

Carl Hedberg, Lars Tydén, Christer Wigren

EwSim - Electronic Warfare Real Time Simulation in the Visual and Infrared Wavelength Range



SWEDISH DEFENCE RESEARCH AGENCY

Command and Control Systems

P.O. Box 1165

SE-581 11 Linköping

FOI-R--0709--SE

December 2000

ISSN 1650-1942

Methodology report

Carl Hedberg, Lars Tydén, Christer Wigren

EwSim - Electronic Warfare Real Time Simulation in the Visual and Infrared Wavelength Range

Issuing organization FOI – Swedish Defence Research Agency Command and Control Systems P.O. Box 1165 SE-581 11 Linköping	Report number, ISRN FOI-R--0709--SE	Report type Methodology report
	Research area code 6. Electronic Warfare	
	Month year December 2000	Project no. E7015
	Customers code 5. Commissioned Research	
	Sub area code 61 Electronic Warfare including Electromagnetic Weapons and Protection	
Author/s (editor/s) Carl Hedberg Lars Tydén Christer Wigren	Project manager Peter Klum	
	Approved by Mikael Sjöman	
	Sponsoring agency	
	Scientifically and technically responsible	
Report title EwSim - Electronic Warfare Real Time Simulation in the Visual and Infrared Wavelength Range		
Abstract (not more than 200 words) A framework, <i>EwSim</i> , for electronic warfare duel simulations, in the visual/infrared wavelength range has been developed and is described in the report. <i>EwSim</i> is based on the commercial product, <i>Vega</i> , with the plug-in for infrared imagery, <i>SensorVision</i> (the plug-in <i>SensorWorks</i> can also be used for sensor performance). Several in-house modules have been adapted to this framework in order to get EW-simulation capabilities. <i>EwSim</i> has real time (or close to real time) simulation capacity which means that it is possible to extend the use of the framework to include EW training of military personnel. This document gives a technical description of <i>EwSim</i> .		
Keywords Electronic warfare, simulation, visual, infrared		
Further bibliographic information	Language English	
ISSN 1650-1942	Pages 5 p.	
	Price acc. to pricelist	

Utgivare Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 Linköping	Rapportnummer, ISRN FOI-R--0709--SE	Klassificering Metodrapport
	Forskningsområde 6. Telekrig	
	Månad, år December 2000	Projektnummer E7015
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 61 Telekrigföring med EM-vapen och skydd	
Författare/redaktör Carl Hedberg Lars Tydén Christer Wigren	Projektledare Peter Klum	
	Godkänd av Mikael Sjöman	
	Uppdragsgivare/kundbeteckning	
	Tekniskt och/eller vetenskapligt ansvarig	
Rapportens titel (i översättning) EwSim - Telekrigsimulering i realtid inom det visuella och infraröda våglängdsområdet		
Sammanfattning (högst 200 ord) Ett ramverk, <i>EwSim</i> , för telekrigduellsimuleringar, i det visuella/infraröda våglängdsområdet har tagits fram och beskrivs i rapporten. <i>EwSim</i> baseras på den kommersiella produkten <i>Vega</i> , med en plug-in, <i>SensorVision</i> , för att skapa infraröda bilder (en plug-in för sensorprestanda, <i>SensorWorks</i> , kan också användas). För att erhålla kapacitet för telekrigsimulering har flera egenutvecklade moduler tagits fram. <i>EwSim</i> har kapacitet för simuleringar i realtid (eller nära realtid) vilket gör det möjligt att utöka användningsområdet av modulerna till träningssimulatorer. Denna rapport är en teknisk beskrivning av <i>EwSim</i> .		
Nyckelord Telekrig, simulering, visuell, infraröd		
Övriga bibliografiska uppgifter	Språk Engelska	
ISSN 1650-1942	Antal sidor: 5 s.	
Distribution enligt missiv	Pris: Enligt prislista	

Contents

1	INTRODUCTION	5
2	VEGA	5
2.1	The Instance List	5
3	FLARE AND SMOKE COUNTERMEASURES	5
3.1	Application Interface	5
3.2	Initializing the EwCM module	5
3.3	The EwCM module	5
3.4	Flare/Cloud	5
3.4.1	Control points	5
3.4.2	Diffusion parts	5
3.4.3	Diffusion size	5
3.4.4	SV or Visual->IR	5
3.4.5	Particle Temperature Function Table	5
3.5	Expendable/Decoy	5
3.5.1	Expendable Aerodynamics	5
3.5.2	Sub-Munitions Table	5
3.6	The Vega LynX EwCM Dispenser Panel	5
3.7	Sequence	5
3.8	Control box	5
4	DIRCM	5
4.1	Application Interface	5
4.2	Initializing the EwCmLaser module	5
4.3	EwCmLaser modules	5
4.4	Laser trigger	5
4.5	Transmitter	5
4.6	Laser	5
5	EXTRACTING IMAGES	5
5.1	Application Interface	5
5.2	Initializing the EwImage Module	5
5.3	The EwImage module	5

5.4	EwImage	5
5.5	SmallTarget	5
6	TARGET SEEKERS/TRACKERS	5
6.1	Application Interface	5
6.2	Initializing the EwSeeker Module	5
6.3	EwSeeker	5
7	MISSILE MOTION	5
7.1	Application Interface	5
7.2	Initializing the EwMotion Module	5
7.3	The EwMotion module	5
7.4	EwMotion	5
7.5	EwGraph	5
8	SUMMARY AND CONCLUSIONS	5
9	REFERENCES	5

1 Introduction

A missile threat is a reality in many military operations. It is therefore very important that measures can be taken to counter these threats. A common method to assess countermeasures and how they are used is by means of simulations. To build the framework for such simulations is a large undertaking. Therefore it is valuable to create this framework with many different military platforms in mind; in the air, at sea, and on land. When simulating visual or infrared (IR) scenarios in a cluttered environment, e.g. ground vehicles in broken ground, then a full three dimensional description with textures for the targets, background, and countermeasures have to be used. For simulation of a scenario where background clutter is negligible, e.g. a missile approaching an aircraft from below on a cloud free day, then simulations using point like objects for the missile target and countermeasures might be appropriate. However, even in such cases 3D descriptions of targets and countermeasures are valuable since hot spots on the target and countermeasures can be obscured by parts of the target for certain geometrical configurations.

Results from simulations in an electronic warfare (EW) duel simulations can be used for different purposes. A simulation where the background, targets, and countermeasures have been modeled in great detail can be used to increase the understanding for certain technical aspects of the scenario. The simulation package, OPTSIM [1], is an example of a simulation environment with very detailed models for EW duel simulations. If the simulations are based on very detailed models this can make the simulations slow and that makes it impossible to do a large number of simulations in a reasonable amount of time. However, the ability to use the simulation results as the basis for tactical recommendations often demands a large number of simulations. When the simulation models are built, the accuracy of the models has to be weighted against the simulation time. Furthermore, if the simulations can be brought to real time performances then the models can also be used in realistic training of personnel for military platforms equipped with countermeasure systems. Traditionally the use of images from a 3D model in an EW simulation, especially if IR imagery is involved, for other purposes than to visualize the simulation result, has been in conflict with real time simulations. In the last few years this has changed and it is now possible to purchase commercial-off-the-shelf products as a simulation framework [2,3] and add in-house created modules for countermeasures, missile seekers, missile dynamics, etc.

This document contains a technical description of a real time framework for visual and IR EW simulations with modules for countermeasures (flare, smoke, and laser), missile seekers, and missile dynamics. The simulation package or framework is called *EwSim* (electronic warfare simulations) and is based on the commercial product, *Vega*, which is responsible for generating images and handles interaction between objects, background, users, etc. Infrared imageries in this framework is created by the use of a plug-in to *Vega* called *SensorVision*. Sensor performance can be added by another plug-in called *SensorWorks*. However, the functionality for EW simulations can not be purchased on the commercial market. Therefore, several in-house modules have been created to extend the functionality of these commercial products. These in-house modules and the principle for their use in simulations are displayed in figure 1.

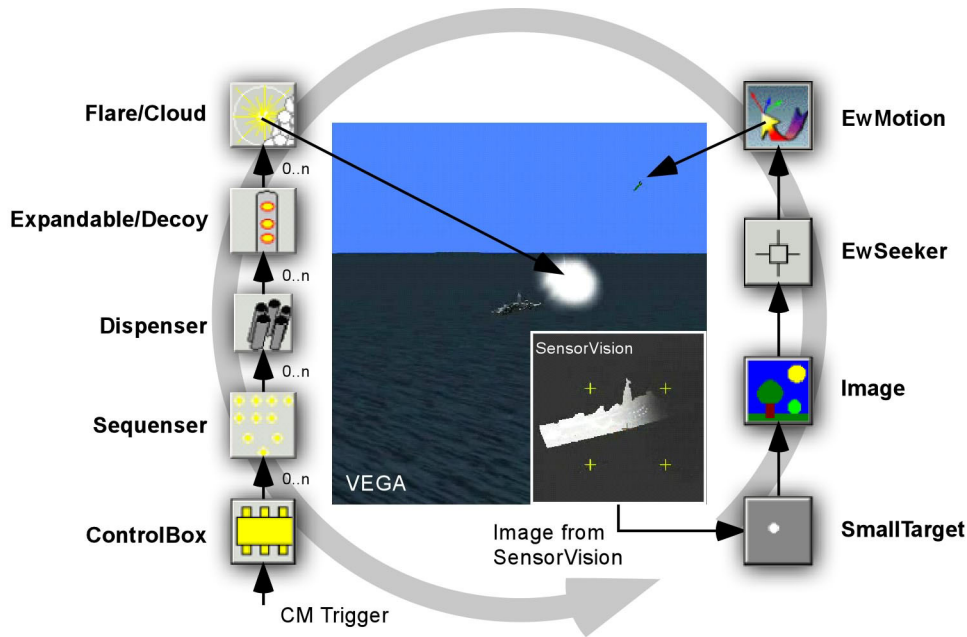


Figure 1 The simulation package for EW simulations called EwSim. EwSim is based on Vega/SensorVision for generating images from 3D models of targets and background. Several modules have been added for generating countermeasures triggered manually or by a warning system. To the left are the modules for creating smoke or flares (ControlBox, Sequenser, Dispenser, Expandable/Decoy, Flare/Cloud). Other modules exist for a laser jammer. To the right are modules needed for the missile simulation. An image is extracted (SmallTarget and/or Image) and sent to a target seeker/tracker (EwSeeker). The estimated target location from the target seeker is sent to the missile dynamics module (EwMotion) which will control the position of the missile in the simulation. The position and orientation of the missile will then determine how the next seeker image is created.

An alternative to Vega, SensorVision, and SensorWorks is to use Mantis and viXsen from CG² [3]. However, the availability of the former products is better and they are also used at other projects within FOI.

The purpose of EwSim is to provide a real time simulation environment for visual and IR EW-simulations. EwSim should be applicable to a wide range of scenarios and for instance effects from details in the signature of the target and background should be included. This means that images of 3D scenarios seen from the target tracker/seeker have to be generated and processed in the target seeker model. The real time property means that the large number of simulations needed for extraction of tactical recommendations can be made within a reasonable amount of time. It also means that the models can be used in training simulators with a human in the loop. The disadvantage is that the real time property in the models demands approximations that were not needed in for instance the OPTSIM models [1]. Therefore, OPTSIM is a good complement to EwSim and can be used to verify details in results from EwSim.

EwSim is not an application but rather a set of modules. This means that EwSim can be tailored for use in different applications and for different users. Applications made for technical specialists can therefore have a different set of parameters than an application made

for military non technical personell. An existing application which has made use of the modules in *EwSim* is an application for flare tactics on helicopters.

2 Vega

Vega™ is a software environment for virtual reality and real-time simulation applications. By combining advanced simulation functionality with easy-to-use tools, Vega provides a means of constructing sophisticated applications quickly and easily. Vega supports rapid prototyping of complex visual simulations.

LynX™ is the graphical user interface (GUI) for defining and previewing Vega applications. These Vega applications are programs that you create using the Vega development environment or using a basic Vega executable provided with all Vega packages. The parameters defined in the LynX are saved in an application definition file (ADF) which contains all the information that is needed for the initialization and some information used during run-time of the application.

A module consists of a collection of classes each represented in LynX by an icon panel. The *EwCM* (Electronic Warfare Counter Measure) module, for instance, currently consists of 5 Classes (panels) *Flare/Clouds*, *Expendable*, *Dispenser*, *Sequence* and *Control box* as shown in figure 2.

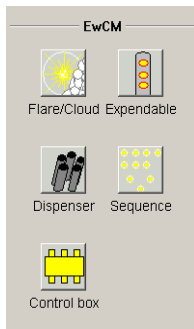


Figure 2 Example of an icon panel from the *EwCM* module

A panel displays the values for a collection of parameters called a class. All widgets in the panel adjust the values of parameters for instances of the class associated with that panel.

2.1 The Instance List

If it is possible to have more than one instance of a particular class, there is an instance list in the upper left corner of the panel. The items in this list are the user-supplied names for the instances of the class. The *Observers* panel is an example of a panel that contains an instance list, and the *System* panel is an example of a panel that can have only one instance and therefore doesn't have an instance list.

All instances of classes must be named. Instance names must be unique per class and are limited to 40 characters in length. Spaces are permitted in class names. It is permissible to have a channel named "left" and a window named "left", but there cannot be two channels named "left".

The widgets, in the panel, display the values of the parameters for the item (class instance) that is selected in the instance list.

3 Flare and Smoke Countermeasures

The Vega *EwCM* Module provides an easy to use library of real-time *Flares/Clouds* suitable for inclusion in a Vega application. The module includes interfaces for the creation of new user-defined *Flare/Clouds* for both Visual and IR images by using Vega and the IR plug-in SensorVision.

Based on the timing and properties set by the user, the *Flares/Clouds* can change shape, scale and color over time. Each *Flare/Cloud* contains a Vega *Display List* which describes the geometry and graphical attributes of the *Flare/Cloud*. Through the Vega Application Programmer's Interface (API), users can not only create and manage the *Flare/Cloud*, but also make *Flare/Clouds* of their own that include texture animation.

3.1 Application Interface

The *EwCM* module includes an API to C/C++ language callable routines for defining *EwCM* effects within Vega. In order to build Vega *EwCM* applications, an application must include the file *EwCM.h*, and link with the *EwCM* library. For Windows users, *psEwCMS.lib* should be used for static executables and *psEwCM.lib* (*psEwCM.dll*) should be used for dynamic executables.

3.2 Initializing the EwCM module

If Vega is to recognize the *EwCM* classes, an application must initialize the module after calling *vgInitSys* to initialize Vega. The function *InitEWCM* initializes module classes and the Special Effects module for use with Vega. A *VG_FAILURE* is returned if an error has occurred, and *VG_SUCCESS* is returned if not. Once this initialization has been done, an ADF containing *EwCM* class instances can be parsed by sending the ADF to *vgDefineSys*.

```
#include <vg.h>
#include <vgfx.h>
#include "EWCM.h"

main()
{
    vgInitSys();

    InitEWCM(); // vgInitFx(); is included
    vgInitSV();

    vgDefineSys( "myfxapp.adf" );

    vgConfigSys();

    while( 1 )
    {
        vgSyncFrame();
        vgFrame();

        /* application specific code */
    }
}
```

3.3 The EwCM module

The LynX *EwCM* panels contain widgets for setting the parameters that define the countermeasure. If you are unfamiliar with how to use the LynX interface, consult the Vega LynX User's Guide before proceeding [4].

The *EwCM* (Electronic Warfare Countermeasure) module currently consists of 5 Classes (panels) Flare/Clouds, Expendable, Dispenser, Sequence and Control box. The relationship between the classes is shown in the following class and object diagrams in figure 3 and 4.

