Realtime System Integration Lab: Active Protection System Case Study

Matthew Kursar
DEVCOM Armaments Center
Picatinny, NJ
matthew.w.kursar.civ@army.mil

Michael Greco
DEVCOM Armaments Center
Picatinny, NJ
michael.j.greco.civ@army.mil

ABSTRACT

Modern battlefields contain a variety of threats to modern armor systems. Defending against these requires a multilayer approach to fully protect the warfighter and ensure combat effectiveness. One of these layers to protecting armor systems is the use of Hard Kill (HK) Active protection systems (APS). These systems consist of multiple interconnected components to rapidly engage and defeat incoming munitions before hitting the vehicle. The process to create one of these systems is a complex effort of development, integration, and testing in a relevant environment. A helpful approach to this development process is a system integration lab. Using system hardware, emulators, and simulation tools the APS can be tested and developed quicker and more efficiently.

This paper will showcase the evolving interoperability of a system integration lab for a specific HK APS solution. A lab environment that blends physical components with virtual representations in a real time engagement to rapidly evaluate designs and performance. We will investigate how to utilize modular design and near real-time capabilities to implement Audio Video Bridge network messaging, electrical relays, and other messaging formats to directly interface with both hardware components and subsystem emulators. The audience will be shown an example use case where the simulated environment for an integration and testing laboratory is created and executed.

ABOUT THE AUTHORS

Matthew Kursar is the team lead for special projects within the Performance Analysis Branch of DEVCOM – Armaments Center, with almost 5 years of experience as an analyst and developer of the Performance Related and Integrated Suite of Models. Mr. Kursar has supported numerous programs by providing critical system level analysis and developing new models and analysis methodologies to best characterize system performance.

Michael Greco is the lead architect and developer of the Performance Related and Integrated Suite of Models (PRISM) modeling and simulation framework. The PRISM tool has been used to support the analysis of many projects locally for DEVCOM - Armaments Center and externally for the larger Army analytical community. Mr. Greco has over 10 years of experience working in force effectiveness analysis with the use of operational and system performance simulations.

Realtime System Integration Lab: Active Protection System Case Study

Matthew Kursar
DEVCOM Armaments Center
Picatinny, NJ
matthew.w.kursar.civ@army.mil

Michael Greco
DEVCOM Armaments Center
Picatinny, NJ
michael.j.greco.civ@army.mil

INTRODUCTION

The modern battlefield poses an increasing danger of man-portable shoulder-fired anti-tank weapons being employed successfully against the modern combat vehicle. Dangers to the warfighter include: urban conflict, advances in anti-tank weapon technology, a greater proliferation of weapons systems, and advanced tactics and situational awareness by a combination of satellite imaging, better training of forward observers, and unmanned aerial systems. These challenges are multi-faceted and cannot be met with a single point solution. Each and every solution comes with new burdens to the vehicle and must integrate successfully without impeding the mission of complementary systems.

With this in mind the Vehicle Protection Suite encompasses all the various technologies that can be used to protect the vehicle, one of which the Active Protection System (APS). These Active Protection Systems can be broken down further into two main applications: Soft Kill (SK) systems, which seek to render incoming threats incapable of hitting the target using systems like lasers, obscurants, or jamming systems; and Hard Kill (HK) systems that seek to kinetically defeat the threat through physical destruction. Destroying the threat includes causing early initiation or deflecting the threat from its intended trajectory. Both of these methods will limit the kinetic energy on the host vehicle and increase survivability.

This paper will focus on the HK problem space. It is a stressing and complex challenge with increasingly short timelines due to the characteristics of anti-tank weapon systems. An engagement can happen suddenly within which a system of systems needs to react, engage, and defeat an incoming threat. This system of systems consists of a sensor capable of detecting and tracking an incoming threat with both a high accuracy and large sample rate to feed a fire control system. The fire control system must be able to process the sensor data, compute a fire control solution, arm the system, and fire its countermeasure against the incoming threat while ensuring confidence in its solution for both safety and survivability concerns. There is rarely enough time for a second attempt, so every shot is crucial.

