processing system for calculating whether the incoming projectile is a threat, and if necessary, how to intercept the projectile.

- A system for releasing physical countermeasures for intercepting and destroying the incoming projectile.

One critical characteristic of a hard-kill APS is how much time it takes from the projectile is detected until it can be neutralized. This property is decisive regarding what types of projectiles the APS is effective against. While modern anti-tank missiles typically have velocities between 100 and 400 m/s, modern kinetic energy penetrators (KEPs) typically have velocities between 1,400 and 1,900 m/s. To be effective against KEPs, an APS will only have a few milliseconds from detection to the projectile have to be neutralized. There are currently no fielded APSs that are effective against KEPs, but such systems are expected to be available in the future.

Many active protection systems consist of a number of modules mounted around the exterior of the vehicle. Each of these modules will typically cover a sector of a circle around the vehicle. An APS module may also contain more than one set of countermeasures, and thus provide redundant protection. Figure 2.3 shows an example of an APS consisting of eight modules, each protecting its own sector around the vehicle. Other APSs may have only one turret-like launcher for releasing countermeasures mounted on top of the vehicle. Systems like this will typically have 360-degree coverage around the vehicle, and typically also contain a larger number of countermeasures.

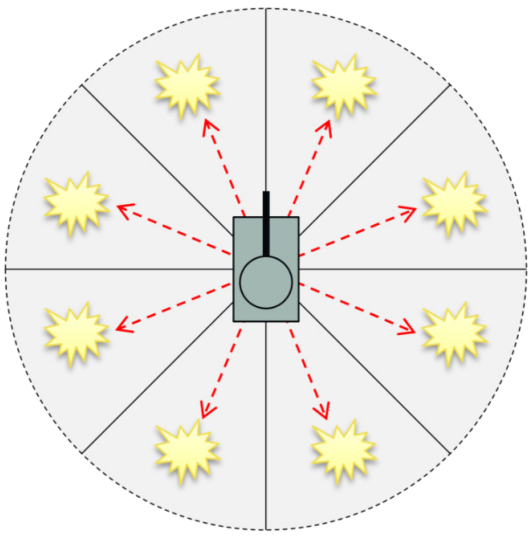


Figure 2.3 Modular APS with eight protected sectors.

3 Virtual Battlespace (VBS)

Virtual Battlespace (VBS) [3] is an interactive, three-dimensional synthetic environment, for use in military training and experimentation. VBS is developed by Bohemia Interactive Simulations (BISim) and is based on game technology from the Armed Assault (ARMA) series.

VBS is used by many military organizations worldwide, including the Norwegian Defence, and has become an industry standard in game-based military simulation. At FFI VBS has been in use since 2008, mainly for conducting virtual simulation experiments for analysis purposes [4][5].

VBS is delivered with a comprehensive content library funded by different nations over the years. VBS version 3.9.2 includes models of 1,965 human characters, 2,285 land, sea, and air vehicles, 4,955 objects, and 835 weapons and weapon platforms. Figure 3.1 shows images from VBS.

VBS has its own scripting language for creating new functionality. In addition there is a C++-based API called VBSFusion for developing plug-ins[1] for VBS.

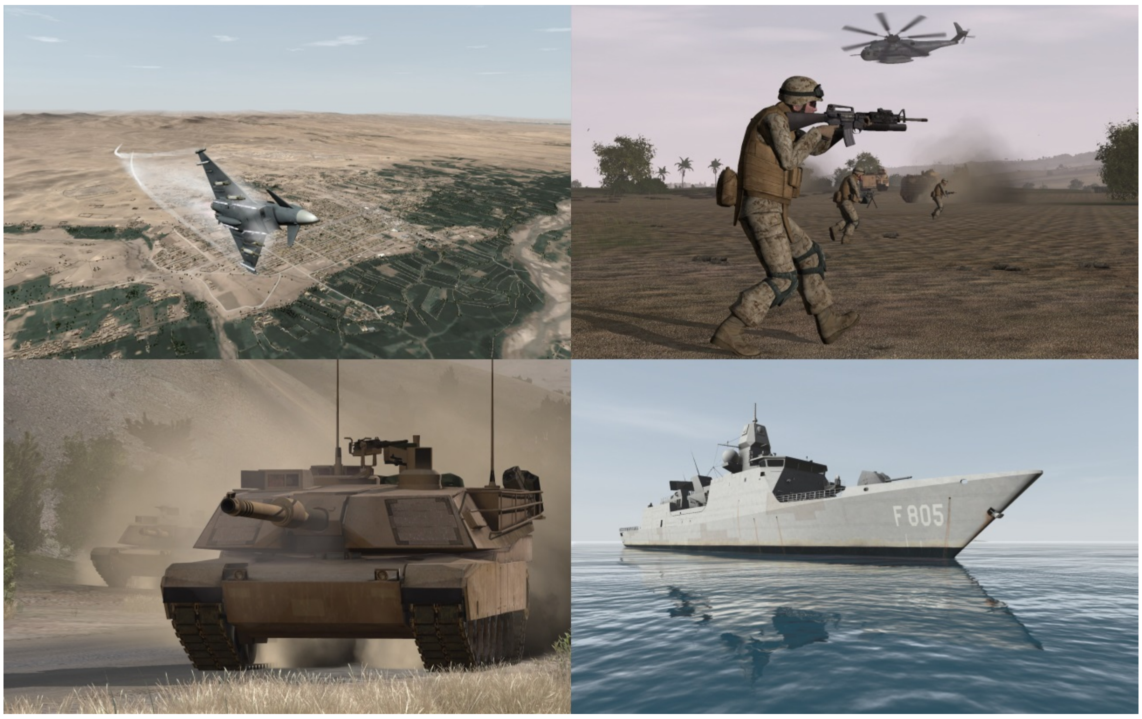


Figure 3.1 Images from VBS (Bohemia Interactive Simulations).

[1] A plug-in is a software component which adds a specific feature to an existing application.

3.1 Distributed simulation with VBS

VBS is a simulation tool for conducting human-in-the-loop (HITL) simulations. A typical setup for a VBS simulation consists of a dedicated VBS server and a number of VBS clients operated by human players. VBS is mainly used for virtual simulation, but it can also be used for constructive simulation with semi-automated forces (SAFs). Furthermore, it is possible to connect a VBS simulation to other DIS (Distributed Interactive Simulation) or HLA (High Level Architecture) compliant simulation components via VBS Gateway [6]. Figure 3.2 illustrates a typical setup for a VBS simulation.