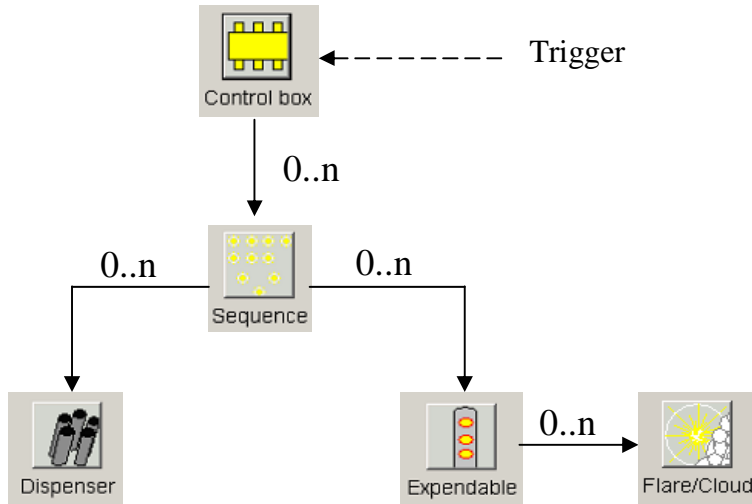


Figure 3 Class diagram of the EwCM module Classes

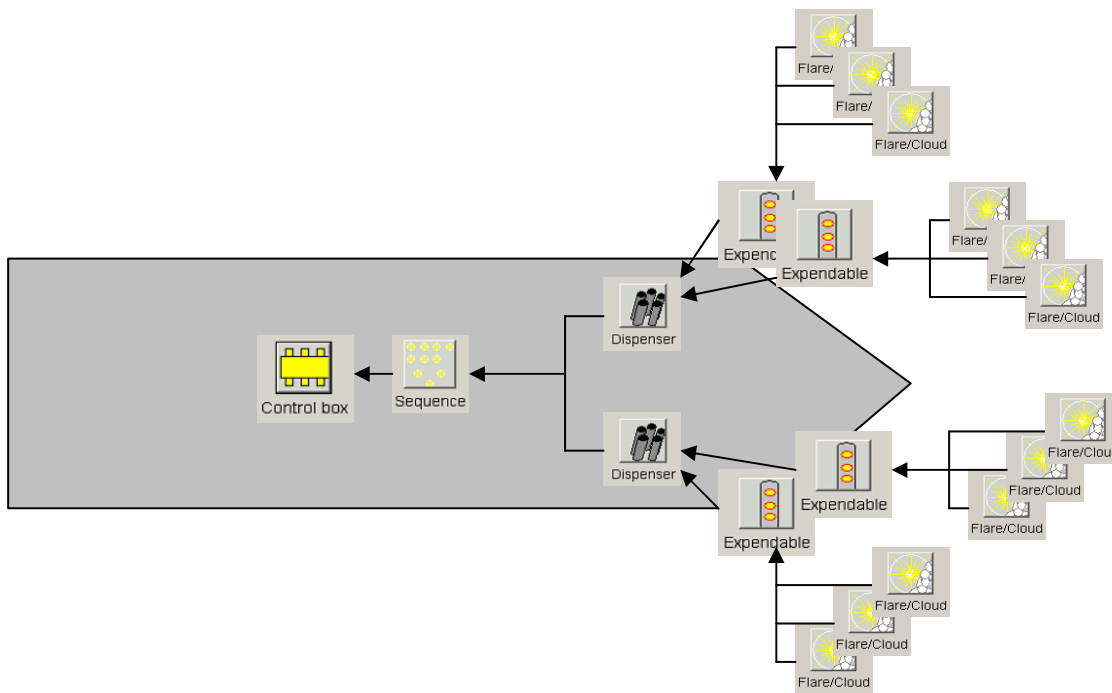


Figure 4 Object diagram that illustrates the relationship between the different objects. The figure is a ship with a control unit. This unit triggers different sequences consisting of dispensers loaded with expendables.

3.4 Flare/Cloud

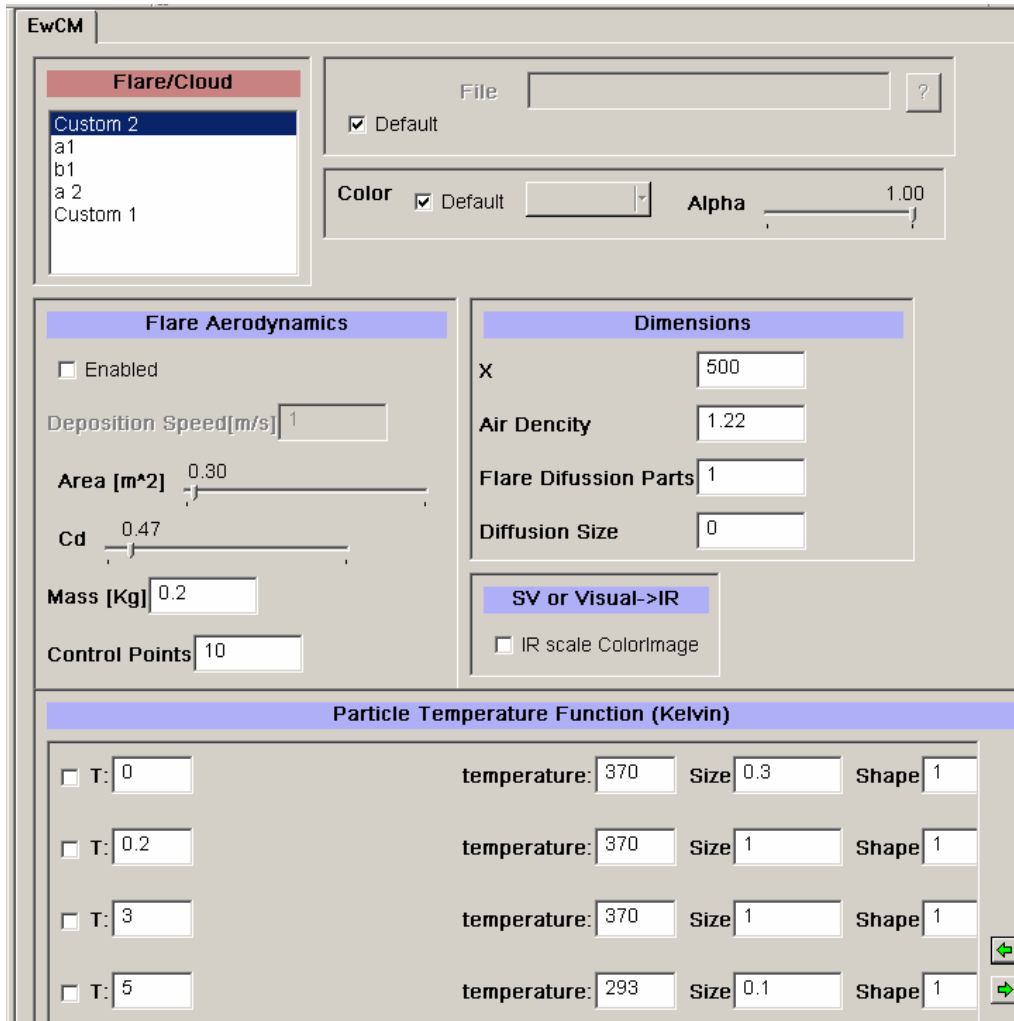


Figure 5 The Flare/Cloud panel.

The motion model for the flare (cloud) is computed by using aerodynamic equations for the flare. The vertical speed is optionally given as a constant by enabling the deposition speed. An approximation is that the mass and area are constant during the trajectory. The equations for the flares motion model are:

$$k = \frac{1}{2} \rho C_d A \quad (1)$$

$$V_x = V_0 e^{-\frac{kt}{m}} \quad (2)$$

$$V_z = V_0 e^{-\frac{kt}{m}} + \frac{mg}{k} \left(1 - e^{-\frac{kt}{m}} \right) \quad (3)$$

$$S_x = \frac{V_{0x}m}{k} \left(1 - e^{-\frac{kt}{m}} \right) \quad (4)$$

$$S_z = \frac{V_{0z}m}{k} \left(1 - e^{-\frac{kt}{m}} \right) e^{-\frac{kt}{m}} + \frac{mg}{k} \left(t + \frac{m}{k} \left(e^{-\frac{kt}{m}} - 1 \right) \right) \quad (5)$$

Variable	Description	Unit
m	Mass of the flare.	kg
A	Cross-sectional area.	m ²
ρ	Density of air (at ground 1.2 kg/ m ³)	kg/m ³
C _d	The drag coefficient of the flare which is around 0.75.	
V _{0x}	The horizontal start velocity.	m/s
V _{0y}	The horizontal start velocity.	m/s
V _x	The horizontal velocity.	m/s
V _z	The vertical velocity.	m/s
S _x	The horizontal position.	m
S _z	The vertical position.	m

3.4.1 Control points

This variable is the number of points determining the flares trajectory. In-between the points the flares position is interpolated with a spline-function.

3.4.2 Diffusion parts

The number of parts the flare is divided into.

3.4.3 Diffusion size

The random velocity component of a flare part is computed by scaling a pre-calculated vector. The vector is calculated when the flare starts so that successive states of a random particle velocity will differ. A random velocity scale of 5.0 would therefore add a vector with random direction of random magnitude between 0.0 and 5.0 units/sec to the overall particle velocity (units is the database unit normally in meters).

3.4.4 SV or Visual->IR

If the checkbox *IR scale ColorImage* is unchecked the flares radiance are calculated using SensorVision and in the Particle Temperature Function table the temperature is set as a function of time on the material set on the flare.

If the “IR scale ColorImage” is checked the intensity of the color image is used to calculate the radiance. Since the intensity in the color image is limited to 0-255 discrete values (graphic card limitation) and a large area of the image often contains information in the lower temperature band and a few small hot spots there is a possibility to set two different resolutions in radiance. The first resolution is set by the 0 to “breakpoint” grey levels in this span the lower radiance “Low Rad” to the break radiance “Break Rad” is set and in the same

way the break radiance to the high radiance is set in the span breakpoint – 255 grey levels. Also in the particle temperature function table the total intensity is set instead of temperature.

Figure 6 When the IR scale color image is checked these widgets appear.

3.4.5 Particle Temperature Function Table

This table determines the flares or smoke clouds radiation, size and shape as a function of time. If the checkbox IR scale ColorImage is unchecked the temperature is in Kelvin in this table (figure 5) otherwise it is the intensity (figure 7). The radiance is the intensity divided by the cross-section of the flare. Size is the diameter in meters and the shape of the flare is given at time T in Seconds. The shape factor is how the flare is scaled relative its motion vector and 1.0 means it is uniformly scaled.

Figure 7 When the IR scale color image is checked the intensity is set instead of temperature.

3.5 Expendable/Decoy

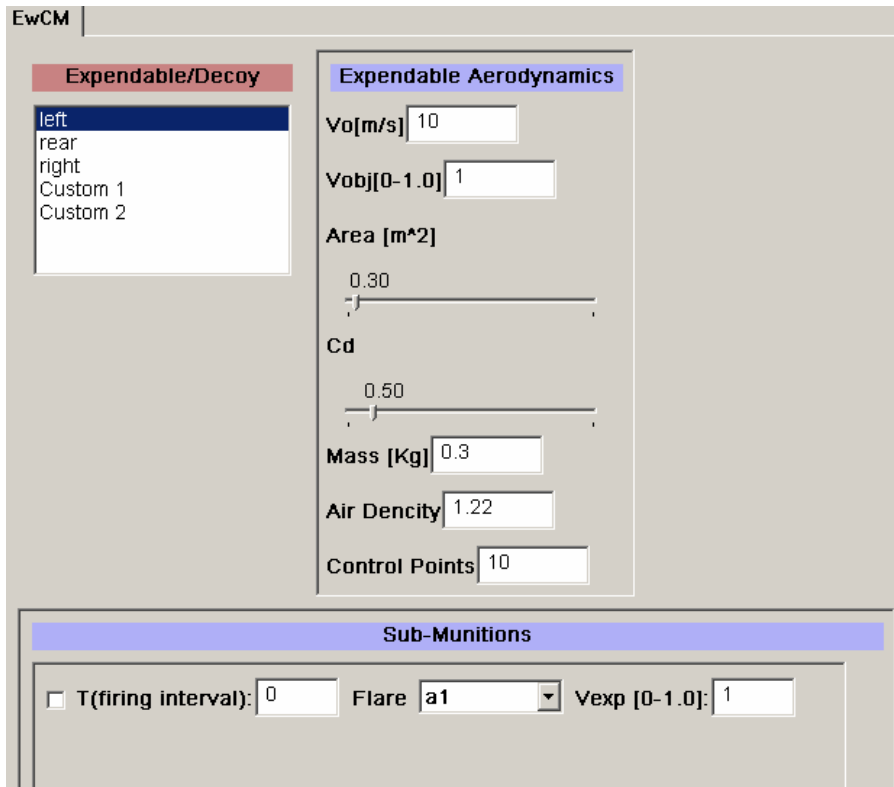


Figure 8 The Expendable panel.

3.5.1 Expendable Aerodynamics

The expendable's motion model is computed by using the same aerodynamic equations as for the flare. The only difference is that the parameter $V_{obj}[0-1.0]$ is the speed ratio from the carrying platform (*Object* in the *Dispenser* panel) that is going to be added to the expendables speed. For example if the V_{obj} is set to 0 the expendable will start with V_o as a initial speed and if the V_{obj} is set to 1.0 the initial speed of the expendable will be V_o plus the speed of the carrier at the time the expendable is fired.

3.5.2 Sub-Munitions Table

In this table the expendable is configured with sub-munitions (*Flare*) and the firing interval for each flare is given in seconds from the firing of the expendable. The $V_{exp}[0-1.0]$ is the speed ratio, initial speed flare/speed of expendable when flare released (see figures 9 and 10).



Figure 9 $V_{exp}[0-1.0]$ set to 1.0 means that the flares gets the expendables initial speed when released. Small dots show the trajectory of the expendable and larger dots show the trajectory of the flare.

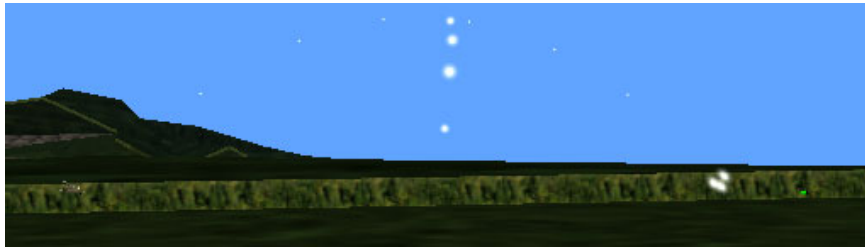


Figure 10 *Vexp[0-1.0]* set to 0.0 means that the flares get no initial speed when released. Small dots show the trajectory of the expandable and larger dots show the trajectory of the flare.

3.6 The Vega LynX EwCM Dispenser Panel

In this class the dispensers are placed on the 3D model of the object. This is done by naming the dispenser instance with the same name as the part representing the dispenser on the 3D-object see figure 11. The direction of the dispenser is given by heading, pitch and roll (all in degrees).