When developing such a complicated system of systems there is a considerable integration effort to consider. DEVCOM-Armaments Center (AC) was tasked as the lead system integrators for developing a HK countermeasure system into the APS framework, along with another vendor's sensor and helping the development of software for the APS Controller. This integration effort included the performance verification of the APS with modeling and simulation tools.

DEVCOM-AC established their own system integration lab (SIL) that came equipped with two powerful simulation machines, shown below in figure 1, and three isolated hardware bays. The isolated hardware bays enabled competing vendors to have an analysis performed without risk of unfair advantages or inadvertent sharing of performance. Furthermore, the SIL has the capability to execute both classified and unclassified analysis and hosts a slew of laboratory hardware, including adequate power supply, network switches, and oscilloscopes.

Figure 1: Laboratory Computer and Hardware Bay

The final piece missing was the simulation tool that would verify system performance. The simulation environment selected for this effort was the Performance Related and Integrated Suite of Models (PRISM). PRISM is a modeling and simulation framework that facilitates the development, integration and execution of system and subsystem based models in time stepped dynamic scenarios. The PRISM framework is a collection of classes and libraries that provide enabling functions for disparate models to interact. It was selected for its capability to quickly assemble full end to end system simulations through modular model interfaces. With C++ as the main programming language, there were opportunities to utilize many packages used in common hardware applications. For more information on the design decision, refer to the Development and Architecture of the Modular Simulation Framework, PRISM [Greco 2021].

SIL BACKGROUND

Active Protection System

As described above, an APS is a system of system solution to a complex problem. For this effort the end state architecture or layout of the subsystem components can be seen below in figure 2. The components utilized for this effort was a collaboration of multiple vendors work adopting to a new modular architecture and a new network standard. The task was to facilitate the integration and development of these systems subsequently verify their performance.

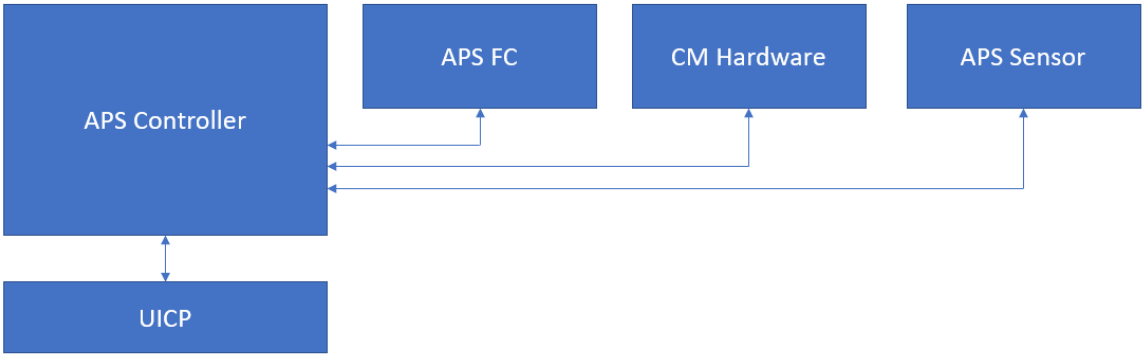

Figure 2: Basic HK APS Subsystem Layout

The components shown above in figure 2 are defined as follows;

User Interface Control Panel (UICP): This component contains the switches and buttons that the user directly interacts with within the final vehicle implementation. This component was physical hardware within the lab that the team interacted with to execute engagement scenarios.

APS Controller: This component acts as the main controller and network switch of the APS. Containing the logic that checks for the proper combination of UICP switches, monitors subsystem health, vehicle status, and approving of the arming and firing of countermeasures. This component was fully present as hardware within the system integration lab.