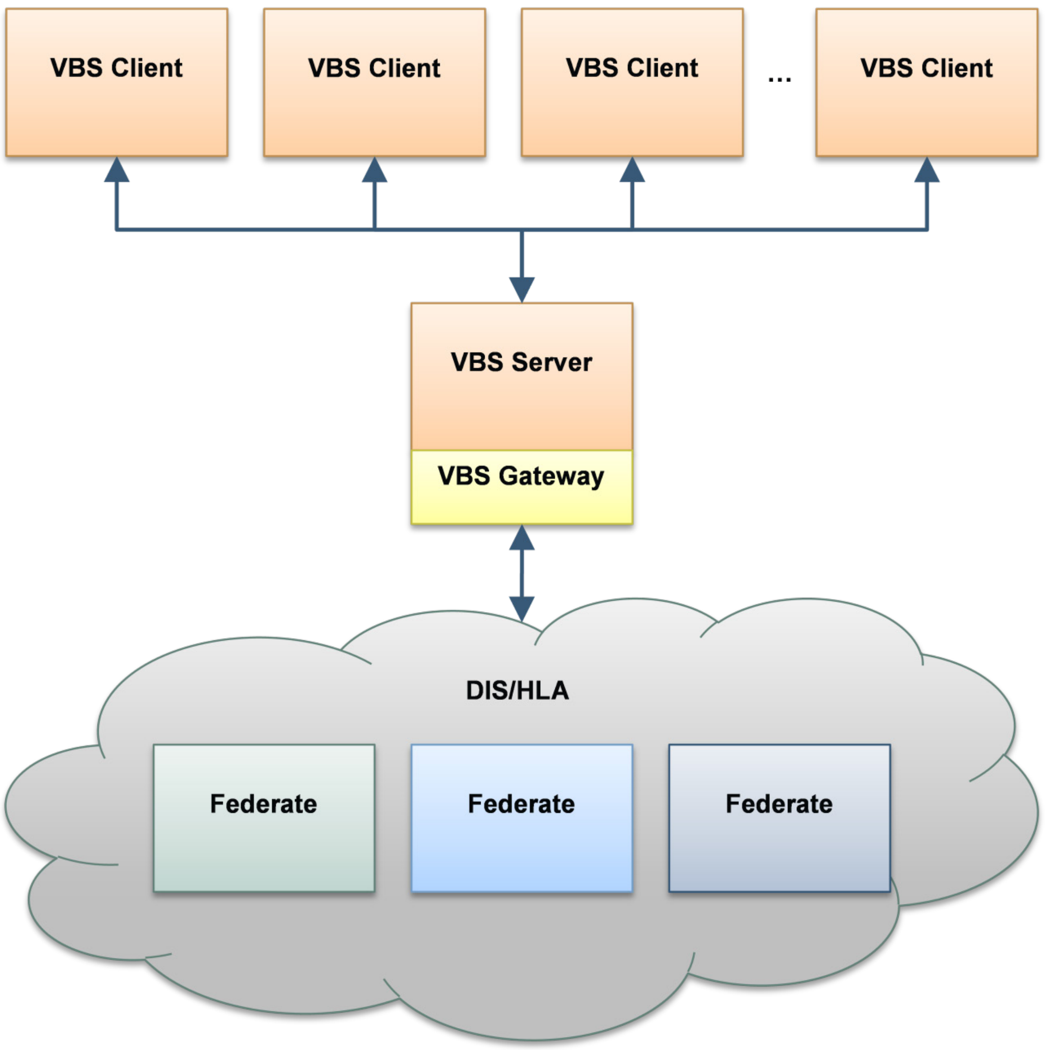


Figure 3.2 A typical setup for a VBS simulation.

Entities in a VBS simulation are simulated on one specific computer, which sends updates about this entity to all the other computers in the VBS simulation. In VBS an entity or an object is said to be *local* to the computer where it is simulated. The basic rules for where entities, objects and projectiles are simulated in a VBS simulation are as follows [7]:

- An avatar (virtual character) is always simulated on the client where the avatar is operated.
- A vehicle is always simulated on the client operated by its driver.
- Artificial intelligence (AI) controlled entities are simulated on the client operated by their leader, if the leader is controlled by a human operator. AI entities without leaders controlled by human operators are simulated on the server.
- AI entities created after mission start will be simulated on the computer where they were created.
- Objects and empty vehicles are simulated on the server.
- Objects and empty vehicles created after mission start are simulated on the computer where they were created.
- Unguided bullets, shells, and rockets are simulated locally on each computer, but the instance of the projectile simulated on the same computer as the entity that fired the projectile acts as the master. Projectile impacts are only simulated for the master instance.
- Guided missiles and grenades are simulated on the computer where the entity that fired the projectile is simulated.

When developing scripts and plugins for VBS, it is important to be aware of these rules.

3.2 Tracking projectiles in VBS

VBS has a main simulation loop which executes the simulation. For each round, the main simulation loop increases time with one *time step/simulation step*, or *tick* and updates the state of the simulated entities, objects and projectiles. Figure 3.3 shows the positions of a projectile in VBS (marked with white spheres), which is tracked for each simulation step. With a typical simulation step lasting between 20 and 30 milliseconds, fast-moving projectiles can travel as much as 50 meters per simulation step.