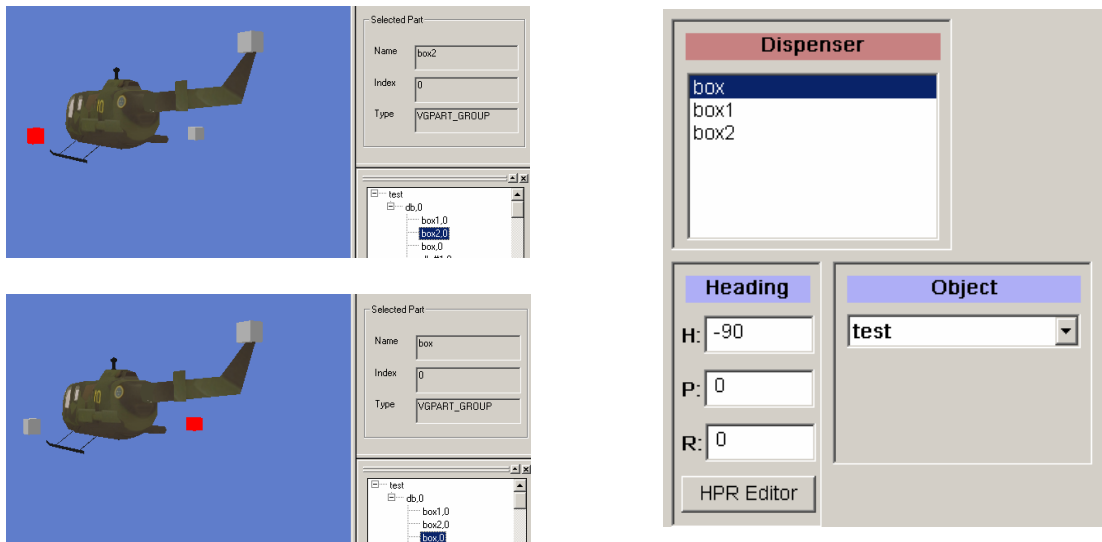


Figure 11 The dispenser's locations are given as parts of the 3D-modell. In the figure the red box called box in the 3D model is instanced as box in the dispenser panel and thereby given the same position.

3.7 Sequence

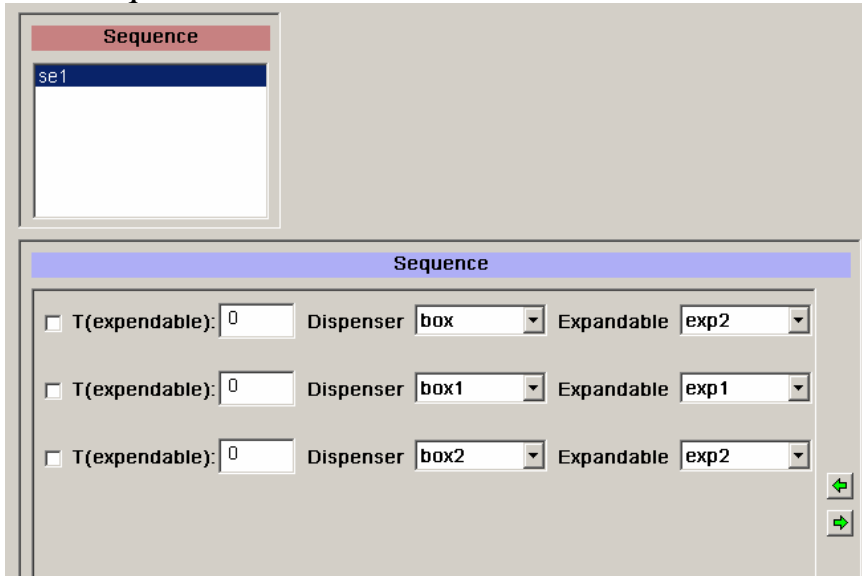


Figure 12 The Sequence panel.

In the sequence panel, sequences are defined as which dispenser instances and which expendable instances to be used as a function of time. The *Dispenser* and *Expendables* are both referenced to the instances of their classes.

3.8 Control box

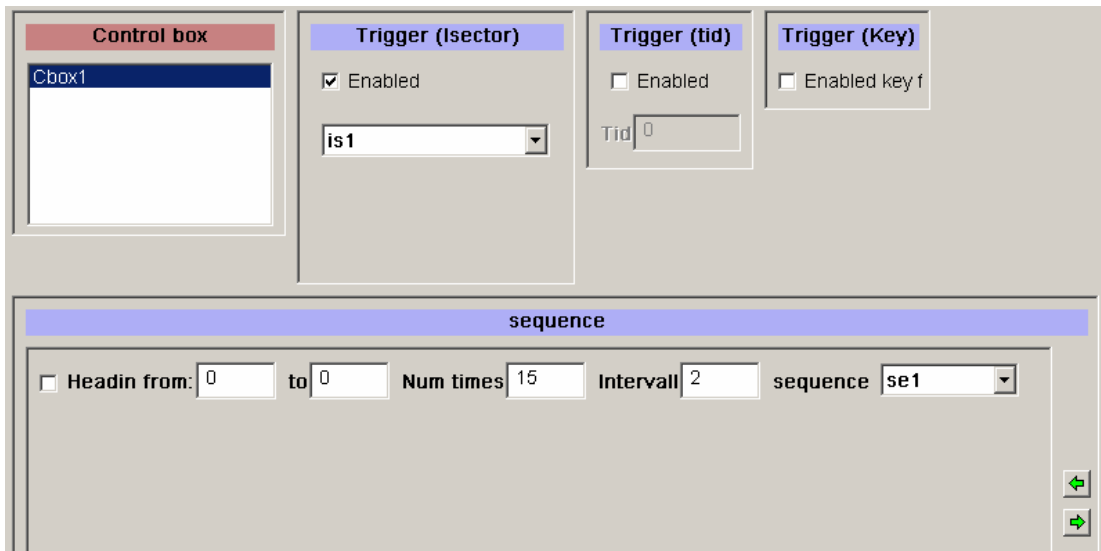


Figure 13 The control box panel.

From this panel the sequences are triggered. The trigger is currently an isector which is a Vega function, a time, or a key on the keyboard. An isector is a Vega method to perform intersection tests between a target and the volume implied by the target object given by the dispenser instance. The intersection test is in the line of sight with a given range specified in the isector class (the isector can for instance trigger on a distance between the platform carrying the countermeasure and the missile). The trigger can be any combination of the three

methods and they correspond to the warning systems range and time to detect the launch the missile or a human in the loop (HIL).

4 DIRCM

A model of a DIRCM system has been developed for use in the real time environment Vega/SensorVision [2]. This model is in part based on a previous OPTSIM model [5]. However, in the OPTSIM model only the laser and the laser control were implemented. In the Vega/SensorVision model there is also an embryo for a warner, i.e. the laser can be triggered based on time from simulation start or based on the distance to the missile. The laser can also be triggered manually. This chapter will describe the Vega/SensorVision DIRCM model.

4.1 Application Interface

The DIRCM module (*EwCmLaser*) includes an API to C/C++ language callable routines for defining *EwCmLaser* effects within Vega. In order to build Vega applications using *EwCmLaser*, an application must include *EwCmLaserModule.h*, and link with the *EwCmLaser* library. For Windows users, *psEwCmLaserS.lib* should be used for static executables and *psEwCmLaser.lib* (*psEwCmLaser.dll*) should be used for dynamic executables.

4.2 Initializing the EwCmLaser module

If Vega is to recognize the *EwCmLaser* classes, an application must initialize the module after calling *vgInitSys* to initialize Vega. The function *InitEwCmLaser* initializes module classes and the Special Effects module for use with Vega. A *VG_FAILURE* is returned if an error has occurred, and *VG_SUCCESS* is returned if not. Once this initialization has been done, an ADF containing *EwCmLaser* class instances can be parsed by sending the ADF to *vgDefineSys*. If the DIRCM module is to be used together with SensorVision then SensorVision has to be initialized and the function *TransmitterUseSV(ewTransmitter* pTransmitter, BOOL bUseSV)* for the transmitters that use an IR laser has to be called with *bUseSV* set to TRUE.

```

#include <vg.h>
#include <vgfx.h>
#include <vgsv.h> //if SensorVision

#include "ewCmLaserModule.h"

main()
{
    //initialize Vega
    vgInitSys();

    //initialize SensorVision (optional)
    vgInitSV();

    //initialize EwCmLaser
    InitEwCmLaser();

    //read ADF
    vgDefineSys( "Adf file name" );

    //configure Vega
    vgConfigSys();

    //if SensorVision
    int nTransmitters = GetNumTransmitters();
    for (
        int nTransmitterIndex=0;
        nTransmitterIndex<nTransmitters;
        nTransmitterIndex++
    )
    {
        TransmitterUseSV(
            GetTransmitter(nTransmitterIndex), TRUE
        );
    }

    //start real-time loop
    while( 1 )
    {
        vgSyncFrame();
        vgFrame();

        /* application specific code */
    }
}

```

4.3 EwCmLaser modules

The Vega/SensorVision module for modeling DIRCM systems has been divided into three types of classes (panels): *Laser trigger*, *Transmitter*, and *Laser*. The *Laser trigger* will activate the *Transmitter* according to a criterion (distance, time from simulation start, or a decision by the user). Each *Laser trigger* can control one or several *Transmitters*. The *Transmitter* will determine the position of the *Laser* and where the *Laser* will be directed. Each *Transmitter* can only have one *Laser*. The different DIRCM classes and how they are related are displayed in figure 14.

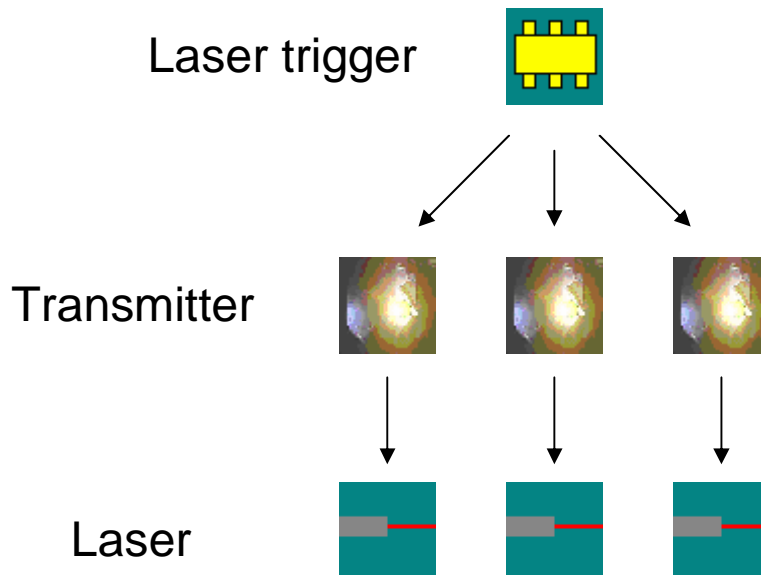


Figure 14 EwCmLaser classes created for use in Vega/SensorVision.

4.4 Laser trigger

A laser creates a very narrow beam and can therefore not be used as a pre-emptive countermeasure without knowledge about where the threat is located. Hence, a laser jammer has to be coupled to a warning system that can trigger the laser and give a rough direction to the threat. A fine tracker then has to track the threat and give a more accurate direction (see chapter 4.5). The warning can be from a missile approach warner (MAW) that detects the flame from the jet engine or warm details on the missile body. Warning could also come from a laser warner. This later type of warner can be triggered when for instance the shooter measures the distance to the target or when an approaching missile is a beam rider guided by a laser beam. After the warning system has triggered the DIRCM system there will be some time before the laser can jam the missile or the shooter. In case of a warning for a missile launch or for a shooter measuring the distance to the target, the time from system alert to jam is a parameter that can be used to trigger the DIRCM. If the missile is detected (exhaust plume, warm parts on the missile, the laser beam of a beam rider) the distance to the missile could be the critical parameter determining whether the missile is detected or not. Presently the warning system is modeled by using one of those two parameters (time or distance).

Figure 15 shows the user interface in LynX to the laser trigger class developed for simulations within the Vega/SensorVision framework.

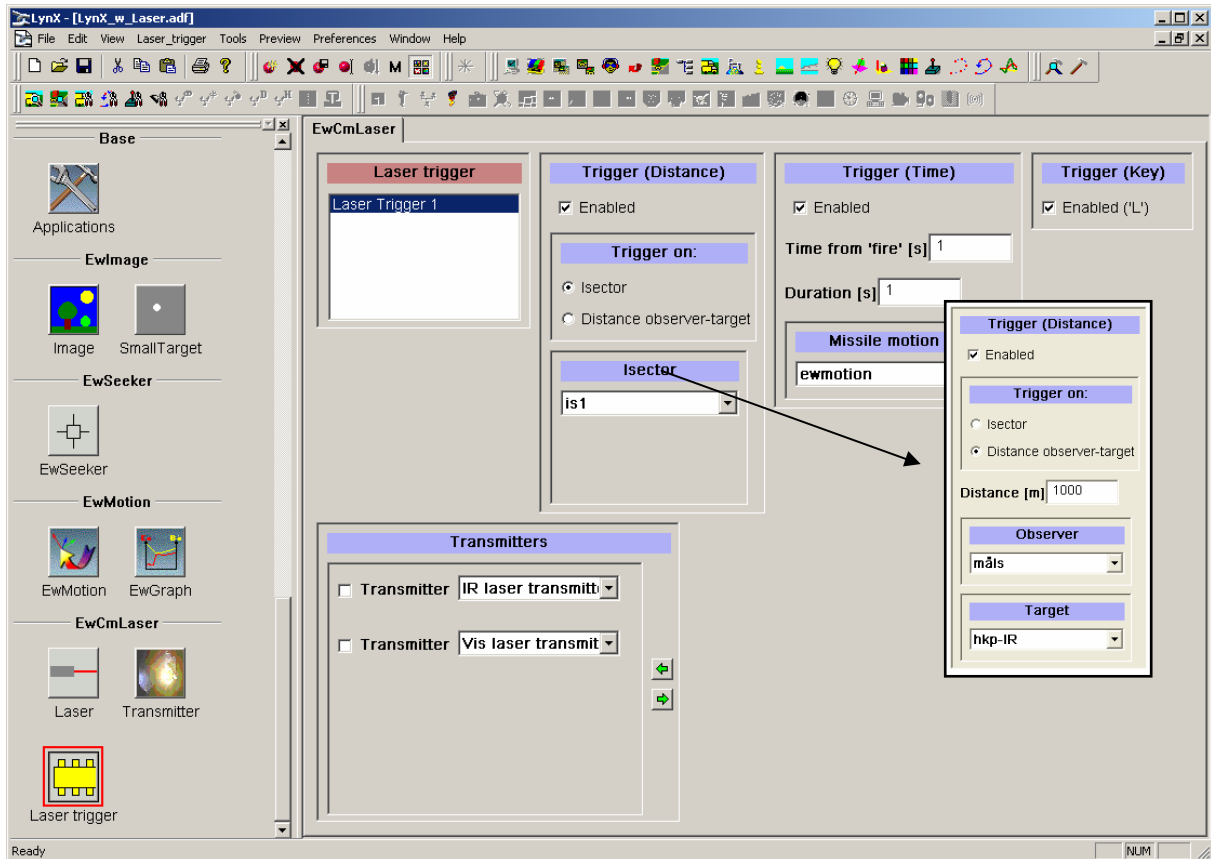


Figure 15 User interface for the Laser trigger class in LynX. More than one trigger can be used if for instance different Transmitters are connected to different triggers or if DIRCM systems are on different platforms. The Transmitters that are controlled by the Laser trigger are set in the Transmitters frame. In this case two Transmitters (Vis laser transmitter and IR laser transmitter) are activated by the Laser trigger “Laser Trigger 1”. The Transmitters can be triggered by a distance (Trigger (Distance)), by a time from simulation start (Trigger (Time)), or by pressing a key on the computer keyboard (Trigger (Key)).

When the *Transmitter* is triggered by a distance, this is controlled by the use of the position of an observer and of a target, or by the use of an *Isector*. An *Isector* is a Vega method to perform intersection tests between a target and the volume implied by a target object. It can trigger on the distance to the surface of an object instead of as in the other case to a point on or in the target. However, the setup of an *Isector* is more complicated than the other method.

4.5 Transmitter

A DIRCM system can have one or several laser transmitters on fixed positions on its platform (*Object*, see figure 17). The *Transmitter* class determines the position of the laser source and direction of the laser beam relative the object.