APS Fire Control (FC): This subsystem receives sensor data and processes it to determine threat trajectory and likelihood of engagement. The APS FC selects how to engage the target and sends its messages to the APS Controller for final approval. This component was present as hardware within the SIL.

APS Sensor: This subsystem measures the position and velocity of incoming threats to the vehicle. Sending its messages to the APS FC for further processing. Targets are tracked by specific criteria at a specified accuracy level and update rate. This component was fully represented in the simulation space for the SIL.

Countermeasure (CM) Hardware: The CM Subsystem performs the final engagement. It houses the explosive countermeasure that is utilized to defeat incoming threats. This specific component also contained a final sensing capability, utilizing lasers to affirm final position of the threat and compute exact timing of engagement. This subsystem was represented as lab hardware, a testbed implementation that lacked explosives but did include the laser sensing. And was additionally fully represented as a digital twin within the simulation environment.

Networking

The capability to communicate quickly and effectively with other systems is a core component of any system integration lab. In this effort we employed numerous network standards and took advantage of their various features Some details of the main network standards used are below:

Open Systems Interconnection Model

The Open Systems Interconnection Model or OSI model is a way of describing a network through the use of 7 layers shown below in figure 3. The layers are organized in increasing abstractness, where Layer 1 describes the physical architecture of the network and the network connections. Here is where the physical wires or wireless technology is described. The top most layer, called the Application Layer is the most abstracted layer of the model. It is at this layer that the end user can interact with data in software such as web browsers and instant messenger programs. The 4th layer in the OSI model is called the Transport Layer. The Transport Layer is where the networking protocols described below are operating. (Shaw 2022)

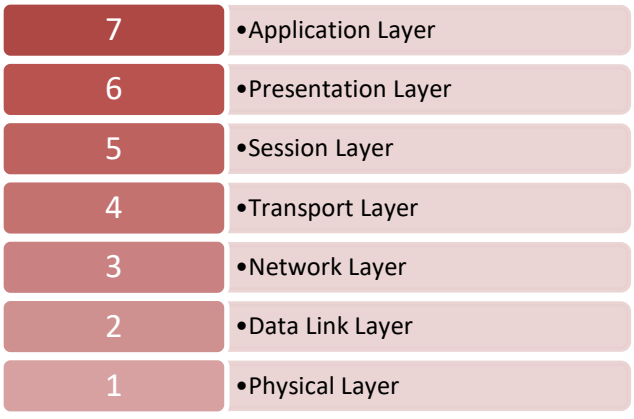

Figure 3 OSI Model Layers

User Data Protocol (UDP) -This is a communication protocol that places an emphasis on speed. It is commonly used for video playback or videogame connections. This method does not fully form a connection before sending a message and has no way to register if the message was received. An analogy is a blind handoff of data where the sender has no confirmation that the intended receiver has successfully obtained the sent data package. Furthermore, the sender has no ability to direct the message to a specific receiver attached to the network. The data message is broadcast to all listeners attached to the network. For guidelines on the use cases of UDP see (Eggert 2017).

Transmission Control Protocol (TCP) - The key aspects to a TCP connection are that it first must establish a connection between the sender and receiver of data. This protocol is not as fast as the UDP method as it comes with added robustness in its ability to ensure message packet ordering as well as its acknowledgements of message received. Sending a message over TCP operates much like a handshake, where the information is received, acknowledged, and both parties are aware of it.

The Asio C++ (https://think-async.com/Asio) library was utilized to establish networking capabilities within the simulation framework that was for integration into the laboratory. Asio is a cross-platform lightweight library that facilitates the formulation of data packages for both UDP and TCP. The library also provides enabling functions for establishing networking endpoints such as listeners and receivers, sending messages, receiving messages asynchronously, and processing the received data. By being able to receive messages asynchronously the simulation environment was able to provide close to real-time updates to the physical components attached to the integration network. The Asio library works on Windows and Linux systems through the creation and management of Handles and Sockets as shown in figure 4 below. Asio can be used with the C++ boost library or as a standalone library. For this application the standalone Asio C++ library was utilized.