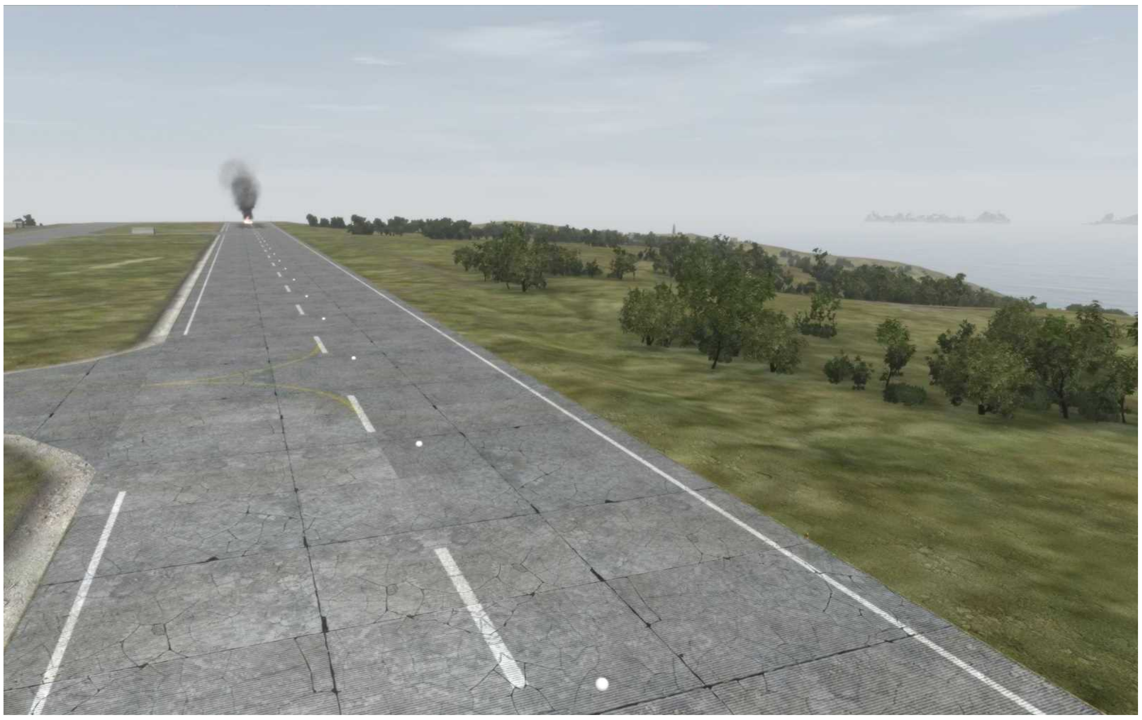


Figure 3.3 The positions of a projectile in VBS (marked with white spheres), which is tracked for each simulation step.

3.3 VBS scripting language

The VBS engine has its own 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

The VBS scripting language now includes more than 2,300 scripting commands and a function library which contains 316 pre-made utility functions. It is continuously growing with each new version of VBS. The syntax and control structures used by the VBS scripting language are somewhat similar to the control structures found in C/C++ and Java. However, the VBS scripting language is not organised in an object-oriented manner, so it can sometimes be challenging to find which scripting commands to use for implementing some desired functionality. The VBS scripting language is used to develop additional functionality for the models in VBS and to create scripted behaviour in missions.

Defined in the VBS scripting language is also a set of event handlers that allows scripts to be executed when certain events occur in the simulated environment. Examples of such events are when a model is initialized, when a weapon is fired, or when a model is hit. Documentation for the VBS scripting language can be found in the VBS Scripting Reference in the Bohemia Interactive Simulations Wiki [7].

3.4 VBSFusion

VBSFusion is a C++-based API for VBS, developed by SimCentric Technologies. It provides a comprehensive object-oriented C++ library for developing plug-ins for VBS. The plug-ins are compiled as dynamic link libraries (DLLs), which can be loaded by the VBS engine.

VBSFusion consists of the following components:

- A set of callback functions which are called by the VBS engine at different stages of the simulation. Examples of such functions are OnMissionStart(), OnMissionEnd(), and OnSimulationStep().
- A collection of data classes that are used to store and maintain data fetched from the VBS engine. The intention of this design concept is to minimize the need for directly accessing the VBS engine and thereby minimizing the negative impact VBSFusion plug-ins have on the performance of the VBS engine.
- A collection of utility classes for accessing the VBS engine directly.
- A set of event handlers that allows functions to be executed when certain events occur.

Figure 3.4 illustrates the components of a typical VBSFusion plug-in. More information about VBSFusion can be found in the VBSFusion User Guide [8].

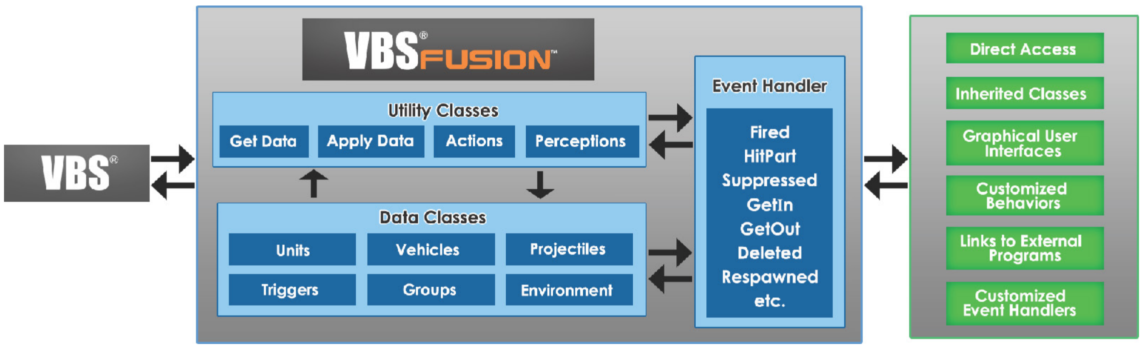


Figure 3.4 The components of a typical VBSFusion plug-in (SimCentric Technologies).

4 Modelling and implementation

4.1 Modelling

The modelled APS behaves like a real APS and performs the sequence of actions summarized in Figure 4.1 and illustrated more closely in Figure 2.2 (see Chapter 2.2).

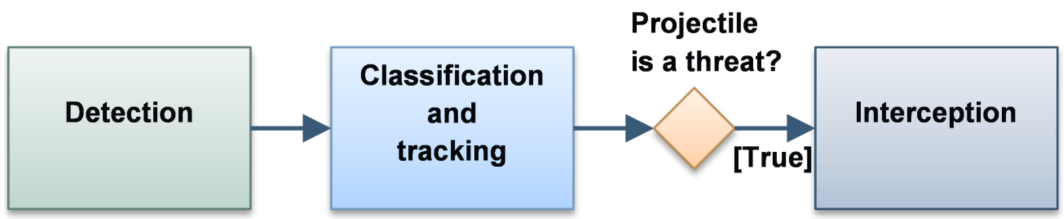


Figure 4.1 The sequence of actions performed by an APS.

4.2 Implementation

We have tried to implement the APS model both using VBSFusion and the VBS scripting language, but as explained in Chapter 4.2.1, the VBS scripting language has functionality that makes the implementation more efficient.

4.2.1 Detection

The brute force implementation for detecting incoming projectiles would be to create a loop that continuously checks for projectiles within a certain radius of the vehicle. While this solution might work in a virtual simulation where each vehicle is simulated on the VBS client operated by its driver, it would not scale very well in a constructive simulation where all vehicles are controlled by artificial intelligence (AI) and simulated on the VBS server.

A more efficient solution would be to have an event handler in the VBS engine that triggers each time a projectile moves within a certain radius of a vehicle. Functionality that let us do just that was introduced in the VBS scripting language in VBS version 3.7, which was released in July 2015.

In the VBS scripting language it is (from version 3.7) possible to create an invisible collision volume around an object. This collision volume will trigger a collision event each time it collides with another object, but it will not affect the simulation of the colliding objects in any way. The collision volume can be shaped either like a sphere or like a rectangular cuboid. Figure 4.2 shows a cuboid-shaped collision volume (to the left) and a sphere-shaped collision volume (to the right) created around a main battle tank (MBT) in VBS.