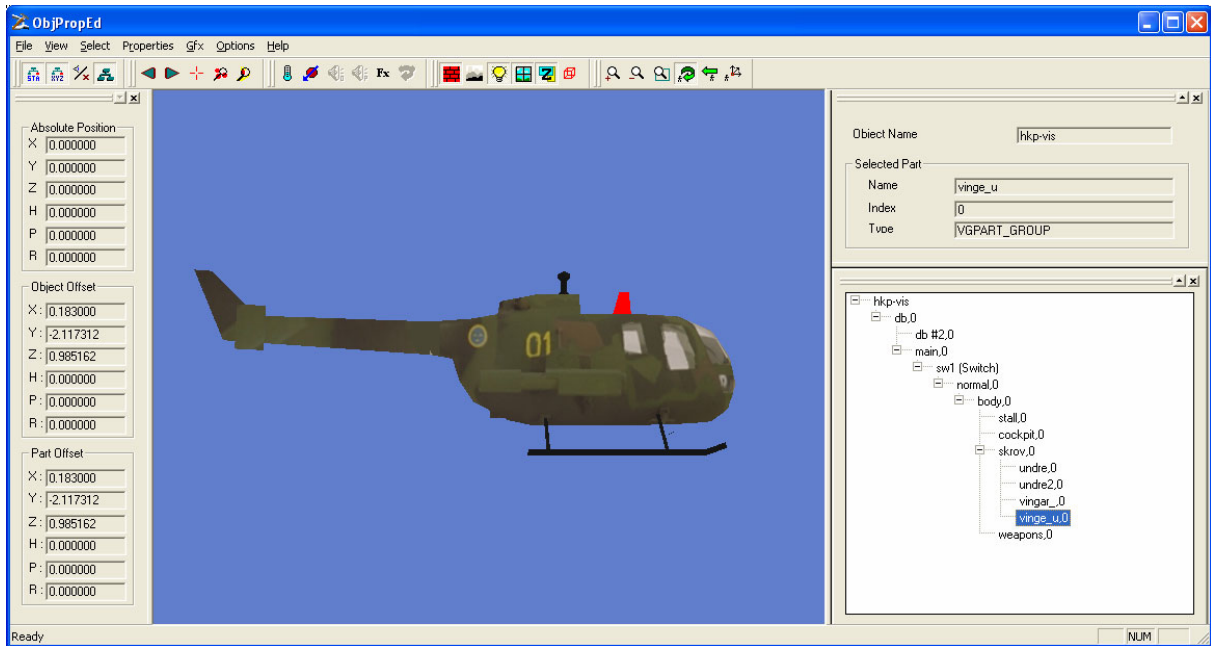


Figure 16 Parts of a helicopter (*hkp-vis*) can be viewed using the Object Property Editor in LynX.

The position of a laser transmitter is set by selecting the object to which the *Transmitter* is attached and it is also possible to select a part (see figure 16) on that object and attach the *Transmitter* to that part. The position of the part with respect to the object is calculated as the average of all vertices of that part. It is also possible to set an offset of the *Transmitter* from the calculated position of the object (or part on the object). Besides the location of the laser the *Transmitter* also determines the direction of the laser beam. This direction can be automatically calculated and always point in the direction of a specific target or set manually (*Heading, Pitch, Roll*). Figure 17 shows the user interface in LynX to the *Transmitter* class.

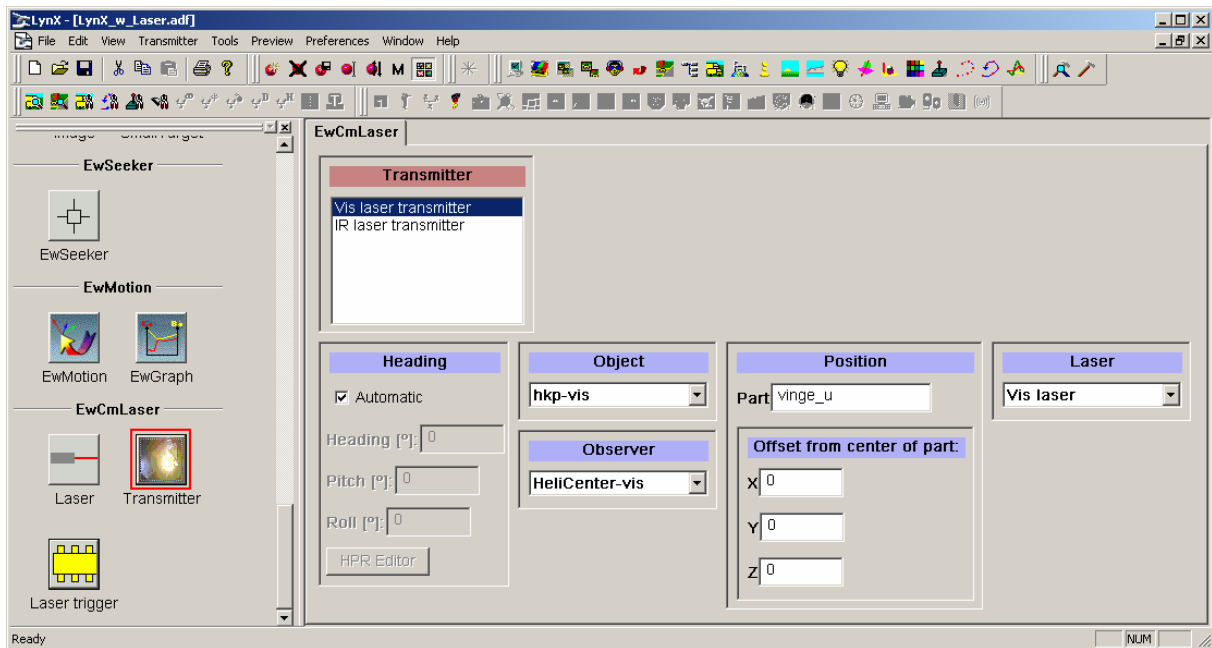


Figure 17 User interface for the Transmitter class in LynX. If there is a need for more than one laser then more than one instance can be used. The direction of the laser beam can be set automatically or manually (controlled by Heading). The position of the Transmitter is set by selecting an object, a part on that object, and an offset with respect to the center of the part. The laser to be used by the Transmitter is selected under Laser.

When the direction of the laser beam is calculated automatically, then the position of the object and an observer is used. The object, in this case, is the object (platform) carrying the DIRCM system and the observer is the target of the laser beam. Both the object and the observer are selected in the user interface to the *Transmitter* class. The positions of the object and the observer are also used when a manual heading is used in order to determine how much, if any, of the laser beam is collected by the observer's optical system (this calculated amount is also affected by the direction of the laser beam, and the orientation of the observer).

4.6 Laser

The *Laser* class is based on the laser jamming module in the OPTSIM model, MAIS [5]. The purpose of this class is to calculate the intensity collected by an observer's optical system and to determine the size of the laser spot seen by this observer. The collected intensity is determined by the distance between the DIRCM system and the observer, the direction of the laser beam, the size of the laser beam and its intensity distribution, turbulence and transmission in the atmosphere, transmission in the optics of the emitting laser, the size of the collecting aperture, the wavelength of the laser, orientation of the observer, and wavelength range of the observer's sensor.

Figure 18 shows the user interface in LynX for the *Laser* class.

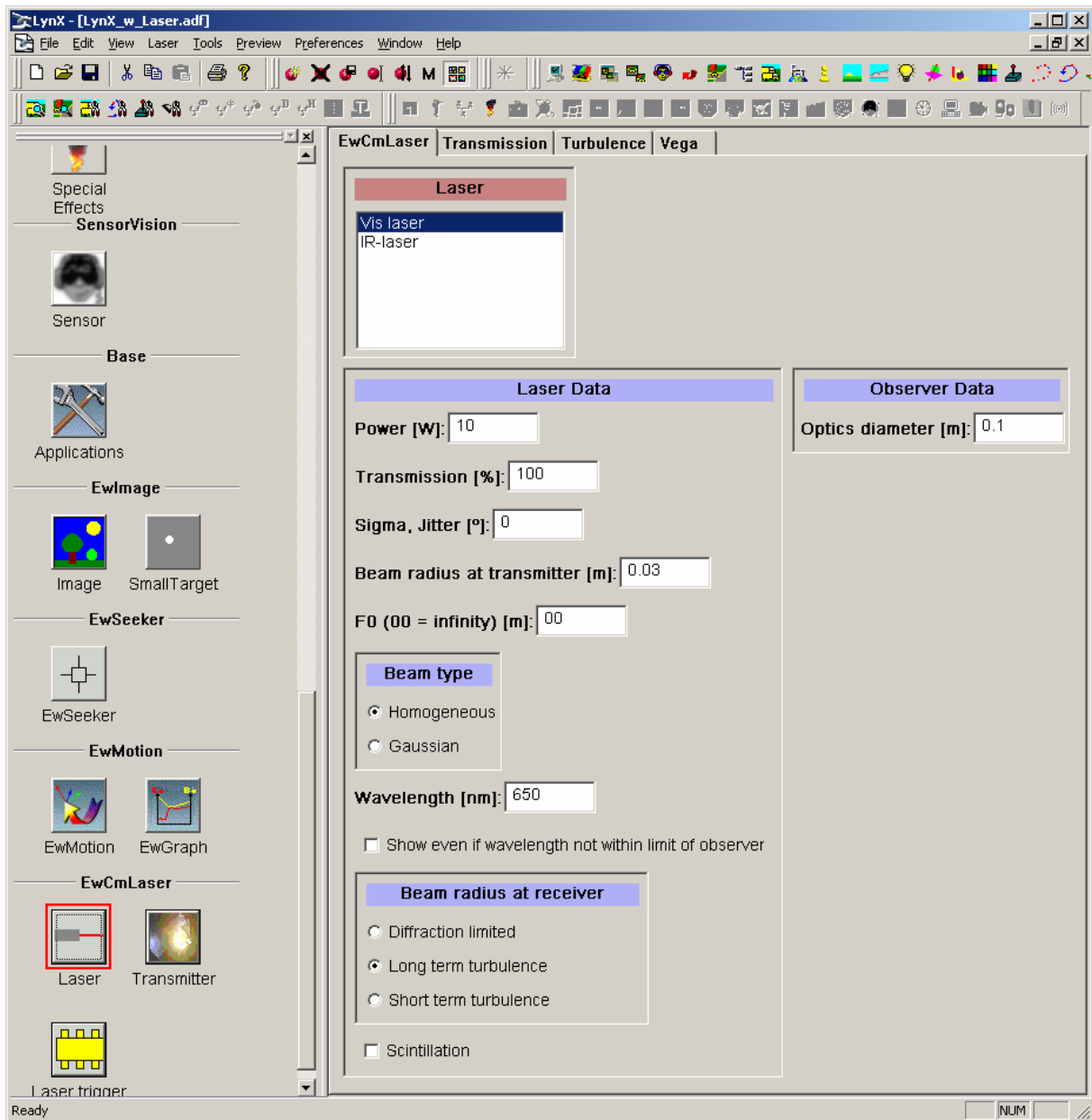


Figure 18 User interface for the Laser class in LynX. More than one laser can be defined. Laser Data is the data describing the laser which is needed to determine the intensity, size and beam quality at the receiver side. Observer Data describes the aperture of the receiver which is needed in order to calculate how much of the laser beam is collected by the receiver.

The transmission and turbulence for the laser wavelength is defined using two separate dialogs; one for transmission and one for turbulence. The user interface for defining the transmission is shown in figure 19.

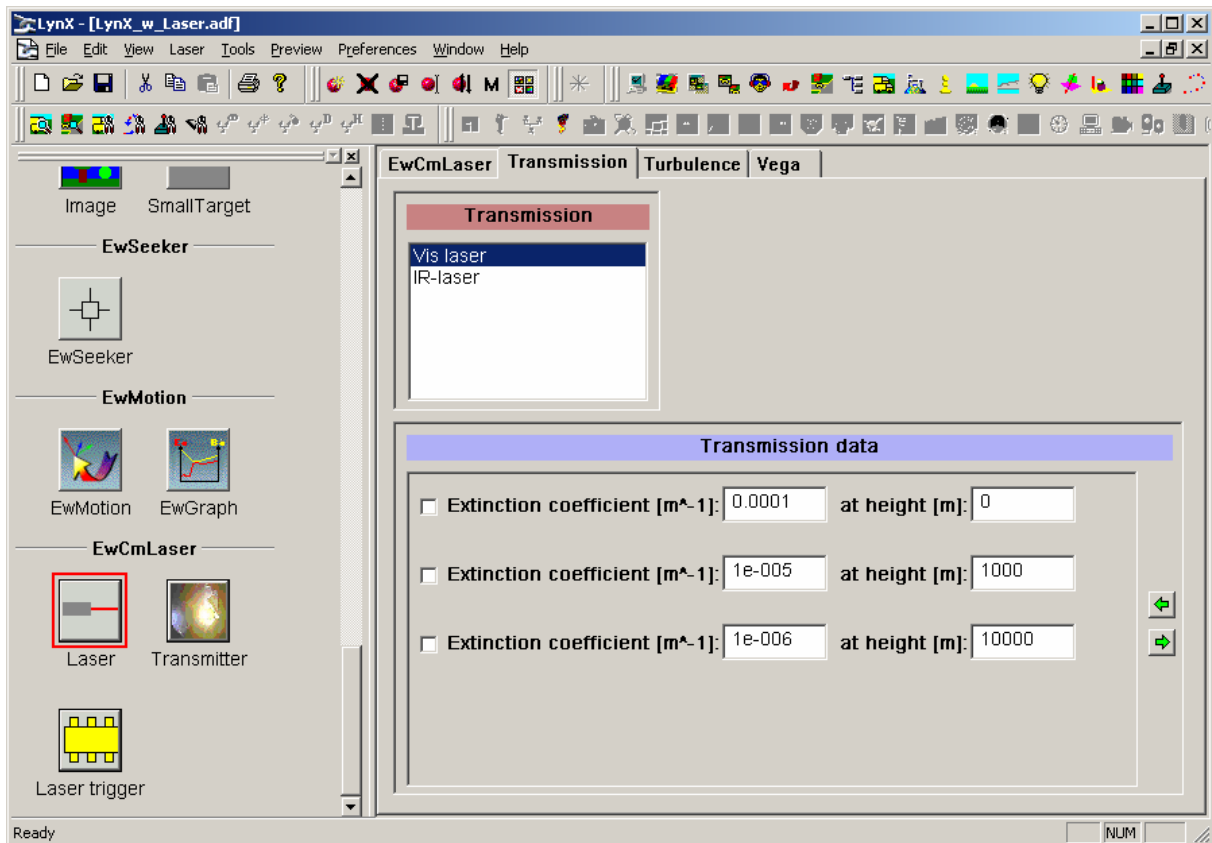


Figure 19 User interface for the input of atmospheric transmission data for a specific laser.

The transmission through the atmosphere for a laser beam at a given time and location depends on the wavelength of the laser and the altitude. The transmission can vary significantly within the wavelength range of a typical sensor and it is therefore not possible to use the same transmission for the laser and the rest of the scene. The strong wavelength dependence makes it necessary to define the transmission for every laser used in the simulation. The transmission is given as extinction coefficients for different heights. When the transmission for the path of the laser beam from the transmitter to the receiver is calculated, the transmission is calculated using a linear approximation for the extinction coefficient, $\alpha(\text{height})$, between the heights used as input to the class. If only one value exists then the extinction coefficient is assumed to be constant. Without any values for the extinction coefficient a constant value of zero is assumed. If the height is outside the range of heights the extinction coefficient which is closest in height is used.

$$\tau = \exp\left(-\int_0^L \alpha(h) ds\right)$$

The effect of atmospheric turbulence is also height and wavelength dependent. Therefore, the dialog for input of turbulence data is very similar to the panel for transmission data. Figure 20 shows the user interface for defining the turbulence.