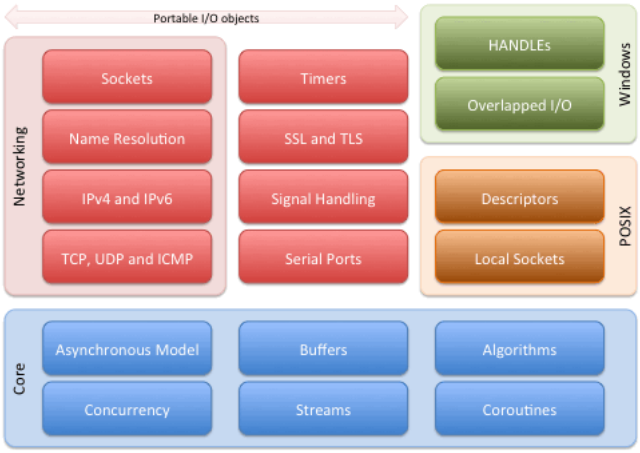

Figure 4 Asio is a portable C++ library that facilitates the asynchronous network communication between systems.

Audio Video Bridge (AVB) – AVB is the primary network for the new HK APS system. Most connections in the new architecture rely on this format. AVB is a network format that originally made its appearance in audio visual streaming applications with time sensitive data. For example, the synchronization of audio out of speakers at a large concert venue, where the speakers farthest from stage must be delayed by some consistent value. This timing is kept consistent by the usage of a grandmaster clock. The grandmaster clock is a system identified to be the "truth" time value, and that truth is communicated and tracked by each and every AVB compliant system connected to the AVB network. Having this common master clock eliminates some of the potential issues of timing differences across hardware components. This was a critical value required to ensure that our simulation was marching in lock step with the rest of the components. Each AVB message sent within the system had an associated master clock time, giving a chronological, and observable sequence of events for post action review.

AVB is set up in the format of talkers and listeners. As stated above the protocol reserves a portion of the Ethernet bandwidth available to set up a series of streams. In this application each of the streams can is like a lane in a highway, and each lane is only allowed to send specific message type(s). A priority can be given to those streams to ensure that the highest priority message(s) are sent first. This is unlike TCP, which would wait for the queued messages to finish sending before sending the next high priority one. When using AVB, that message jumps the line. One final unique aspect of AVB is that each piece of AVB hardware is capable of acting as an AVB switch. This means that in certain applications the required number of cables to connect multiple devices is greatly reduced and can be greatly advantageous to tactical systems. More details on AVB can be found here (IEEE)

To work directly with the AVB network an additional C++ library was required. A modified version of the openAVB library was integrated into the simulation environment. This library was needed to form compliant data packages and ensure that the time critical messages were sent using the reserved time sensitive bandwidth. The Asio library was still used to establish the asynchronous functions that would handle the receiving and processing of messages on the network. The openAVB library can be found as part of the OpenAvnu project shown in figure 5 which is located here: https://github.com/Avnu/OpenAvnu

Figure 5 Logo for the Avnu Alliance; The community which establishes the time sensitive AVB protocols and standards.

Discrete Input/Output

Discrete triggering, or discrete input/output (I/O) sometimes referred to as digital I/O is a simple way to send a signal. For a simpler way of sending messages, or a more basic way to interface between hardware and software.

In discrete input/output the signal will be either on or off, an example of such being a light or power switch. They are commonly used throughout many household goods. Furthermore, these signals can be communicated through hardware to and from a computer. Taking advantage of relevant hardware and proper supporting libraries, software can be created that directly reads or writes to the multiple I/O interfaces.

The Advantech PCI-1730 Digital IO Card was installed into the simulation computer in the system integration laboratory. Example code was provided with the Advantech card which showed how to trigger a discrete signal from C++ code. That example code was then integrated into the simulation framework, such that during a specific event within the simulated test environment the signal could be sent to additional external components.