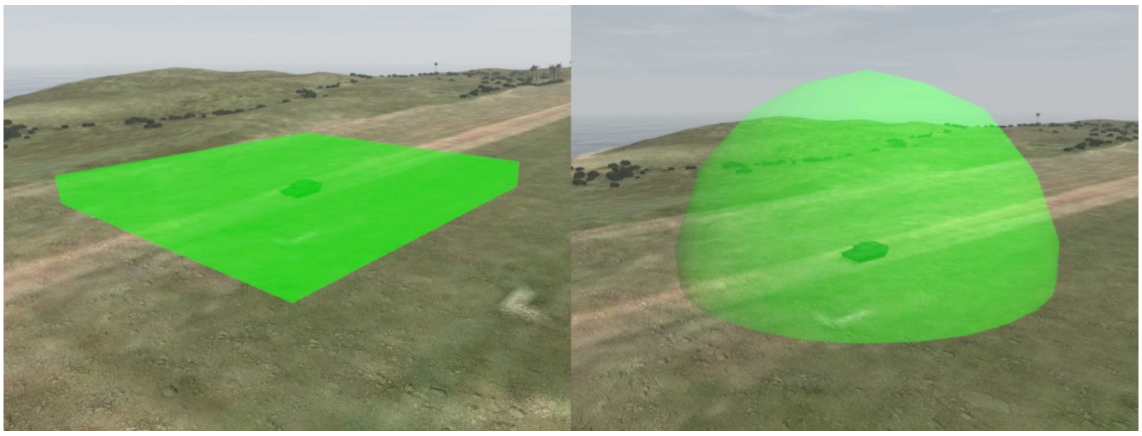


Figure 4.2 Visualization of a cuboid-shaped collision volume (to the left) and a sphere-shaped collision volume (to the right) created around a vehicle in VBS.

Which collision volume shape to use, depends on the capabilities of the modelled APS. If the APS only protects against horizontal attacks, a cuboid-shaped collision volume with low height is most suitable. If the APS also protects against vertical attacks, or projectiles flying straight above the target and firing EFPs downward, a sphere-shaped collision volume may be the best choice.

The size of the collision volume must be set based on what types of projectiles the modelled APS is able to intercept. As mentioned in Chapter 3.2, fast-moving projectiles in VBS can travel as much as 50 meters per simulation step. If the size of the collision volume is set too small, the projectile will be outside the collision volume in one simulation step and already have hit the vehicle in the next. If the APS is only capable of intercepting anti-tank missiles, a collision volume radius of 10 to 15 meters will be sufficient, but if the APS is to intercept kinetic energy penetrators (KEPs) the collision volume needs to have a radius of at least 50 meters. The APS model may use multiple collision volumes; for instance one large collision volume reacting to fast-moving projectiles and one smaller collision volume reacting to slower moving projectiles.

4.2.2 Classification and tracking

We have implemented a collision event handler script which is executed each time an object collides with the invisible collision volume around the vehicle. It is also triggered if an object is created inside the collision volume (and thus collides with the interior of the volume).

The next step the APS model has to perform is to classify the object colliding with the collision volume. The most straight forward way to do this in VBS is to classify the projectiles based on their object classes, but it is also possible to use speed and weight or size (volume of bounding box) of a projectile for classification. We have used the projectile class to classify the projectiles, and depending on the capabilities of the modelled APS, we specify a set of projectile base classes that the APS will react to.

If the incoming projectile is classified as a potential threat, the APS model needs to track the projectile and calculate whether it is going to hit the vehicle or not. If the APS is going to be able to react to fast-moving projectiles, it is essential that these calculations are executed very fast (within a fraction of a simulation step).

In VBS the position and velocity vector of the incoming projectile can be found directly. To quickly estimate if the projectile is going to hit the vehicle, we calculate the *angular diameter* of the vehicle seen from the position of the projectile and check if the horizontal direction of the projectile lies within this angle, as illustrated in Figure 4.3. The angular diameter δ can be found using the formula:

$$\delta = 2 \arctan\left(\frac{d}{2D}\right), \tag{4.1}$$

where d is the actual diameter or length of the vehicle, and D is the distance from the projectile to the vehicle. The length of the vehicle can be calculated from the bounding box of the 3D model. Figure 4.4 illustrates the bounding box of a main battle tank (MBT).

If we assume that the vehicle will have a maximum velocity of 60 km/h, it will not travel more than 0.5 meters during one simulation step. To compensate for this potential movement we can add one meter to d in equation (4.1).

Depending on the shape of the collision volume, it may also be necessary to check if the vertical direction of the projectile lies within the angular diameter of the vehicle, which can then be calculated using the height of the vehicle.

It should also be checked that the vehicle's sensors has line of sight (LOS) to the incoming projectile, and this can be done by checking that at least one of the upper corners of the vehicle's bounding box has LOS to the projectile.

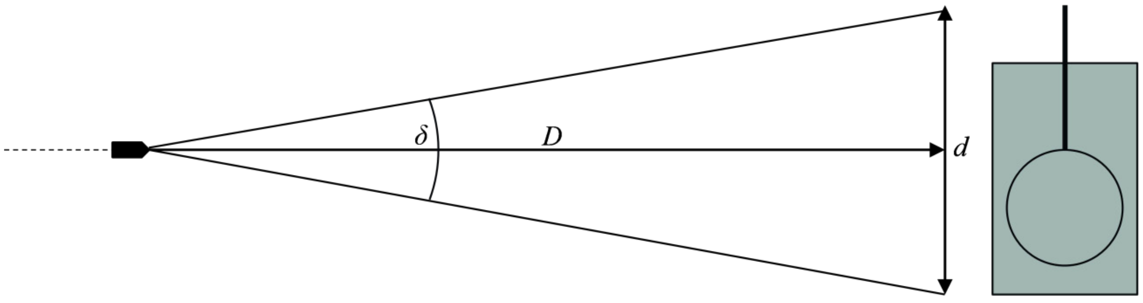


Figure 4.3 Angular diameter of the vehicle seen from the position of the projectile.