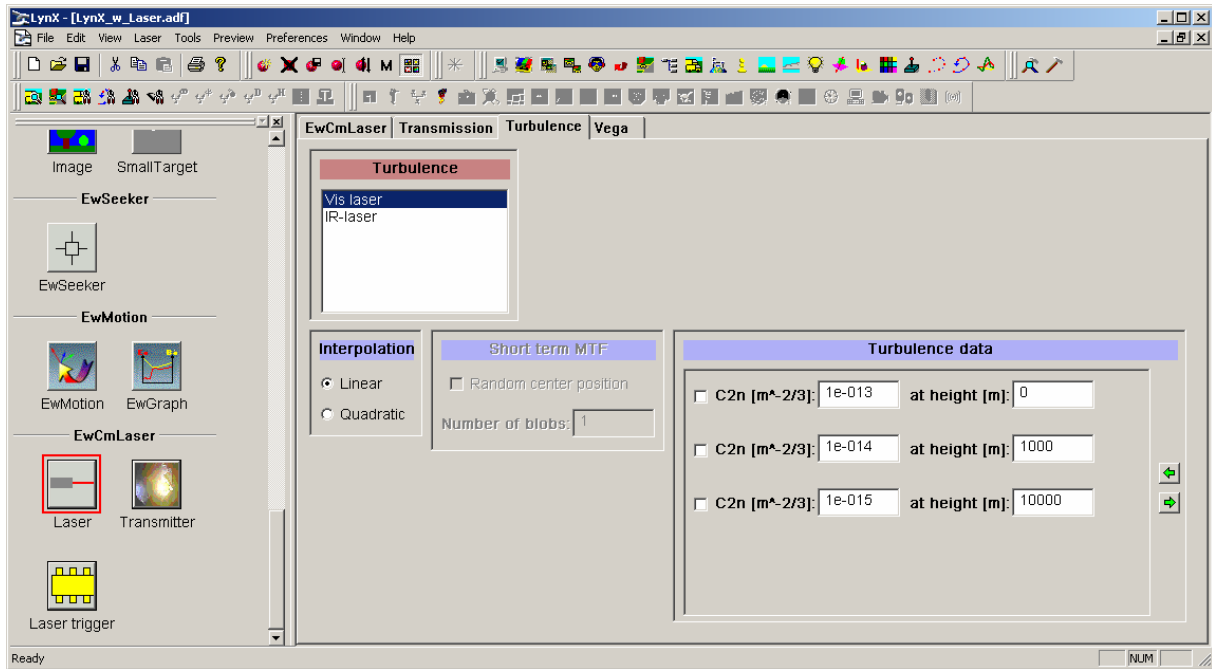


Figure 20 User interface for input of atmospheric turbulence data for a specific laser. The frame labeled Short term MTF is enabled when Beam radius at receiver is set to Short term turbulence in the EwCmLaser panel.

In the Turbulence dialog, the refractive-index structure constant, C_n^2 , is given as input as a function of height. In the executable code C_n^2 might be needed for a different height than the heights provided and in that case linear or quadratic interpolation will be used. Turbulence, defined by C_n^2 , will affect the beam size and beam quality at the receiver. Both these properties will depend on the integration time of the receiving sensor. If the integration time is short with respect to the time constant of the turbulence then the effect of the turbulence is called short term otherwise it is called long term. Short term turbulence gives a narrower beam and a smaller image of the laser spot. However, with a short integration time the image of the laser spot will dance or move around as a function of time. As the integration time increases, this effect will decrease. In the laser model the effect of an increased integration time can be simulated by using several blobs with a random center position. For long enough integration times (many blobs) the effect of turbulence will be the same as for long term turbulence.

When the effect of the laser jammer is created in the image, this can be made using an *analytical* mode or a *real time* mode. The *analytical* mode is identical to the calculation method used in the previous OPTSIM model [5]. In the *real time* mode the intensity of the laser spot is calculated using the same equations but the size and intensity distribution within the created image of the laser spot is created using an approximate method. The *real time* method assumes that the MTF

$$MTF(f) = \exp\left(-3.44 \left[\frac{\lambda f}{r_0}\right]^{5/3}\right) \quad (\text{long term turbulence})$$

$$MTF(f) = \exp\left(-3.44 \left[\frac{\lambda f}{r_0}\right]^{5/3} \left[1 - b \left(\frac{\lambda f}{D_o}\right)^{1/3}\right]\right) \quad (\text{short term turbulence})$$

(λ = wavelength [m], f = spatial frequency [rad^{-1}], r_0 = Fried's coherence diameter [m], $b = 1$ for near field, and 0.5 for far field, D_o is the diameter of the receiver's aperture [m])

can be approximated with a Gaussian MTF which means that the MTF is equal to $1/e$ for frequencies of $\frac{1}{\sqrt{2\pi}\sigma}$, where σ is the radius where the point spread function is $1/e$ times its peak value. The intensity of the laser in the image is calculated as a radiance and then used directly (analytical mode) or transformed to a temperature (real-time mode). In a visual image the radiance is not known and therefore the user has to give a radiance value for white pixels in the image (*Max radiance in image [W/sr/m²]*). In *analytical* mode the laser spot can be observed even if its size is very small. The image of the laser spot in the *real time* mode will be created by using a disk with a size in meter at the transmitter that will result in an image with the correct size (according to the approximation described above). However, if the calculated beam size is very small it will be difficult to see the laser spot and it is therefore possible to set a minimum size for the laser spot in pixels. If the calculated size is smaller than the minimum size the spot size will be increased and the intensity of the laser spot will be decreased so that the total intensity in the laser spot is constant. The parameters that control how the image of the laser spot is created are in the *Vega* dialog in the *Laser* panel. The user interface for these parameters is shown in figure 21.

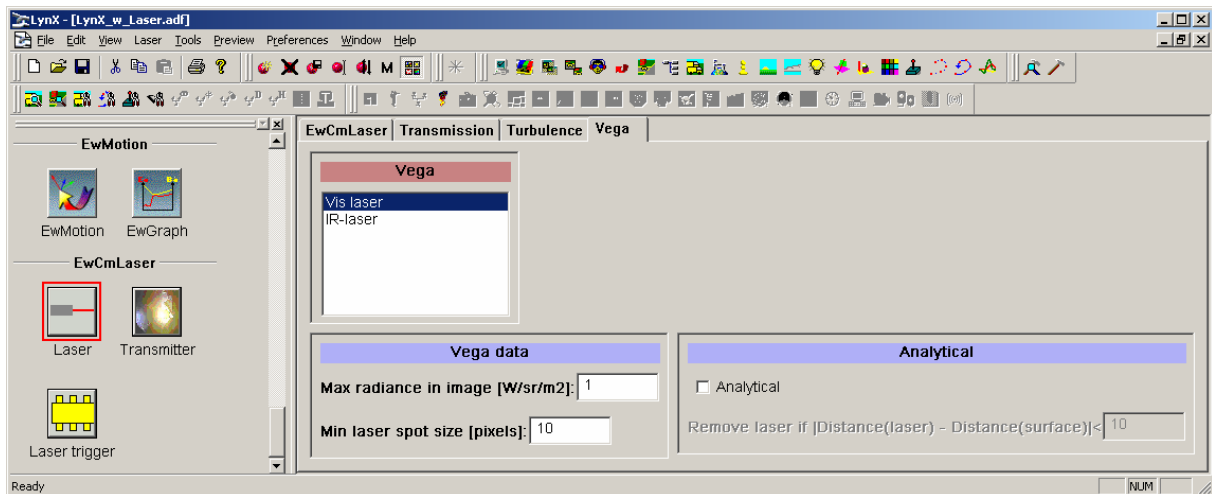


Figure 21 User interface for input of parameters that control how an image of a laser transmitter is created.

5 Extracting Images

The Vega *EwImage* module provides an easy way to use images generated by Vega to do deeper image processing in in-house modules. The module includes interfaces for both Visual and IR images by using Vega and the IR plug-in SensorVision.

The icon panel of the *EwImage* (Electronic warfare Image) module currently consists of 2 Classes (panels) Image and SmallTarget as shown in figure 22.

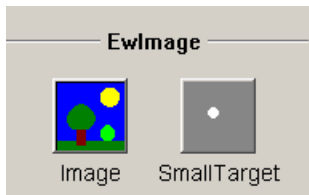


Figure 22 Icon panel from the *EwImage* module.

5.1 Application Interface

The *EwImage* module includes an API to C/C++ language callable routines for using *EwImage* functions within Vega. In order to build Vega *EwImage* applications, an application must include the file *EwImage.h*, and link with the *EwImage* library. For Windows users, *psEwImageS.lib* should be used for static executables and *psEwImage.lib* (*psEwImage.dll*) should be used for dynamic executables.

5.2 Initializing the EwImage Module

If Vega is to recognize the *EwImage* classes, an application must initialize the module after calling *vgInitSys* to initialize Vega. The function *InitEwImage* initializes module classes for use with Vega. Two parameters are to be sent with the initialization, the address of *m_bUseFrame* and *m_bUseFrameImage*. *m_bUseFrame* is used to specify which frames to use, and *m_bUseFrameImage* should be TRUE the frame just before *m_bUseFrame*. It is the pipelining of Vega that causes this management of frames. Otherwise the frame that Vega is about to render and the actual frame seen are separated by one or more frames. This can cause serious problems if the image processing results is fed to some sort of movement control. A VG_FAILURE is returned if an error has occurred, and VG_SUCCESS is returned if not. Once this initialization has been done, an ADF containing *EwImage* class instances can be parsed by sending the ADF to *vgDefineSys*.

```

#include <vg.h>
#include <vgperf.h>
#include <pf.h>
#include <prmath.h>
#include "vgwin.h"

#include "ewImage.h"

/*
=====
    Main application entry point
=====
*/
void main( int argc, char *argv[] )
{
    vgWindow*   window;
    float       m_frameRate;
    float       m_frameTime;
    float       m_frameDelta;
    BOOL        m_bUseFrame = FALSE;
    BOOL        m_bUseFrameImage = FALSE;
    int         m_nSkipFrames;
    int         m_nCountFrames,i;

// init, define, and config the system

    vgInitSys();           // initialize Vega
    vgInitSV();           // initialize SensorVision (optional)
    vgInitSW();           // initialize SensorWorks (optional)

// ----- Add init functions for implemented modules -----

    InitEwImage(&m_bUseFrame,&m_bUseFrameImage); //init the image module

// -----

    vgDefineSys( argv[1] ); // read in the ADF

    vgConfigSys();         // configure Vega

// ---- check if frames must be skipped -----
    m_nSkipFrames = (int)vgGetProp(vgGetSys(), VGSYS_NUMSTAGES);
    m_nCountFrames = m_nSkipFrames-1;

    m_frameRate = m_nSkipFrames*vgGetProp(vgGetSys(), VGSYS_FRAMERATE);
    m_frameTime = pfGetFrameTimeStamp();
    m_frameDelta = 1.0f / m_frameRate;

// ----- run 2 frames to get all systems ready -----
    for (i = 0; i < 2; i++)
    {
        vgSyncFrame ();
        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        vgFrame();
    }
}

```



```

// ----- the real-time loop -----
if (m_nSkipFrames > 1) // Always when using SensorVision
{
    while ( 1 )
    {
        vgSyncFrame ();

        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        m_bUseFrame = !(m_nCountFrames % m_nSkipFrames);
        m_bUseFrameImage =
            ((m_nCountFrames++ % m_nSkipFrames) == m_nSkipFrames-1);
        vgFrame ();
    }
}
else // not possible with sensorvision
{
    m_bUseFrame = TRUE;
    m_bUseFrameImage = TRUE;
    while ( 1 )
    {
        vgSyncFrame ();

        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        vgFrame ();
        /* application specific code */
    }
}
}

```

5.3 The EwImage module

The LynX *EwImage* panels contain widgets for setting the parameters that define the image. If you are unfamiliar with how to use the LynX interface, consult the Vega LynX User's Guide before proceeding [4].

The *EwImage* module currently consists of 2 Classes (panels) *Image* and *SmallTarget*.

5.4 EwImage

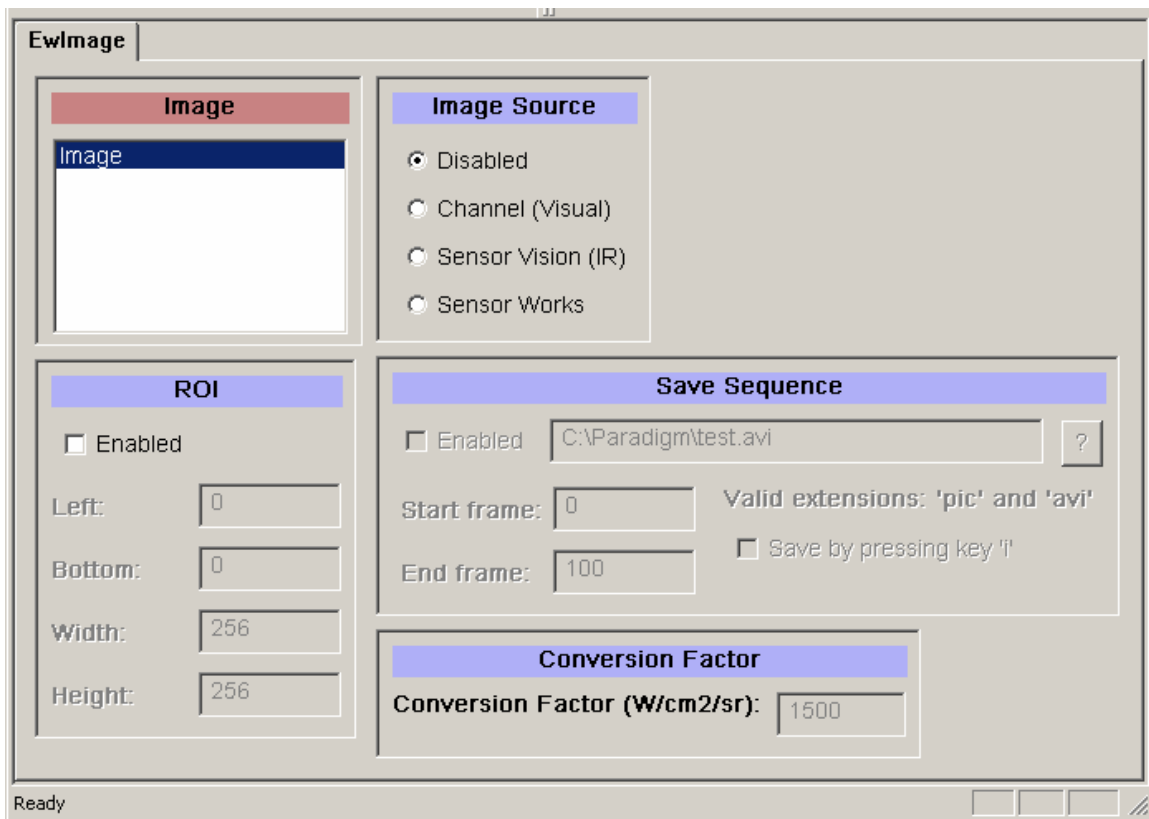


Figure 23 The Image panel.

To use the *Image* module connect an instance of the class to a channel, SensorVision or SensorWorks instance.

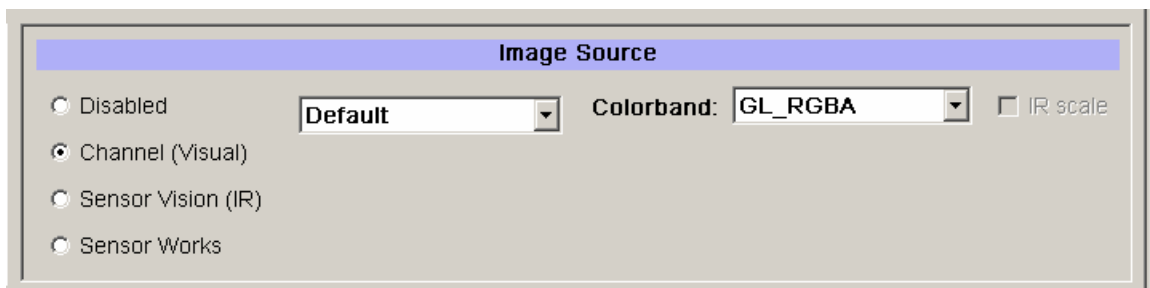


Figure 24 The Image Source panel.