Real Time Analysis

The simulation tool selected was initially designed to run as a faster than real time analytical engine using a defined timestep to perform physics based calculations. This was typically used to submit large batches of jobs to high performance compute schedulers to enable Monte Carlo analyses.

In order to act as a system emulator and threat scene generation tool, a new operating mode had to be developed. For this effort the simulation would need to run at real time to appropriately assess system performance. Models had to perform their individual action, messages had to be created and sent, all in quick succession before the next simulation timestep. There could be no getting ahead, or falling behind, of the hardware's actions within the virtual and lab environment.

The way this requirement was met was syncing to the system clock by taking a time sample before the simulation updated using the C++ "chrono" library, perform the simulation update, sample the system clock after the update,

and then finally calculate the elapsed time; giving us our next simulation step time delta. This process is illustrated in Figure 6 below.

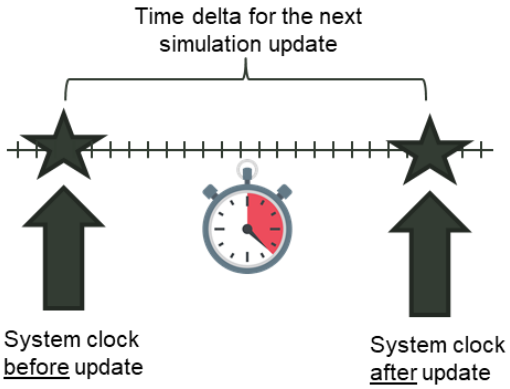

Figure 6: Illustration of time delta calculation.

Initial testing was performed on a Microsoft Windows-based machine. The average time deltas for a simple scenario were around 1 millisecond. It was observed that the simulation was updating faster than the clock could determine a difference in time. There were multiple simulation updates all with no difference in the "before" and the "after" system clock samples. A fail-safe was built in to use the last valid time delta which ended up being about 1 millisecond. The resolution of these time deltas was not fast enough to meet the lab system requirements so, a CentOS 7 Linux machine was then setup with the test scenario. The resolution on the system clock measurements increased by an order of magnitude. Now instead of the 1 millisecond cap on the time step resolution the simulation run on the Linux system was reporting time deltas in the nano seconds (see Figure 7). The new time delta resolution exceeded the laboratory requirements and further development of the simulation environment was allowed to continue.

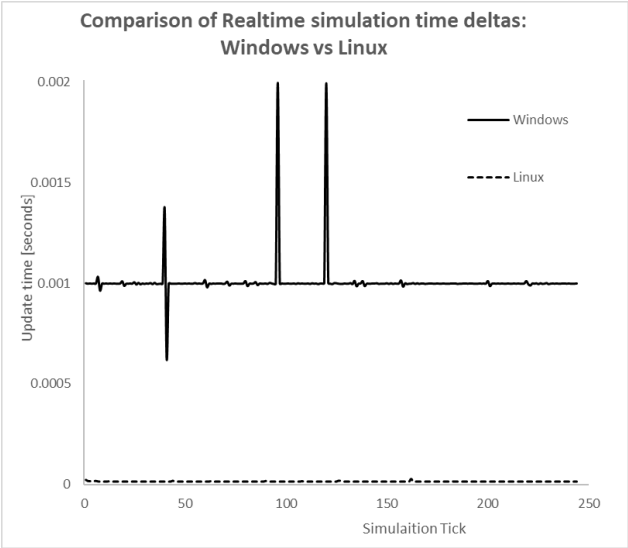

Figure 7 Resolution on the Linux system clock was sufficiently stable and small enough to meet system mission requirements.

SYSTEM INTEGRATION

The activities performed in the system integration lab can be broken down into two categories. First was the integration and adoption of the new architecture. This involved building a real time simulation in parallel to vendor development of their own corresponding subsystems. And supporting the development of subsystem interfaces and transitioning to the end state network standard.