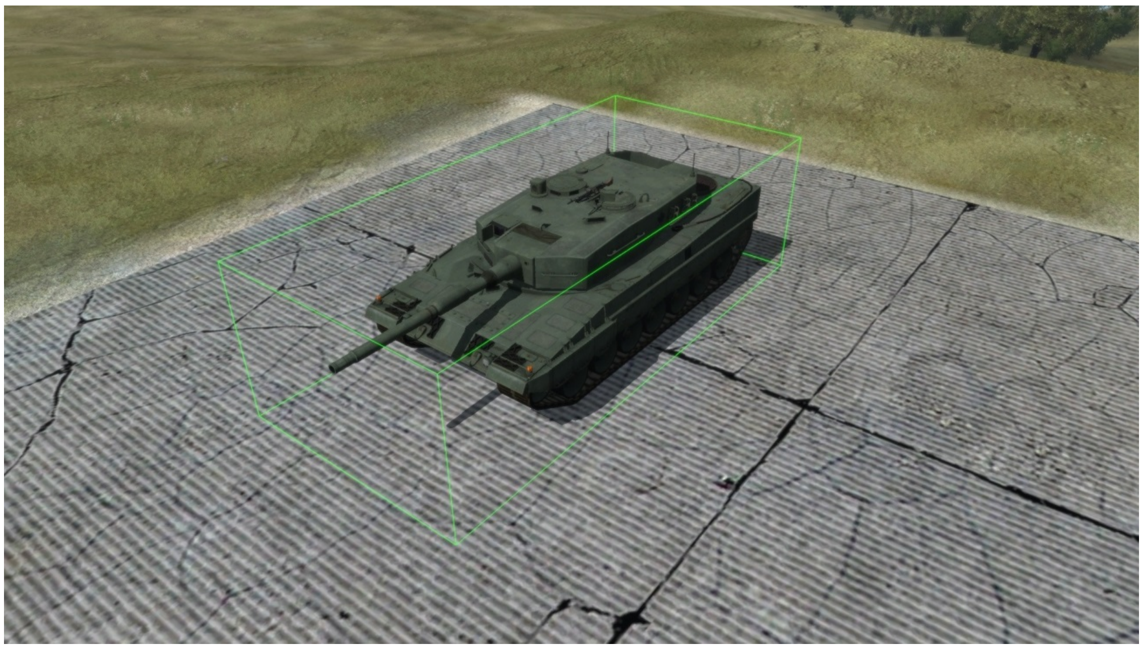


Figure 4.4 The bounding box of the 3D model of a main battle tank in VBS.

4.2.3 Interception

Before intercepting the incoming projectile the APS model has to determine which APS module to trigger and then check if there are any countermeasures left in this APS module. Some APSs may also require a certain amount of time to recharge after an activation of countermeasures.

To intercept the incoming projectile we create an invisible cuboid-shaped object (an invisible "wall") between the vehicle and the projectile. This object only has a fire geometry (but no visual geometry). The size of the cuboid will depend on the size of the sectors which are covered by the modelled APS, but we have modelled it to be one meter thick. How far from the vehicle the cuboid is created will depend on the specifications of the modelled APS.

Depending on the capabilities of the modelled APS, it is possible to adjust the material properties of the cuboid so that it can be penetrated by KEPs. The KEP will then hit the vehicle, but with a reduced velocity and consequently cause less damage. High explosive (HE) projectiles in VBS will always explode when they hit the cuboid. The cuboid is programmed to be deleted as soon as it is hit by the incoming projectile. It is also programmed to delete itself after 0.3 seconds, so even if it is not hit by the projectile it will only exist for a very short time.

The main structure of the collision (CollisionStart) event handler script, which is executed every time an object collides with the invisible collision volume, is outlined in Figure 4.5. Figure 4.6 shows a series of images illustrating how the APS model is implemented in VBS and visualizes the detection, interception, and destruction of an incoming projectile.

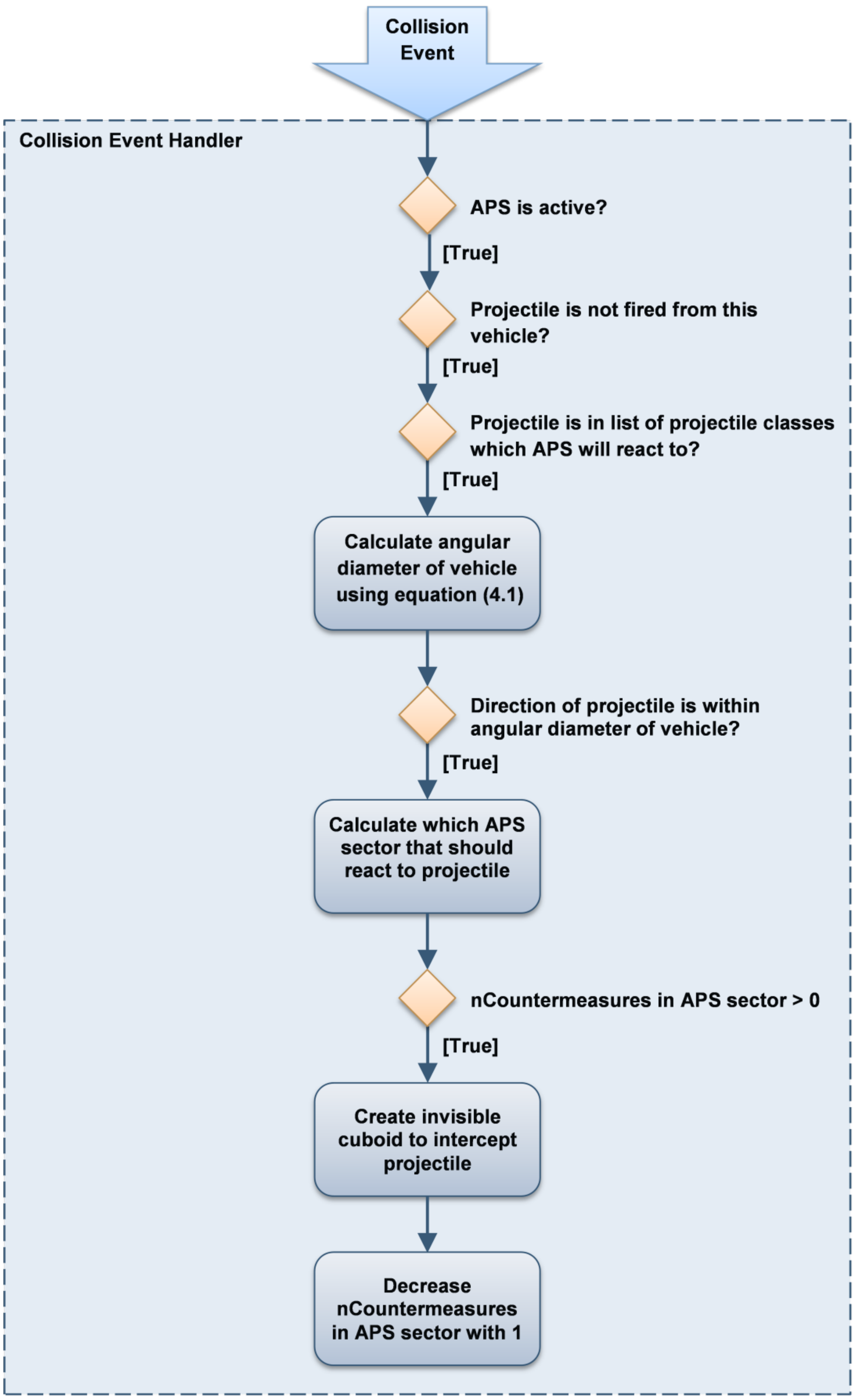

Collision Event
Collision Event Handler
APS is active?
[True]
Projectile is not fired from this vehicle?
[True]
Projectile is in list of projectile classes which APS will react to?
[True]
Calculate angular diameter of vehicle using equation (4.1)
Direction of projectile is within angular diameter of vehicle?
[True]
Calculate which APS sector that should react to projectile
nCountermeasures in APS sector > 0
[True]
Create invisible cuboid to intercept projectile
Decrease nCountermeasures in APS sector with 1