Table 1 Channel Options

Parameter	Description	Type of array (image)
Colorband		
GL_RGBA	All bands	COLORREF
GL_LUMINANCE	luminance image	float
GL_RED	red band image	float
GL_GREEN	green band image	float
GL_BLUE	blue band image	float
IR scale (not an option for mode GL_RGBA)	Converts a colorband to an intensity image by, level by level replace the values from a scale of 1 to 255, with a user defined scale with two different slopes. See figure 25. Used when creating an IR image without SensorVision.	

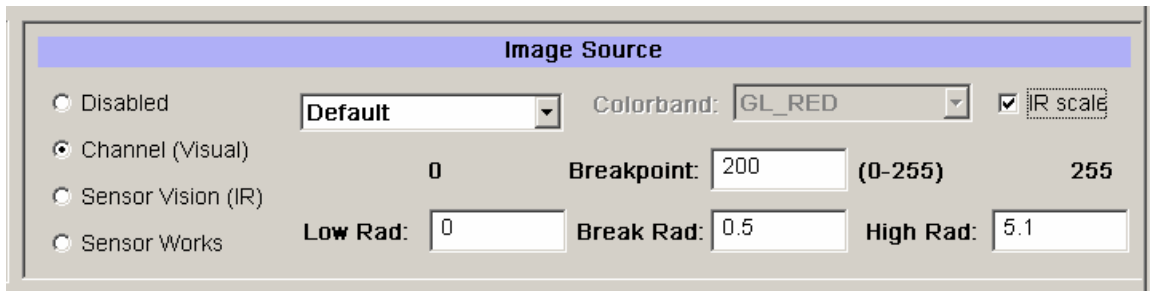


Figure 25 The Image Source panel with IR scale checked.

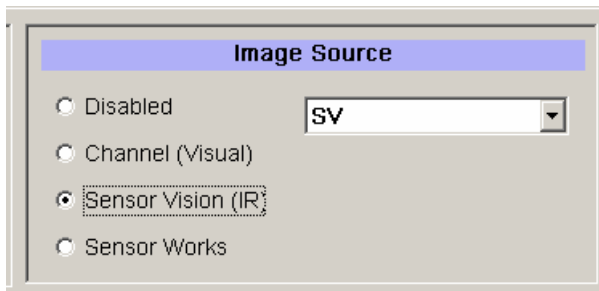


Figure 26 The Image Source panel. SensorVision and SensorWorks are checked.

When SensorVision or SensorWorks is checked there is only an instance to set. See figure 26.

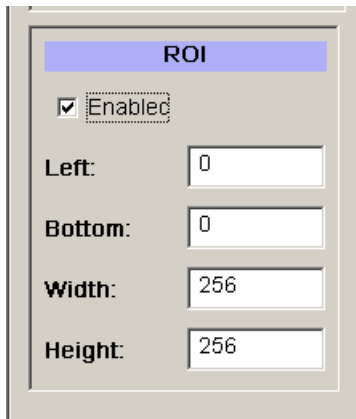


Figure 27 the ROI panel.

In the ROI panel, figure 27, the image size can be limited and specified by the parameters. If the movie format avi is used, ROI should also be used, because the size is determined before the sequence is started. (Some compression mode requires the size to be a multiple of 2).

5.5 SmallTarget

The *SmallTarget* module was created to make it possible to process images with sub-pixel targets, which would disappear in *SensorVision*. The module creates an enhanced image of the target. This image is then used to compute the intensity for the pixels which shall be replaced. The replacement is done in the array produced in the *Image* module; it will not be visible on the screen.

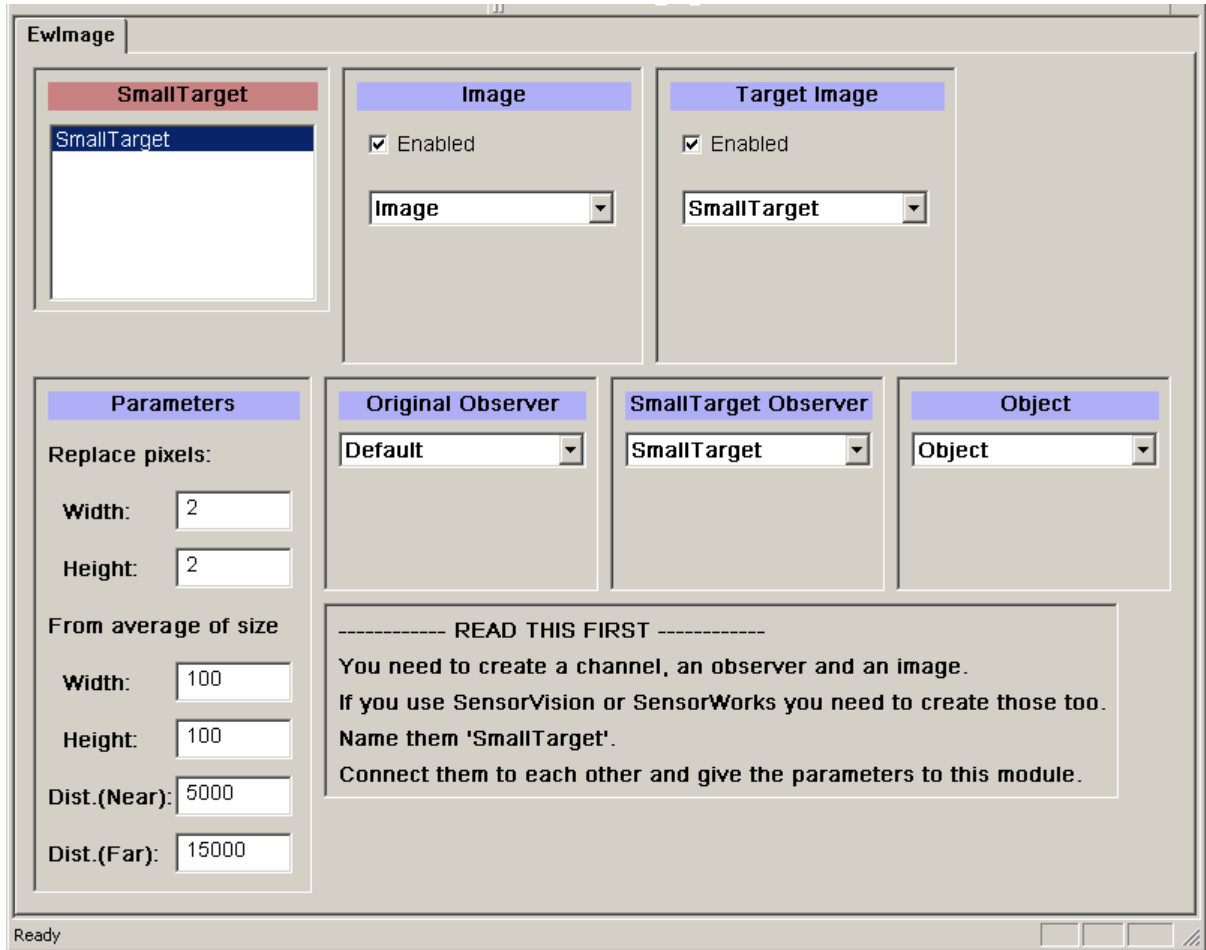


Figure 28 The SmallTarget panel.

Table 2 Description of the parameters:

Parameter	Description
Image	The original image no enhancement.
TargetImage	An image which will have the enhanced image of the target (has to be created and connected to this module).
Original Observer	The observer of the Image .
SmallTarget Observer	The observer of the target.
Object	The target.
Parameters:	
Replace pixels	Specifies the number of pixels replaced in the Image .
From average of size	The size of the TargetImage which is used to calculate the average intensities of the replacement pixels.
Dist.(Near) and Dist.(Far)	VEGA parameters which defines the near and far plane distances of the viewing frustum.

6 Target seekers/trackers

The Vega *EwSeeker* module provides an easy to use library of real-time seekers suitable for inclusion in a Vega application. The module includes interfaces for different types of seekers. It is possible to use both visual and IR images by using Vega and the IR plugin SensorVision.

A module consists of a collection of classes each represented in LynX by an icon panel. The *EwSeeker* (Electronic warfare Seeker) module currently consists of 1 class (panel) *EwSeeker* as shown in figure 29.

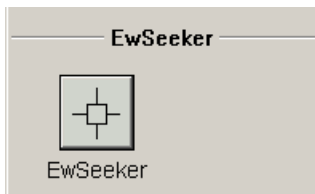


Figure 29 Icon panel from the *EwSeeker* module.

6.1 Application Interface

The *EwSeeker* module includes an API to C/C++ language callable routines for using *EwSeeker* functions within Vega. In order to build Vega *EwSeeker* applications, an application must include the file *EwSeeker.h*, and link with the *EwSeeker* library. For Windows users, *psEwSeekerS.lib* should be used for static executables and *psEwSeeker.lib* (*psEwSeeker.dll*) should be used for dynamic executables.

6.2 Initializing the EwSeeker Module

If Vega is to recognize the *EwSeeker* class, an application must initialize the module after calling *vgInitSys* to initialize Vega. The function *InitEwSeeker* initializes the module class for use with Vega. One parameter is to be sent with the initialization, the address of *m_bUseFrame*. *m_bUseFrame* is used to specify which frames to use. It is the pipelining of Vega that causes this management of frames. Otherwise the frame that Vega is about to render and the actual frame seen are separated by one or more frames. This can cause serious problems if the image processing results is fed to some sort of movement control. A *VG_FAILURE* is returned if an error has occurred, and *VG_SUCCESS* is returned if not. Once this initialization has been done, an ADF containing *EwSeeker* class instances can be parsed by sending the ADF to *vgDefineSys*.

```

#include <vg.h>
#include <vgperf.h>
#include <pf.h>
#include <prmath.h>
#include "vgwin.h"

#include "ewCm.h"
#include "ewImage.h"
#include "ewSeeker.h"
/*
=====
    Main application entry point
=====
*/
void main( int argc, char *argv[] )
{
    vgWindow*   window;
    float       m_frameRate;
    float       m_frameTime;
    float       m_frameDelta;
    BOOL        m_bUseFrame = FALSE;
    BOOL        m_bUseFrameImage = FALSE;
    int         m_nSkipFrames;
    int         m_nCountFrames, i;

// init, define, and config the system

    vgInitSys();           // initialize Vega

    InitEWCM();           // initialize countermeasure module

    vgInitSV();           // initialize SensorVision
    vgInitSW();           // initialize SensorWorks

// ----- Add init functions for implemented modules -----

    InitEwImage(&m_bUseFrame, &m_bUseFrameImage); //init the image module
    InitEwSeeker(&m_bUseFrame); //init the seeker module
// -----

    vgDefineSys( argv[1] ); // read in the ADF

    vgConfigSys();        // configure Vega

// ---- check if frames must be skipped -----
    m_nSkipFrames = (int)vgGetProp(vgGetSys(), VGSYS_NUMSTAGES);
    m_nCountFrames = m_nSkipFrames-1;

    m_frameRate = m_nSkipFrames*vgGetProp(vgGetSys(), VGSYS_FRAMERATE);
    m_frameTime = pfGetFrameTimeStamp();
    m_frameDelta = 1.0f / m_frameRate;

// ----- run 2 frames to get all systems ready -----
for (i = 0; i < 2; i++)
{
    vgSyncFrame ();
    pfFrameTimeStamp( m_frameTime );
    m_frameTime += m_frameDelta;
    vgFrame();
}

```

```

// ----- the real-time loop -----
if (m_nSkipFrames > 1) // Always when using SensorVision
{
    while ( 1 )
    {
        vgSyncFrame ();

        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        m_bUseFrame = !(m_nCountFrames % m_nSkipFrames);
        m_bUseFrameImage =
            ((m_nCountFrames++ % m_nSkipFrames) == m_nSkipFrames-1);
        vgFrame ();
    }
}
else // not possible with sensorvision
{
    m_bUseFrame = TRUE;
    m_bUseFrameImage = TRUE;
    while ( 1 )
    {
        vgSyncFrame ();

        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        vgFrame ();
        /* application specific code */
    }
}
}

```


6.3 EwSeeker

The seeker module which has been adapted to the VEGA/SensorVision environment contains several different seeker algorithms. Today these are *Correlation* [6], *Centroid* [7] and *Reticle* [8]. All of these models has an origin in standalone programs and has been moulded to fit in the VEGA/SensorVision environment. The missile is armed with the key 'a' (arm), to lock the missile point at the target with the mouse and launch the missile by pressing the 'f' button (fire).(see figure 30).

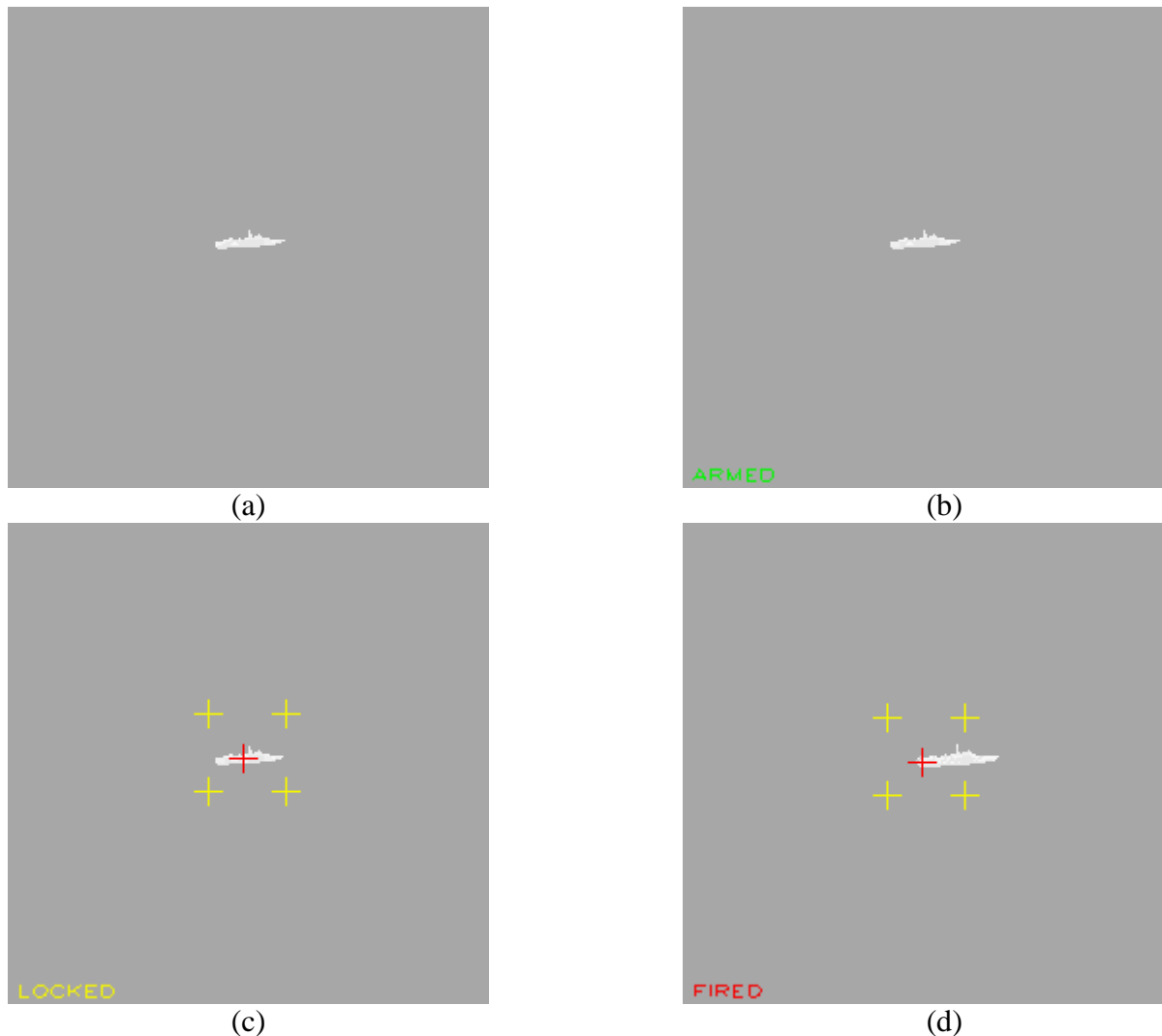


Figure 30 How the missile status is presented to the user (a) shows the seeker image before the missile is armed, (b) shows the image after the missile has been armed (now it is possible to point at the target), (c) the target has been pointed out by the user, (d) the missile has been launched.