The second category was testing of the fully integrated system. Performing a series of engagement scenarios with the real time system to verify message flow, fault conditions, and HK APS effectiveness.

Connecting Simulation and Hardware

The system as shown in figure 2 was further divided into the hardware, lab tools for testing, and the simulation software for modeling subsystems and the overall engagement. Shown below in figure 8 a breakdown of the relationships between simulation software, lab tools, and vehicle hardware.

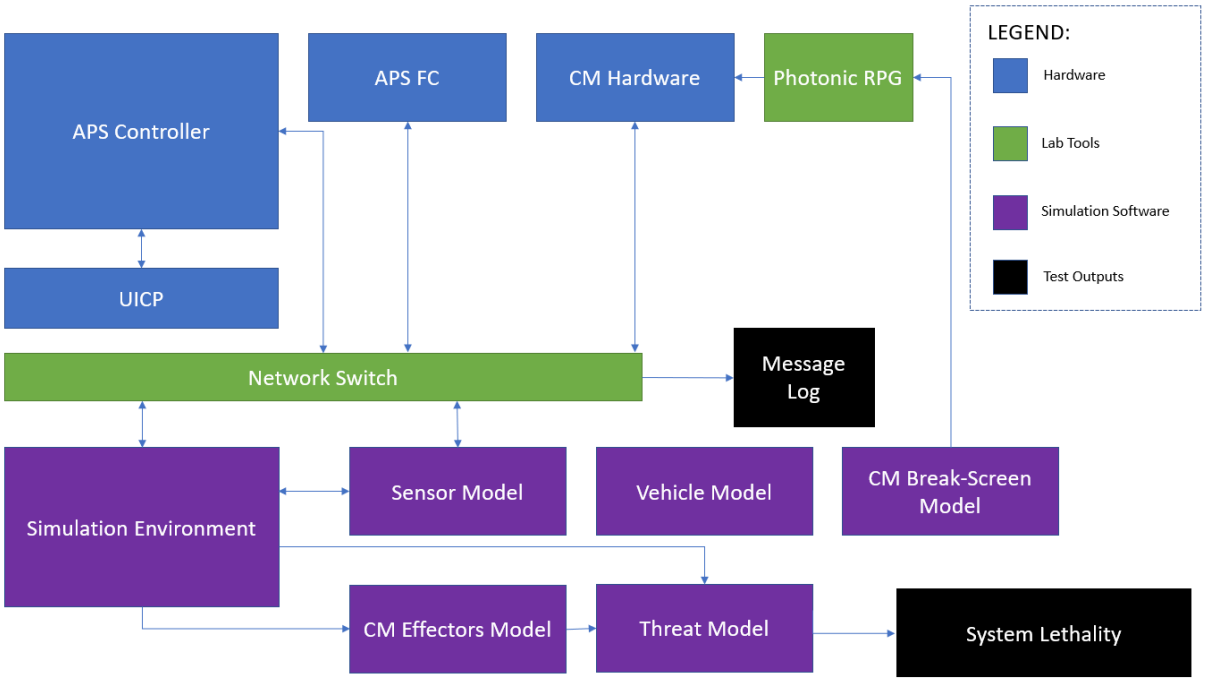

Figure 8: The HK APS SIL setup for development and test

The arrows in figure 8 represent some of the key pathways in which information flowed across various models and components. Information can be in the form of network messages, discrete I/O, and important data shared through the simulation environment to virtual models that require them.

Tying the whole lab environment together was the inclusion of a Micron Networking switch. All network messages in the system were passed through the switch on its way to the various components. This enabled the team to perform a variety of things. The first and foremost is easily extract the content of messages for review. Utilizing Wireshark, and custom parsing scripts we were able to track and decipher each message sent across the system, including the contents of their messages. This was helpful in debugging, observing, and verifying basic system behavior.