Figure 4.5 *Outline of the collision event handler script which is executed every time an object collides with the invisible collision volume around a vehicle equipped with an APS.*

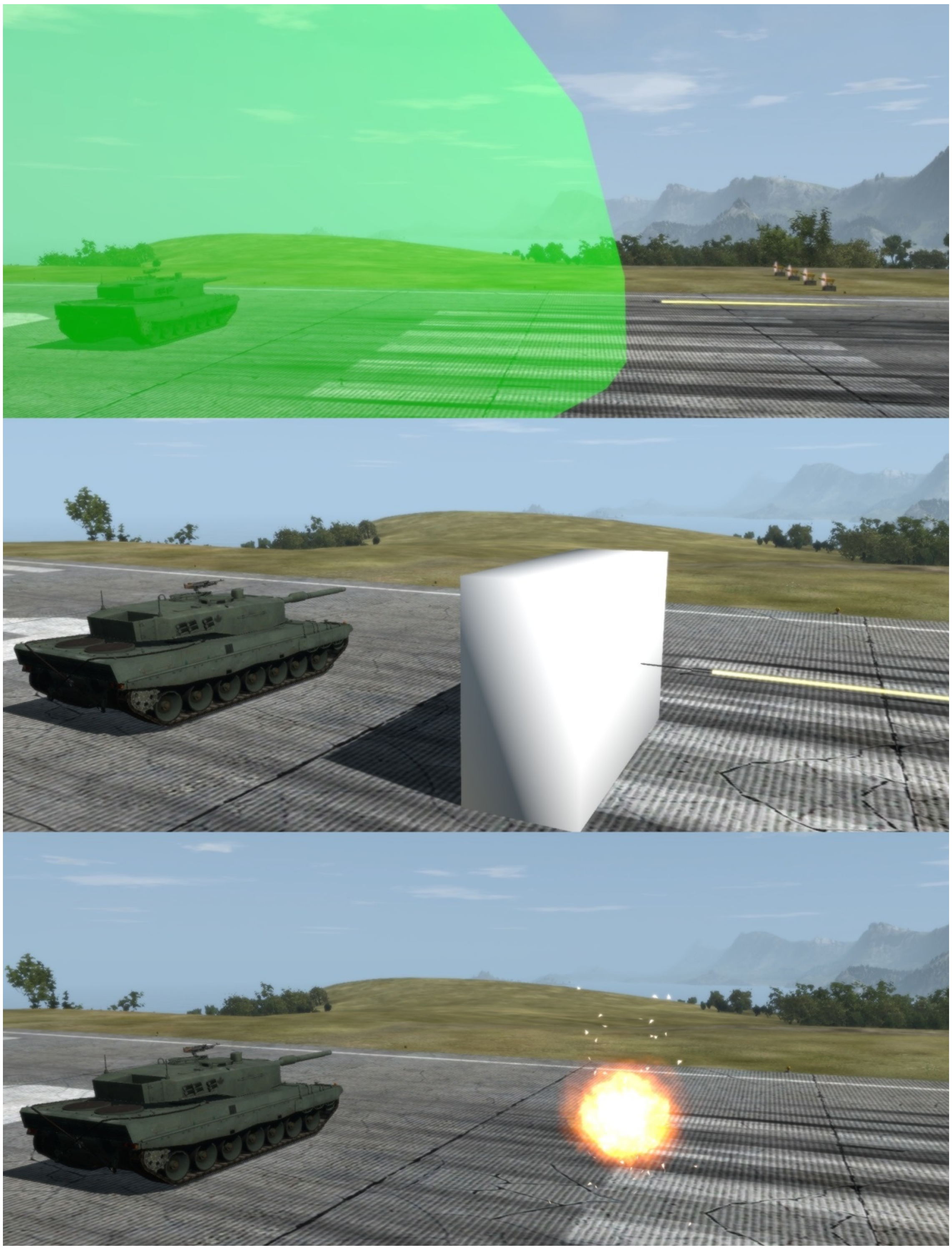


Figure 4.6   A series of images illustrating how the APS model is implemented in VBS, with detection (from the top), interception, and destruction of an incoming projectile.

4.3 Configuration

To make the configuration of the APS model easy, we have defined a class named ActiveProtectionSystem in the configuration file (config.cpp) for the vehicle equipped with APS. The ActiveProtectionSystem class is defined as an internal class in the configuration class defining the vehicle.

In the ActiveProtectionSystem class we have also defined an internal class for each sector covered by the APS. We have named these classes: class Sector1, class Sector2, etc., and the variable nSectors in the ActiveProtectionSystem class specifies how many sectors that are defined. Figure 4.7 illustrates four possible ways of dividing the circle around a vehicle into sectors covered by different APS modules.

The parameters we have defined in class ActiveProtectionSystem are listed in Table 4.1, and the parameters we have defined in class SectorN are listed in Table 4.2. These parameters are read by the initialization (Init) event handler script for the vehicle and assigned as object variables for the vehicle (using the setVariable scripting command). In addition we have defined the parameter bHasAPS in the configuration class for the vehicle. This parameter is used to determine if a vehicle is equipped with an APS. Figure 4.8 shows an example of a vehicle configuration class containing the definition of an APS with four sectors.

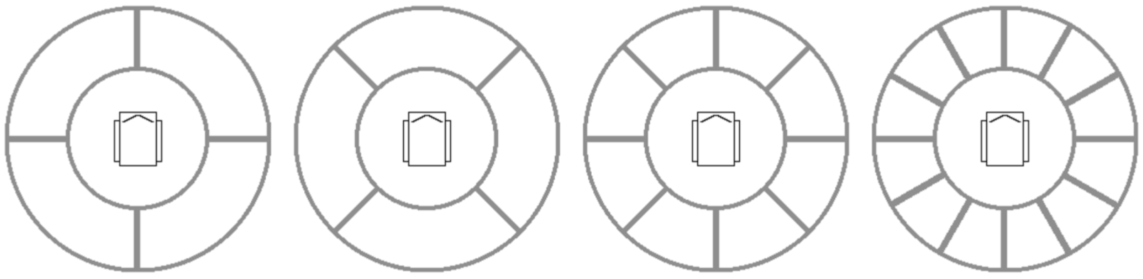


Figure 4.7 Examples of APSs with four, eight, and twelve sectors.