The GUI for the *Correlation* seeker is shown in the Figure 31. An instance is created and is shown in the box beneath *EwSeeker*. An image from the *Image* module must be set and the seeker type must be picked. Each seeker has a set of parameters which can be changed; the parameters are the most relevant for each seeker and the default values are set at the beginning.

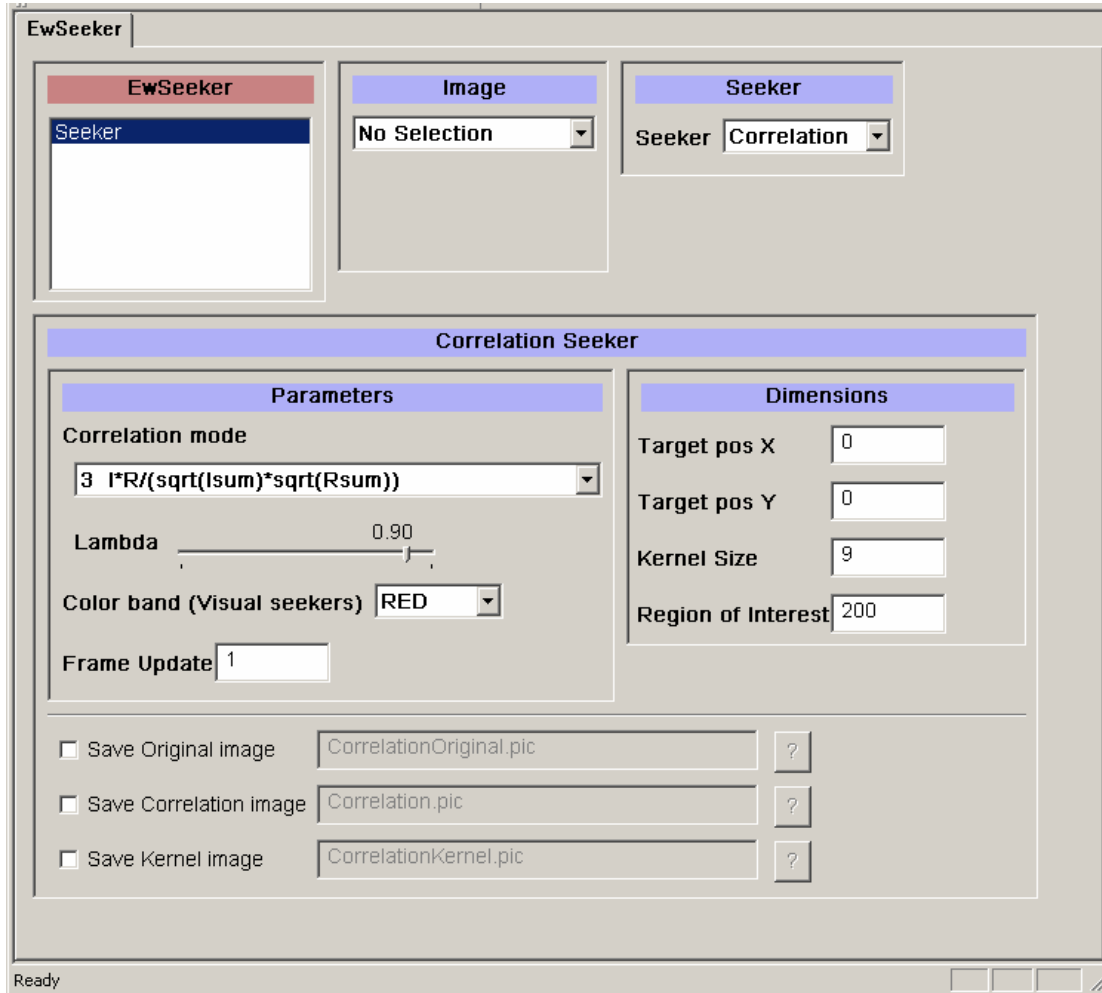


Figure 31 The Vega panel in the EwSeeker module. This example shows the Correlation parameters.

Table 3 Correlation parameters:

Parameter	Description
Correlation mode	What correlation algorithm to use (see reference [6]).
Lambda	Specify how the reference image (correlation kernel) shall be updated by mixing the reference image from the new target position and the old reference mage (0 = only the new image, 1 = only the old image).
Color band (Visual seekers)	Specifies the color (red, green or blue) which shall be used by the correlation on visual images.
Frame Update	Specifies how often the reference image shall be updated.
Dimensions	
Target pos XY	The target position (has no function today but if the program extends to handling mass simulations it will).
Kernel size	Specifies the correlation kernel size.
Region of Interest	Specifies the size of the search area in the image.
Save Original image Save Correlation image Save Kernel image	select images that can be saved in an image sequence. The three images which can be saved are the unprocessed original image, the correlation image and the kernel.

The GUI for the *Centroid* seeker is shown in the Figure 32

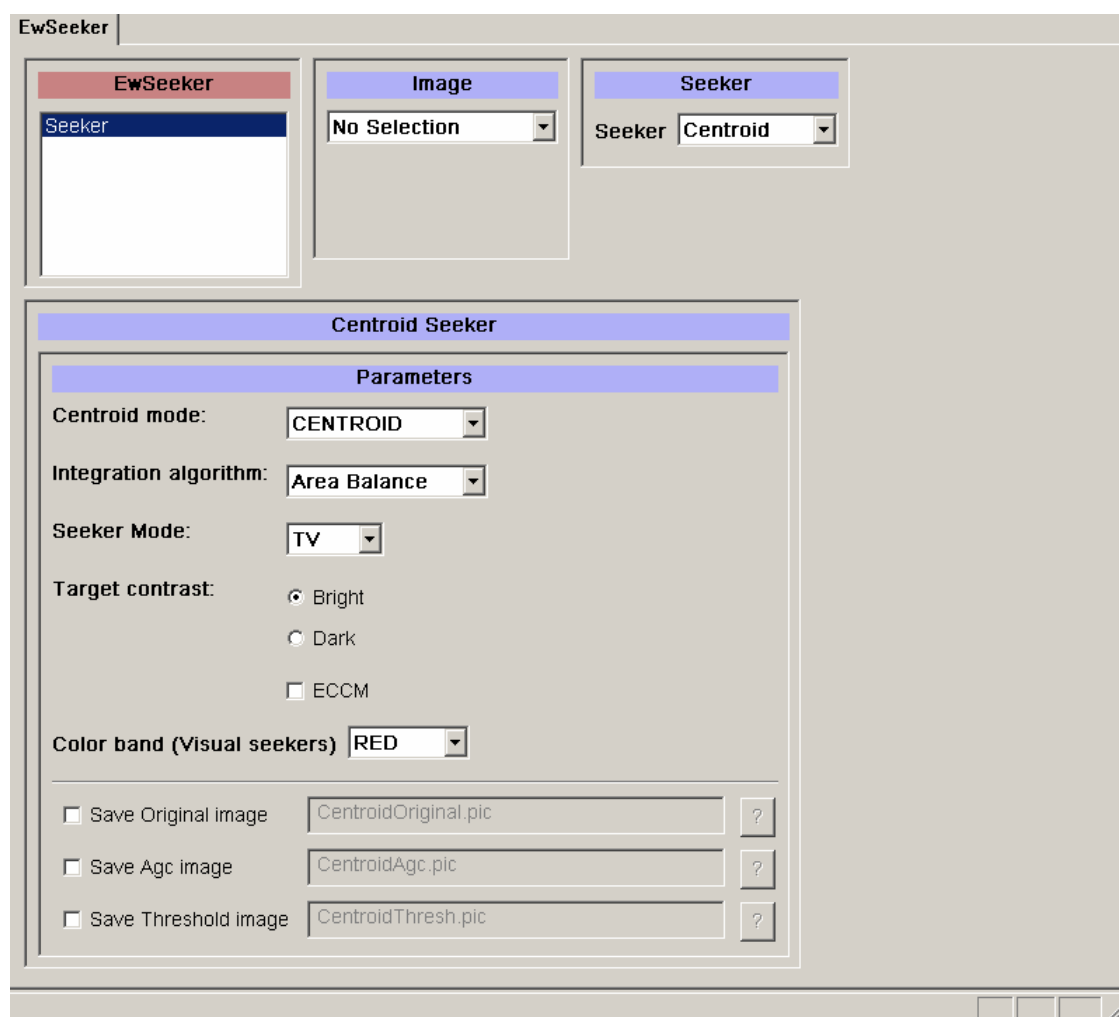


Figure 32 The Vega panels in the EwSeeker module. This example shows the Centroid parameters.

Table 4 Centroid parameters (for extensive explanations see reference [7])

Parameter	Description
Centroid mode	Specifies the start mode the seeker has. (It is possible to force the seeker to start with a correlation mode but that is not recommended).
Integration algorithm	Specifies how the target position is calculated after the extension of the target is discovered. (Area Balance, Center of mass and Intensity).
Seeker mode	Specifies how the target area and background areas are defined (TV, Land or Ship).
Target contrast	Specifies the target to be bright on dark background or dark on bright background.
ECCM	A simple Counter-CounterMeasure.
Color band (Visual seekers)	specifies the color (red, green or blue) which shall be used by the correlation on visual images.
Save Original image Save Agc image Save Threshold image	Select images processed in the Centroid seeker, so that they are saved. The three images which can be saved are the unprocessed original image, the agc (automatic gain control) image and the threshold image.

The GUI for the *Reticle* seeker is shown in the Figure 33

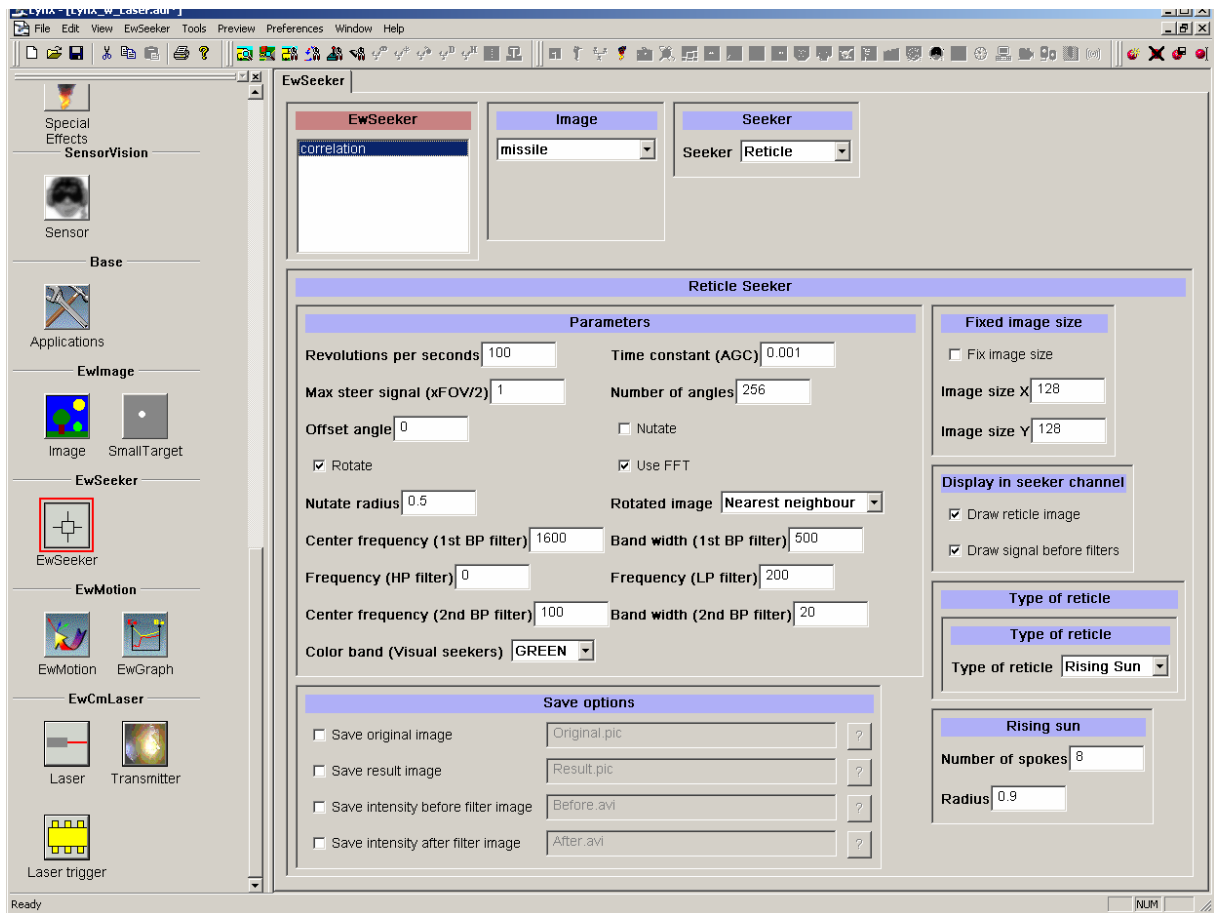


Figure 33 The Vega panels in the EwSeeker module. This example shows the Reticle parameters.


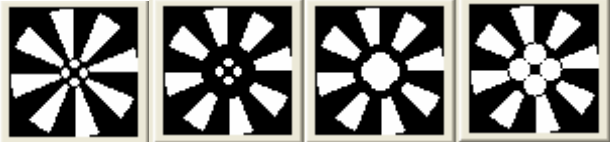

Table 5 Reticle parameters (for more details see reference [8] and [9]):

Parameter	Description
Revolutions per second	Number of revolutions per second of the reticle.
Time constant	Many reticle seekers can only give an estimate of the radial position of a target in an image (angular position is more exact). To automatically adjust the sensitivity to radial position an automatic gain control (AGC) can be used with a time constant.
Max steer signal	A reticle seeker will give the angular position and an estimate of the radial position of the target. This means that the estimated radial position can be too large and this can result in a loss of track if the estimated radial position is sent to the missile motion model (see chapter 7). Therefore, the radial position sent to the missile motion model can be constrained to a maximum value.
Number of angles	The signal from the reticle in the model is obtained by image processing, using an image of the scene in the seeker's field of view and an image of the reticle. The reticle signal as a function of reticle angle is then read from the processed image. The signal is a discrete function with a limited number of angles which is controlled by this parameter.
Offset angle	To find the target position from the reticle signal the signal has to be processed. This process does not consider how the reticle is oriented and therefore an offset might have to be added to the calculated angular position of the target. Presently, the offset angle should always be zero since the image of the reticle is created with the correct orientation. In the future, and in the original stand-alone program on which the <i>Reticle</i> seeker model is based, it is possible to load an image of a reticle and use that image in the seeker model.
Rotate Nutate	A reticle seeker can have a rotating (spin scan) or nutating (conical scan) reticle. Select either Rotate or Nutate.

Table 5 continued

Parameter	Description
Use FFT	In the <i>Reticle</i> seeker model two methods to obtain the reticle signal as a function of angle exist. One method is to rotate or nutate the reticle over the background image and for every angle multiply the intensities of the pixels in the background image with the corresponding pixels of the reticle image, sum the intensities of all pixels in the resulting image. An alternative method is to use FFT (fast Fourier transforms) to speed up the process.
Nutate radius	If the seeker is a conical scan reticle then the image of an object in the center of the seeker's field of view will move along a circle with a radius from the centre of the reticle equal to this parameter.
Rotated image	For a spinning reticle when the FFT method is not used the rotation of the reticle can use a nearest neighbour approximation or interpolate.
Center frequency (1st BP filter) Band width (1st) Frequency (HP filter) Frequency (LP filter) Center frequency (2nd) Band width (2nd)	Signal processing to obtain the target's position is made using a series of band pass (BP), low pass (LP), and high pass (HP) filters.
Color band (Visual seekers)	Specifies the color (red, green or blue) to be used by the tracker on visual images.
Fixed Image size Image size X Image size Y	When the FFT method is used in the calculations, the size of the images has to be a multiple of two (2, 4, 8, 16, 32, 64, 128, 256, 512, ...) in both the horizontal and vertical direction. If this is not the case the <i>Reticle</i> seeker model can change the size (by removing or adding pixels on all sides of the image).
Display in seeker channel	
Draw reticle image	An image of the reticle can be displayed in the seeker channel on top of the background image after the missile has been armed and locked but before it has been fired.
Draw signal before filters	The reticle signal can be displayed in the seeker window after the missile has been armed and locked.

Table 5 continued

Parameter	Description
Type of reticle	
Rising sun	
Number of spokes	8 in the images above
Radius	1 in the image to the left and 0.5 in the image to the right
FM1	
Number of spokes	8 in the images above
Radius	1 in all images above
Inner radius	0.224 in the first image 0.424 in the last three
Radius of inner structure	0.069 in the two first images above 0.169 in the last two
Distance from centre to inner structure	0.138 in the three first images above 0.238 in the last
NutAm1	
Number of radial fields	2 in the first image above 3 in the second image above
Radius	1 in all image above
1: Number of spokes	2 in all images above
2: Number of spokes	4 in all images above
3: Number of spokes	does not exist in the first image above, 8 in the last image

7 Missile motion

The Vega *EwMotion* module provides an easy to use library for real-time missile motion and is suitable for inclusion in a Vega application. The module includes interfaces for different types of motion models, a general motion model and more system specific motion models. A bridge to use the ACSL (Advanced Continuous Simulation Language) [10] models are under development. The ACSL models are detailed models of specific threat systems and are developed in another project at FOI (*technical threat system analysis*). The second class draws a graph of error angle and distance, between missile and target, over time. It also shows when countermeasures are launched. The parameters and positions of missile and target can be logged.

A module consists of a collection of classes each represented in LynX by an icon panel. The *EwMotion* (Electronic warfare Motion) module currently consists of 2 classes (panels) *EwMotion* and *EwGraph* as shown in figure 34.

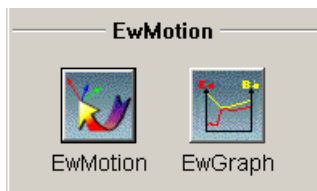


Figure 34 Icon panel from the *EwMotion* module

7.1 Application Interface

The *EwMotion* module includes an API to C/C++ language callable routines for using *EwMotion* functions within Vega. In order to build Vega *EwMotion* applications, an application must include the file *EwMotion.h*, and link with the *EwMotion* library. For Windows users, *psEwMotionS.lib* should be used for static executables and *psEwMotion.lib* (*psEwMotion.dll*) should be used for dynamic executables.

7.2 Initializing the EwMotion Module

If Vega is to recognize the *EwMotion* classes, an application must initialize the module after calling *vgInitSys* to initialize Vega. The function *InitEwMotion* initializes module classes for use with Vega. One parameter is to be sent with the initialization, *m_bUseFrame*. *m_bUseFrame* is used to specify which frames to use. It is the pipelining of Vega that causes this management of frames. Otherwise the frame that Vega is about to render and the actual frame seen are separated by one or more frames. This can cause serious problems when the image processing results in fed to the movement model. A *VG_FAILURE* is returned if an error has occurred, and *VG_SUCCESS* is returned if not. Once this initialization has been done, an ADF containing *EwMotion* class instances can be parsed by sending the ADF to *vgDefineSys*.

```

#include <vg.h>
#include <vgperf.h>
#include <pf.h>
#include <prmath.h>
#include "vgwin.h"

#include "ewCm.h"
#include "ewImage.h"
#include "ewSeeker.h"
#include "ewMotion.h"
/*
=====
    Main application entry point
=====
*/
void main( int argc, char *argv[] )
{
    vgWindow*   window;
    float       m_frameRate;
    float       m_frameTime;
    float       m_frameDelta;
    BOOL        m_bUseFrame = FALSE;
    BOOL        m_bUseFrameImage = FALSE;
    int         m_nSkipFrames;
    int         m_nCountFrames, i;

// init, define, and config the system

    vgInitSys();           // initialize Vega

    InitEWCM(); // vgInitFx(); is included

    vgInitSV();           // initialize sv
    vgInitSW();

// ----- Add init functions for implemented modules -----

    InitEwImage(&m_bUseFrame, &m_bUseFrameImage); //init the image module
    InitEwSeeker(&m_bUseFrame); //init the seeker module
    InitEwMotion(&m_bUseFrame); //init the motion module
// -----

    vgDefineSys( argv[1] ); // read in the ADF

    vgConfigSys();        // configure Vega

// ---- check if frames must be skipped -----
    m_nSkipFrames = (int)vgGetProp(vgGetSys(), VGSYS_NUMSTAGES);
    m_nCountFrames = m_nSkipFrames-1;

    m_frameRate = m_nSkipFrames*vgGetProp(vgGetSys(), VGSYS_FRAMERATE);
    m_frameTime = pfGetFrameTimeStamp();
    m_frameDelta = 1.0f / m_frameRate;

// ----- run 2 frames to get all systems ready -----
    for (i = 0; i < 2; i++)
    {
        vgSyncFrame ();
        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
    }
}

```

```

    vgFrame();
}

// ----- the real-time loop -----
if (m_nSkipFrames > 1) // Always when using SensorVision
{
    while ( 1 )
    {
        vgSyncFrame ();

        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        m_bUseFrame = !(m_nCountFrames % m_nSkipFrames);
        m_bUseFrameImage =
            ((m_nCountFrames++ % m_nSkipFrames) == m_nSkipFrames-1);
        vgFrame();
    }
}
else // not possible with sensorvision
{
    m_bUseFrame = TRUE;
    m_bUseFrameImage = TRUE;
    while ( 1 )
    {
        vgSyncFrame ();

        pfFrameTimeStamp( m_frameTime );
        m_frameTime += m_frameDelta;
        vgFrame();
        /* application specific code */
    }
}
}

```

7.3 The EwMotion module

The LynX EwMotion panels contain widgets for setting the parameters that define the Motion. If you are unfamiliar with how to use the LynX interface, consult the Vega LynX User's Guide before proceeding.

The *EwMotion* module currently consists of 2 classes (panels) *EwMotion* and *EwGraph*.

7.4 EwMotion

The screenshot shows the EwMotion panel with the following configuration:

- EwMotion** (Main Panel):
 - Motion: [Empty List]
 - Seeker: No Selection
 - Observer: No Selection
 - Object: No Selection
 - Motion model: Javelin
- Parameters** (Sub-panel):
 - Seeker:
 - Max Change: 1 [%/Frame]
 - K1: 4
 - Predict Pitch
 - Hund Curve
 - Missile:
 - Flying Mode: MODE_TOP
 - Speed: 100 [m/s]
 - Max g: 400 [m/s²]
 - Initial Pitch: 0 [°]
 - Peak Altitude: 0 [m]
 - Descend Pitch: 0 [°]

Figure 35 The EwMotion panel with motion model Javelin selected.

The EwMotion panel (see Figure 35) specifies the seeker, the observer and the object connected to the motion. The motion model must also be picked.

Table 6 *EwMotion Javelin Parameters (see reference [11] for more details)*

Parameter	Description
Max Change	Specifies how much the missile can turn each frame.
K1	A constant
Speed	The missile speed in m/s.
Max g	The maximum g-force allowed in m/s^2 .
Initial Pitch	The start pitch for the missile launch.
Peak Altitude	The top of the missiles flying path.
Descend Pitch	The pitch the missile is descending toward the target.
Predict Pitch	Sometimes the target will move out of the seekers field of view to fast. In such cases the missile motion model can predict the pitch angle based on how the pitch has increased during the last frames.
Hund Curve	Makes the missile always trying to fly straight at the target.
Flying Mode	Specifies if the missile first shall elevate to a certain altitude (MODE_TOP) or if it shall begin towards the target almost from the beginning (MODE_DIRECT).

7.5 EwGraph

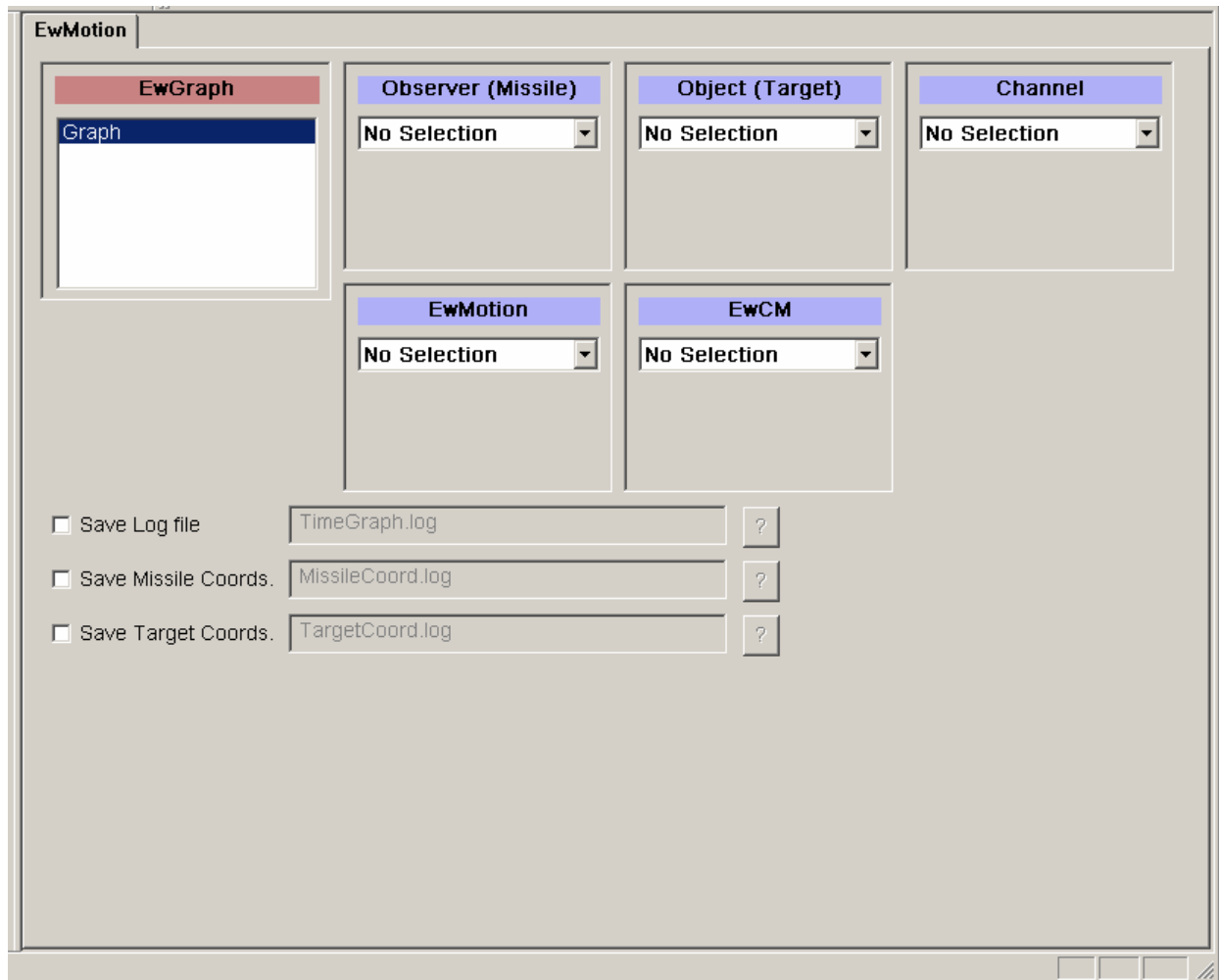


Figure 36 The EwGraph panel.

The EwGraph panel (see Figure 36) specifies in which channel the graph is drawn and which objects to use when calculating the values shown in the graph.

Table 7 EwGraph Parameters

Parameter	Description
Observer (missile)	The observer connected to the missile.
Object (Target)	The target
Channel	The channel which the graph is drawn in.
EwMotion	The motion instance which is connected to the missile.
EwCM	The countermeasure instance connected to the target.
Save Log file Save Missile Coords. Save Target Coords.	To save logs with graph data, the position of the missile, and the position of the target as a function of time.

8 Summary and conclusions

Simulation is a powerful tool for assessing the electronic warfare duel. This document has described a new framework for EW duel simulations, in the visual/IR wavelength range, called *EwSim*. *EwSim* is based on the commercial product, Vega, with the plug-in for IR imagery, SensorVision (the plug-in SensorWorks can be also used for sensor performance). Several in-house modules have been adapted to this framework in order to get EW-simulation capabilities. *EwSim* has real time (or close to real time) simulation capacity which means that it is possible to extend the use of the framework to include EW training of military personnel.

The *EwSim* is not an application but rather a set of modules that can easily be tailored to an application. An existing application which has made use of the modules in *EwSim* is an application for flare tactics on helicopters.

Future work with *EwSim* should include improvements of the models. For instance *EwCM* could be extended and include simulation of multispectral waterfog. *EwCmLaser* could include the effect of a laser jammer on a realistic imaging sensor, or intensity modulations in order to jam a reticle seeker. It should also be possible to include a realistic fine tracker in a DIRCM simulation, either based on IR or retro-reflection. The functionality of *EwSeeker* could be improved by developing other types of seekers/trackers. A target seeker based on rosette scanning is under development. Missile dynamics from ACSL models is a possibility. However, the connection between ACSL and *EwSim* has not been fully tested and more work is needed in order for this connection to work more seamlessly. An improved generic missile dynamic model is also under development.

A missile might be limited to the wavelength range covered by *EwSim* (visual/IR). However, an enemy might have access to more than one type of missile. Furthermore, future missile seekers might have multi-sensor capability which accentuates the need for a simulation environment that can handle an extended wavelength range (visual-IR-radar).

The components of EW-suites are expensive and it is not always possible to equip every platform with every component of the suite. Instead it is likely that future systems on a battlefield will have the components of the EW-suite distributed on many platforms and rely on a communication system to share information. To be able to simulate a distributed EW-suite it is therefore essential to include models of the communication link in the simulation.

9 References

- [1] C. Hedberg, L. Tydén och C. Wigren, *Generell metod för simulering av elektro-optiska telekrigdueller*, FOA rapport FOA-R--99-01160-616--SE (1999)
- [2] <http://www.multigen.com>
- [3] <http://www.cg2.com>
- [4] LynX User's Guide
- [5] Christer Wigren och K. Ove S. Gustafsson, *Implementering av laserstörmodell*, FOI-rapport FOI-R--0249--SE, November 2001.
- [6] Lars Tydén och Lars Berglund, *Laborationshandledning Korrelationsmålföljare*, FOA-rapport FOA-D--97-00335-616--SE, Augusti 1997.
- [7] Lars Berglund and Carl Hedberg, *Laborationshandledning Centroidmålföljare*, FOA rapport FOA-D--97-00335-616--SE, Augusti 1997.
- [8] Olsson, *Simulering retikelmålsökare Metodbeskrivning*, FOA rapport C 30685-8.3, 1992.
- [9] Christer Wigren, *Realtidssimuleringar av målsökare och motmedel (rök, vattendimma) i en komplex bakgrundsmiljö*, FOI-rapport FOI-R--0694--SE, December 2002.
- [10] ACSL Reference Manual Edition 11, 1999: AEGIS Simulation, Inc
- [11] Christer Wigren, *Pansarvärnsrobot mot motmedelsskyddade stridsfordon - en simuleringsstudie*, FOI rapport FOI-R--0261--SE