The switch was also utilized to mirror specific messages to the simulation environment. More specifically, additional listeners were set up from the simulation environment to capture data required for the overall engagement simulation. For example, this APS sensor does not need to know what time the munition fired; however, the simulation does need this information to compute resultant lethality.

Throughout the development and integration process there was a need to utilize these network and hardware connections. The development of logic for specific subsystems benefitted from early test procedures. And not all components made the network transition at the same time. We used the existing connections and network messages to perform tests on system logic.

To perform these early and iterative tests the team took advantage of UDP, TCP and AVB messages between simulation models and hardware components as each completed their required interfaces. The simulation made use of separate files that defined message format, and only required swapping out the particular model or "hook" related to the network format utilized. Using common function names, the overall engagement simulation required little to no changes between updates.

SYSTEM TESTING

Test Procedure

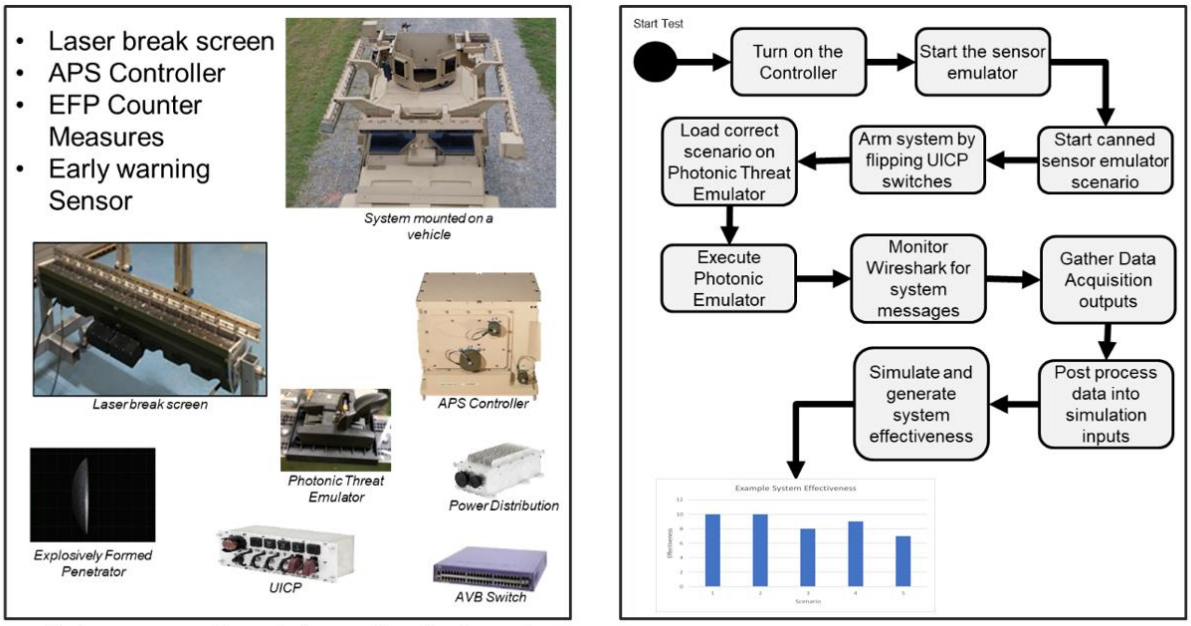

Figure 9: Lab setup and test procedure

The traditional use case of the simulation tool typically consists of setting up a large series of analyses and completing them in parallel using an HPC. With this effort that approach is not an option as one of the key aspects of the system is the inclusion of a human in the loop. This required the physical flipping of switches that were monitored by the APS controller. These human interfaces are required for safety of the final system and need to be included for a true system representation. The overall flow of the testing is shown in figure 9 above and went as follows:

1. Turn on the power to all subsystems.(Lab power supply)
2. Human Flip the Power Switch on the hardware UICP

3. Execute the command line Simulation executable with proper input deck
4. Human interact with the UCIP - Flip the specific switches that signal to the APS that it is now active and armed.
5. Engagement simulation begins
6. System Responds to incoming Threat
7. Ingest Fired Command from countermeasure Hardware
8. Execute Lethality Calculation