Parameter	Description
bAPSActive	If true, the APS is activated.
detectionVolumeType	Collision volume type (cuboid or sphere).
detectionRange	Radius of collision volume (in meters).
detectionHeight	Height of collision volume (in meters). Used only for cuboid.
interceptDistance	Intercept Distance from vehicle (in meters).
handledProjectiles	Array of projectile classes handled by APS.
nSectors	Number of sectors covered by the APS.
GUIType	Enumeration used for configuration of GUI.
vehicleType	Enumeration used for configuration of GUI.

Table 4.1 The parameters defined in class ActiveProtectionSystem.

Parameter	Description
nCountermeasures	Number of APS countermeasures in sector.
minAngle	Minimum horizontal angle of sector (in degrees).
maxAngle	Maximum horizontal angle of sector (in degrees).

Table 4.2 The parameters defined in class SectorN.

```
class CfgVehicles {
    class FFI_NO_Army_Leopard_2A4_APS_W_X : VBS2_CA_Army_Leopard_2A4_W_X {
        bHasAPS = true;

        class ActiveProtectionSystem {
            bAPSActive = true;
            GUIType = 2;
            vehicleType = 1;
            interceptDistance = 10;
            detectionVolumeType = "cuboid";
            detectionRange = 40;
            detectionHeight = 10;
            handledProjectiles[] = {
                vbs2_ShellBase, vbs2_MissileBase, vbs2_RocketBase
            };
            nSectors = 4;

            class Sector1 {
                nCountermeasures = 2;
                minAngle = -45;
                maxAngle = 45;
            };

            class Sector2 {
                nCountermeasures = 2;
                minAngle = 45;
                maxAngle = 135;
            };

            class Sector3 {
                nCountermeasures = 2;
                minAngle = 135;
                maxAngle = 225;
            };

            class Sector4 {
                nCountermeasures = 2;
                minAngle = 225;
                maxAngle = 315;
            };
        };
    };
};
```

Figure 4.8 Example of a configuration class for a vehicle with an APS with four sectors.

5 Graphical user interface

We have implemented a graphical user interface (GUI) for the APS. The GUI is designed to be used by the vehicle commander, when the APS model is used in virtual simulations. The GUI shows if the APS is active or turned off and visualizes the status of each of the defined sectors. The vehicle commander can turn the APS on and off by using the action menu in VBS. Figure 5.1 shows examples of different statuses of an APS with four sectors, each having two sets of countermeasures. In the first example (from the left) the APS is fully charged with countermeasures, and all sectors have status *green*. In the second example, the front sector has used one set of countermeasures and has status *yellow*. Finally, in the third example, the front sector has depleted both sets of countermeasures and has status *red*. Figure 5.2 shows images from VBS with the APS GUI visible in the upper left corner.

The GUI has been implemented as a plug-in for VBS using VBSFusion. The GUI is quite simple, so it should be possible to implement it using the VBS scripting language as well, but VBSFusion has an API for drawing head-up displays (HUDs) for VBS which is much easier to use.

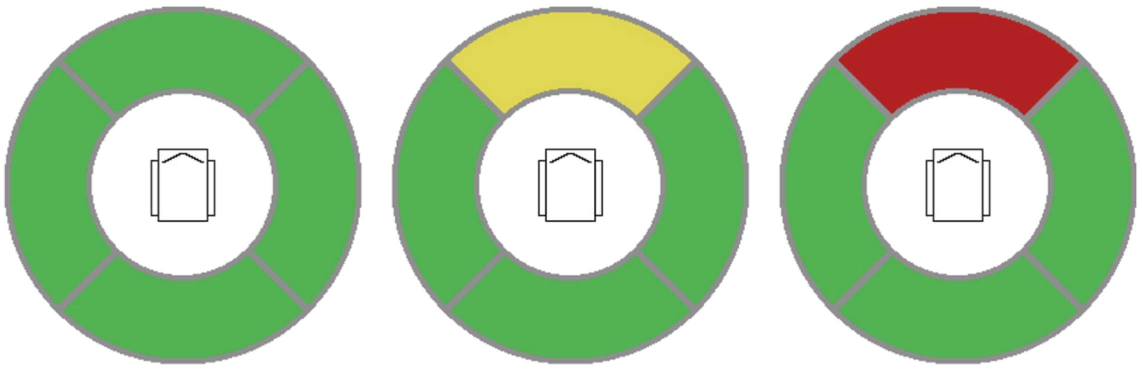


Figure 5.1 Examples of different statuses for an APS with four sectors, each having two sets of countermeasures.