System Evaluation

This specific system was being designed and tested to defeat Rocket Propelled Grenades (RPGs) at short range by means of launching a row of Explosively Formed Penetrators (EFPs). These RPG threats can be launched from a variety of ranges and positions, sometimes employing precursor or coordinated attack tactics that require the simulation of a dual threat defeat. This challenging but real scenario requires modeling and simulation to truly test the system prior to any test events. The system integration lab simulated all aspects of the scenario; including the threat launcher, threat flyout, APS sensor, laser break screen, the countermeasure launcher, flight of the EFPs, and the resulting collisions on the threat body.

Several specific engagement scenarios had been defined by requirements and were evaluated for system performance. The system was evaluated on its ability to respond to the various situations. Using the captured data to verify message flow, fault conditions, and HK APS effectiveness.

Threat launchers were positioned within the simulation at varying azimuth and elevation angles relative to the platform. Some threats originated at ground level while others were positioned in the air, which simulated an RPG being fired from a rooftop. The threat launcher has the ability to fire a multitude of different threat scenarios, giving the test engineers the ability to mix and match threat systems.

After the launcher fired the threat at the vehicle, the threat would instantiate into the simulation environment (see Figure 10 below). A representative flyout model moved the threat towards the protected vehicle.

Figure 10 Threat launcher and flyout.

As the threat was moving towards the protected vehicle, a simulated APS sensor would detect it. In real time, the simulated sensor would formulate threat detection messages, compile the messages in a valid system-compliant package, and then send the message to the fire control. The fire control would listen to incoming messages from the sensor emulator and make multiple determinations, including deciding if the threat posed a danger to the vehicle, and if so, which counter measures should be activated to engage the incoming threat. Once the set of counter measures and the laser sensor were armed, the physical system would wait until the laser sensor was triggered.

Within the simulated environment, the threat continued towards the platform. Once the simulated laser screen was ‘broken’ by the incoming threat the simulation would send a discrete signal to another physical piece of hardware,

which used pulsing light to emulate the crossing of the threat across the physical break screen in the CM hardware. This started the final fire control calculation of timing the detonation based on the measured position. The system "fired" it's countermeasure at the calculated time. For this system integration lab, the firing of the countermeasure was accomplished by triggering a light on the hardware itself, no explosives were used. The controller then gathered the fire report in which the details of which countermeasures fired and at what time delta after the threat broke the screen. This fire report was then read back into the simulation.

Figure 11 Representative threat hits the laser break screen.

At this stage the engagement had been completed. To simulate the detonation of the countermeasure the simulation switched from a real time operating mode to a fixed time step operation. This was to ensure adequate fidelity in the timestep of the fragment effectiveness analysis. Once the fire report was read back into the simulation environment, the virtual counter measures would fire the EFPs at the reported fire time. Each individual EFP flight was simulated along with the incoming threat to estimate where on the threat body the EFPs hit shown in figure 11 above. Using those impacts an estimated lethality could be calculated. Final review concluded by verifying the correct order and content of the network messages sent over the switch. This paired with the estimated lethality of the specific engagement gave a full picture of the systems performance.

REFERENCES

Greco (2021), Development and Architecture of the Modular Simulation Framework, PRISM, DTIC #: AD1123532

Eggert, L., (2017, March). UDP Usage Guidelines. Datatracker. https://datatracker.ietf.org/doc/html/rfc8085

IEEE, (25 April 2011), 1733-2011 - IEEE Standard for Layer 3 Transport Protocol for Time-Sensitive Applications in Local Area Networks, https://ieeexplore.ieee.org/servlet/opac?punumber=5762651

Shaw, Keith (22 October 2018). "The OSI model explained: How to understand (and remember) the 7 layer network model". Network World. Retrieved 16 May 2020.