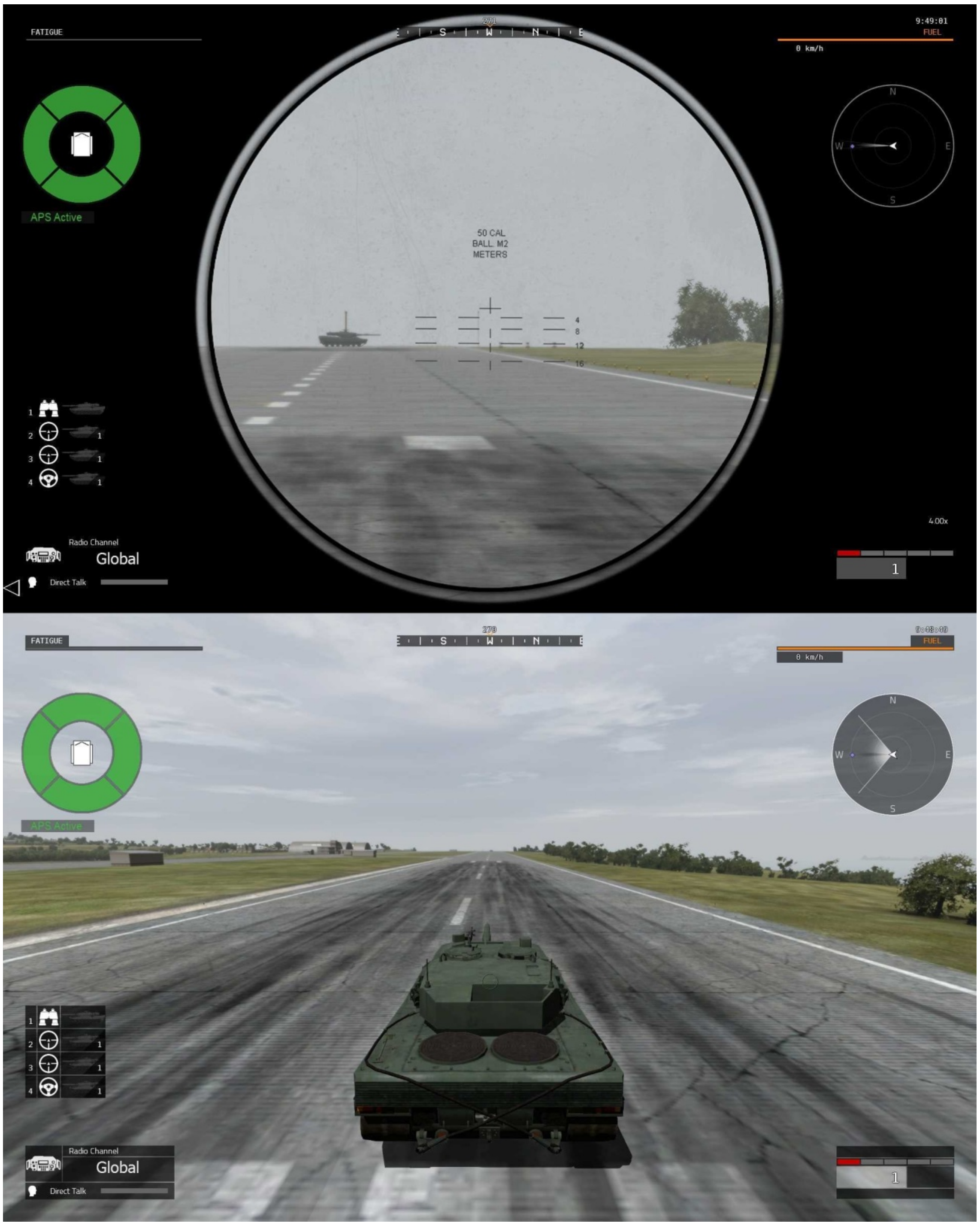


Figure 5.2 Images of from VBS with the APS GUI visible in the upper left corner.

6 Summary and conclusion

This report has described the conceptual model and the implementation of a hard-kill active protection system (APS) for entities in the simulation tool Virtual Battlespace (VBS). The APS model has high fidelity and performs the same sequence of actions as a real APS. It detects, classifies and tracks, and intercepts incoming projectiles. The APS model is generic and can easily be configured to simulate most existing hard-kill APSs. It can be used by both *virtual* and *constructive* entities in VBS.

The APS model has been implemented using the VBS scripting language. For virtual entities a graphical user interface (GUI) showing the status of the APS has been designed and implemented using VBSFusion, which is a C++-based application programming interface (API) for VBS.

References

[1] P-I. Evensen & D.H. Bentsen, *Simulation of Land Force Operations – A Survey of Methods and Tool*, FFI-rapport 2015/01579, 2016.

[2] M. Halsør, S. Martinussen, P. I. Evensen & B. Hugsted, *Uttesting av BMS i syntetisk miljø*, FFI-rapport 2007/00139, 2007.

[3] Bohemia Interactive Simulations (BISim) – Virtual Battlespace 3 (VBS3), https://bisimulations.com/virtual-battlespace-3.

[4] P-I. Evensen & M. Halsør, *Experimenting with Simulated Augmented Reality in Virtual Environments*, Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2013, Paper No. 13028, 2013.

[5] P-I. Evensen & M. Halsør, *Using Virtual Environments to Evaluate the Operational Benefit of Augmented Reality*, NATO Modelling and Simulation Group (NMSG) Annual Symposium 2014 (STO-MP-MSG-126), Paper No. 5, 2014.

[6] Bohemia Interactive Simulations (BISim), *VBS Gateway Version 2.3 for VBS3 3.9.2*, 2016.

[7] Bohemia Interactive Simulations (BISim) Wiki – VBS Scripting Reference, https://resources.bisimulations.com/wiki/Main_Page.

[8] SimCentric Technologies, *VBSFusion v3.9.2 User Guide*, 2016.

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
APS	Active Protection Systems
BISim	Bohemia Interactive Simulations
DIS	Distributed Interactive Simulation
DLL	Dynamic Link Library
EFP	Explosively Formed Projectiles
GUI	Graphical User Interface
HE	High Explosive
HITL	Human-In-The-Loop
HLA	High Level Architecture
HUD	Head-Up Display
IDF	Israel Defence Forces
KEP	Kinetic Energy Penetrator
LOS	Line Of Sight
MBT	Main Battle Tank
SAF	Semi-Automated Forces
VBS	Virtual Battlespace

About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

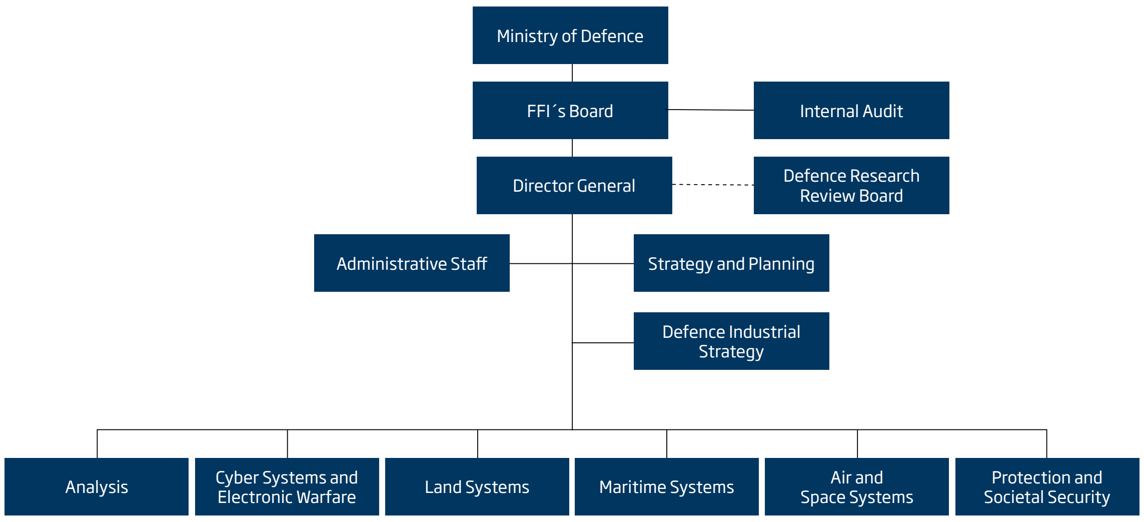
FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

FFI's organisation



Forsvarets forskningsinstitutt
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: ffi@ffi.no

Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: ffi@ffi.no

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment