



**DARPA-PA-18-02**

**DARPA PA for AIE**

**Milestone 3 Report**

**November 4, 2019**

|  |  |
| --- | --- |
| **AIE Opportunity #** | **DARPA-PA-18-02-05** |
| **Proposal Title** | **Reservoir Algorithm Implementation Using a Sensay Device** |
| **Proposer Organization** | Look Dynamics Inc. |
| **Type of Organization** | Small Business |
| **Proposer’s Internal Reference Number** | LK-19-31 |
| **Technical Point of Contact (POC)** | Name: Rikk Crill  Address: 105 S. Sunset St., Suite T  Longmont, Colorado 80501  Telephone: 303-917-9845  Email: rcrill@lookdynamics.com |
| **Administrative POC** | Name: David Bruce  Address: 105 S. Sunset St., Suite T  Longmont, Colorado 80501  Telephone: 303-588-1442  Email: dbruce@lookdynamics.com |
| **Other Team Members** | Technical POC Name: Dr. Gokhan Sahin  Organization: Miami University  Organization Type: University |

Proprietary Statement: This document contains information proprietary to Look Dynamics Inc. Disclosure to others requires written consent from Look Dynamics Inc.

**List of Acronyms and Terminology**

|  |  |
| --- | --- |
| Antilles | Fourier Wedge Modulator |
| FOV | Field of View |
| FPA | Focal Plane Array |
| fps | Frames per Second |
| HGV | Hypersonic Glide Vehicle |
| IR | Infrared |
| km | kilometers |
| LD-MU | Look Dynamics-Miami University |
| LEO | Low Earth Orbit |
| PNN | Photonic Neural Network |
| PWM | Pulse Width Modulated |
| ReLU | Rectified Linear Unit |
| Sensay | Sensor Display |
| SNR | Signal-to-Noise Ratio |
| Trixel | Transmit Receive Pixel |

**1. Milestone 3 Summary and Programmatic Activities**

The Look Dynamics and Miami University team is pleased to submit this Milestone 2 Report for DARPA’s DARPA-PA-18-02 program. The team has continued to make significant progress during the Milestone 2 period of the program and all activities are either on schedule or ahead of schedule.

In terms of programmatics, the process of setting up the subcontract with Miami University has been completed. The team continues to use the secure site to coordinate project activities and a Gitlab software configuration control site has been set up and is being used. The team had weekly teleconferences to coordinate all program activities. The team is using a documented peer review process, and material generated for this Milestone report was generated using this process.

In addition to the activities detailed in the contract, the team has added the task of looking forward to the implementation of the Antilles filter in Phase 2. As described in Section 4, an approach using graphene that avoids issues with silicon is being investigated.

The team is aware of a project teleconference on September 5, a Phase 1 meeting in October, and a Peach conference in San Diego in March. The team plans to participate in each of these events.

The following sections describe the main technical activities the team performed during the Milestone 2 period of the program. Section 2 describes the generation of synthesized data to be used in simulation of the PNN. The application chosen is the detection and tracking of HGVs using infrared images obtained from a LEO satellite. The team has completed simulations that generate valid HGV tracks. This program is listed in Appendix 1. This program stores starting points for tracks and these starting points are used to begin HGV tracks in training the neural network. A Matlab update program has been written for use in the neural network training program. This program is listed in Appendix 2. Section 3 describes the hardware operation of the PNN hardware. The time to complete each step is calculated, the latency through the system for different modes of operation has been determined, and an initial estimate of power consumption is given. As noted above, Section 4 describes additional work the team is performing in anticipation of Phase 2 work. Implementation of the Antilles filter using a graphene based circuit is described.

**2. Simulator for Generating Data for HGV Tracking and Detection Application**

The application chosen for this project is the detection and tracking of HGVs using infrared data from a satellite in a LEO orbit. When a vehicle is launched, it is known that detection of the radiation from the launch event is a fairly easy task. After launch, a fairing is removed and initially the HGV is at a relatively low temperature, which may not allow it to be detected, especially against the background of the earth. After the fairing is removed, the vehicle will begin to increase in temperature if it is in the Earth’s atmosphere or it will eventually increase in temperature when it re-enters the atmosphere. The focus of this project is on the more difficult aspect of the application, detection of the HGV after the fairing is removed and the vehicle increases in temperature and tracking of the vehicle after it is heated by the atmosphere. Therefore, some data sets will consist of a target heating up and appearing during a trial, while most data sets will assume the HGV has already increased in temperature before entering the FOV of the imager and therefore will be detected as soon as the target enters the edge of the FOV.

This section describes the process of generating the data for the images that will be used in the neural network processing. This section also describes the simulator that was developed to generate the data for target tracks and its interface with the simulation of the PNN. The geometric parameters used in generating the simulated data were derived previously in Technical Report 1. The team has developed a MATLAB program to generate simulated data for target tracks across the FOV in random directions that allows the SNR for the target to be easily adjusted. The simulator generates data as needed rather than store large amounts of data. As noted milestone report 1, a vehicle will enter the edge of the FOV for most cases, which will be our assumption throughout this section. When necessary, the spherical coordinates (R, theta, phi) are used, where R is the distance from the origin (the origin is at center of the Earth), theta is the rotation angle in the x-y plane where theta equal zero is along the x-axis, and phi is the rotation from the positive z-axis. A right handed coordinate system is assumed.

The simulator generates sufficient data to succinctly characterize a large number of HGV tracks in which the HGV remains within the view of the FPA for a sufficient number of frames to allow the PNN to operate (referred to as “valid cases” hereafter). This is done to avoid cases where the HGV cuts across the edge of FPA for only a few frames where it would not be expected a detection should occur. The following describes the process to generate each such HGV track and ensure its validity, which is also summarized in the flowchart of Figure 1. The process begins by choosing an HGV to be at a random point in the image on the FPA. The entire FPA is 4096 x 4096 pixels. The entire FPA is broken up into 768 x 768 pixel overlapping tiles because the PNN to be constructed in Phase 2 will use a 768 x 768 pixel format. This will allow comparison of Phase 1 results with those of Phase 2. For this milestone, the sub-tile chosen is the one in the center of the 4096 x 4096 array.



**Figure 1. Flowchart for Generating Simulated Data for HGV Tracks**

For the initial randomly chosen point on the FPA, the coordinate system is aligned such that the FPA and lens in front of the FPA have their centers along the x-axis, and have y and z components equal to zero, as shown in Figure 2. In addition to the Cartesian coordinates shown, standard spherical coordinates are used in some calculations. For spherical coordinates, R is the distance from the origin to a point, theta is the angle of rotation in the x-y plane where a point along the x-axis is at zero degrees and theta increases as a vector is rotated in the counterclockwise direction in the x-y plane, and phi is the angle that a point is rotated from the positive z axis. The FPA and lens rotate in the x-y plane with theta initially at zero radians and increasing. The center of the lens is at the satellite altitude above the surface of the Earth, and the FPA is one focal length of the lens beyond the lens. In this configuration, a random point on the FPA is selected as the first location of the HGV’s image within the FPA. This is illustrated as point A in Figure 2. From this geometry, point A on the FPA can be mapped to point B, which is the actual location of the HGV.

****

**Figure 2. Depiction of Initial Geometry for Simulated HGV Data Generation**

The satellite altitude is set to be 800 km above the Earth, the missile altitude is a randomly chosen value between 50 and 80 km above the Earth, and the missile speed is randomly chosen between Mach 5 and 25. Although HGVs change altitude and speed during flight, for the current simulation these parameters are held constant. This is justified by the short time an HGV will be in the FOV of a sub-tile, during which time the speed and altitude for an HGV will change very little.

The focal length of the lens is calculated as the distance from the center of the lens, which is assumed to be along the x-axis, to the center of the FPA. Note that the center of the lens is at the altitude of the satellite. Thus, the focal length is given by (using Matlab notation):

flen = (FPApix/2)\*pitch/tan((FOV/2)\*pi/180)

where FPApix is the number of pixels along the side of full FPA and is 4096, the pixel pitch is 10 um, and the FOV is 120 degrees as computed in Technical Report 1.

The number of pixels along a sub-tile side 768, and is denoted as numpix in the code. The convention used in generating the initial coordinates of the randomly chosen point on the FPA is that viewed from the center of the Earth looking at the FPA, where column zero is on the left side of the FPA and row zero is at the bottom of the FPA. Hence, the random pixel location coordinates (xFPA, yFPA, and zFPA) are calculated by:

FPARow=rand(1,1)\*numpix;

FPACol=rand(1,1)\*numpix;

yFPA=(numpix/2-FPACol)\*pitch;

zFPA=(FPARow-numpix/2)\*pitch;

xFPA=Erad+satalt+flen;

The first location of the HGV, depicted as Point B in Figure 2, is calculated as the intersection of the line that goes through point A and the center of the lens and the sphere on which the HGV travels. Note that the line connecting point A and point B will necessarily go through the center of the lens, providing two points with known coordinates that uniquely define the line. The equation of the line is obtained from the following reference: <http://mathforum.org/library/drmath/view/65721.html>. Then the intersection of the line and the sphere that contains the HGV is obtained from the following reference:

<http://paulbourke.net/geometry/circlesphere/>. The detailed equations for the calculations can be found in the attached simulation code shown in Appendix 1.

The next step is to find a point a distance away from the initial missile location, where the distance is equal to the distance the HGV will travel during one frame time. Given point B (the first location of the HGV) and the speed of the HGV, the next position of the HGV can be any point on the HGV sphere with distance from B equal to the distance traveled by the HGV in one FPA frame, which is chosen as 1/30 of a second, corresponding to 30 fps.

The point is chosen at a uniformly distributed angle from the initial missile location. To ensure a uniform distribution, we consider a hypothetical point on the North pole (x=0, y=0, z= Erad+missalt), choose a random angle for theta, where theta is the angular rotation in the x-y plane, find the point on the sphere corresponding to that point, and then rotate that point using the same rotation that would be required to rotate the North pole to the initial HGV position. Phi can be calculated from the arc distance (distance missile travels) being equal to the radius times phi in radians. The point is called P3 in the following code and is calculated as:

P3R=Erad+missalt

P3theta=rand(1,1)\*2\*pi

P3phi=(misssp/fps)/(Erad+misalt)

where, Erad is the Earth radius, missalt is the altitude of the HGV, missp is the speed of the HGV, and fps is the number of frames per second. The Cartesian coordinates of P3 prior to rotation are computed as:

xP3=P3R\*sin(P3phi)\*cos(P3theta);

yP3=P3R\*sin(P3phi)\*sin(P3theta);

zP3=P3R\*cos(P3phi);

The rotation required to rotate the north pole to the point on the HGV sphere (point B) that was determined from the random mapping on the FPA is determined using the following references:

<https://en.wikipedia.org/wiki/Rotation_matrix>, and <https://math.stackexchange.com/questions/114107/determine-the-rotation-necessary-to-transform-one-point-on-a-sphere-to-another>. This same rotation is applied to P3 to get the next point for the path

of the HGV. The rotation order is to first rotate by an angle of phi about the y-axis, which preserves the y-coordinate, which makes theta (the angle in the x-y plane) equal to zero. This is followed by a

rotation of theta about the z-axis, which preserves the z coordinate. The rotation by phi about the y-axis is given by:

P3byphi=[cos(missPhi) 0 sin(missPhi); 0 1 0; -sin(missPhi) 0 cos(missPhi)] ...

\*[xP3; yP3; zP3]

This is followed by rotation by theta about the z-axis, which gives P4, the second HGV location in Cartesian coordinates as:

P4=[cos(missTheta) -sin(missTheta) 0; sin(missTheta) cos(missTheta) 0; ...

0 0 1]\*P3byphi;

The two locations of the HGV, namely P3 and P4, uniquely define the Great Circle passing through them. Following the reference <https://www.nosco.ch/mathematics/en/great-circle.php>, the Great Circle equation parameters are found as:

u=[xMiss yMiss zMiss]/(Erad+missalt);

w=cross([xMiss yMiss zMiss].',P4);

v=cross(u,w)/sqrt(w(1)^2+w(2)^2+w(3)^2);

The next step is to calculate the intersection of the line passing through the new HGV location and the center of the lens with the plane in which the focal plane is located at the second point in time. The following reference gives the solution to this problem: <http://paulbourke.net/geometry/pointlineplane/>. It is noted that the line from the origin to the center of the FPA is normal to the FPA. Therefore, P3 can be the point where the line from the origin instersects the middle of the FPA, and this line is normal to the FPA. x/y/z/Missarr(2) denotes the HGV’s coordinates at the second point in time.

The utilize this method, the first step is to calculate the positions of the lens and the FPA at the second point in time. Both of these stay in the x-y plane, but rotate from the x-axis toward the positive y-axis. The angle in radians by which the satellite rotates in the x-y plane for each increment of time is given by:

anginc=(2\*pi/orbtim)\*(1/fps)

where orbtime is 90 minutes times 60 seconds. The new lens and FPA center positions are computed by:

xLensLoc=(Erad+satalt)\*cos(t\*anginc);

yLensLoc=(Erad+satalt)\*sin(t\*anginc);

xFPALoc=(Erad+satalt+flen)\*cos(t\*anginc);

yFPALoc=(Erad+satalt+flen)\*sin(t\*anginc);

where t is a time parameter of the Great Circle equation. The location of the image of the second HGV location on the FPA is computed by:

u2=dot([xFPALoc yFPALoc],[xFPALoc-xMiss ...

yFPALoc-yMiss])/dot([xFPALoc ...

yFPALoc],[xLensLoc-xMiss ...

yLensLoc-yMiss]);

xFPA=miss(i1,13)+u2\*(xLensLoc-xMiss);

yFPA=miss(i1,14)+u2\*(yLensLoc-yMiss);

zFPA=zMiss+u2\*(0-zMiss);

This is followed by the calculation of the corresponding row and column on the FPA. First, the distance between the edge of the FPA near column 1 and the current point on the FPA corresponding to the HGV is found. The coordinates are given by:

if miss(i1,7)<0

xFPAedge=xFPALoc+(numpix/2\*pitch) ...

\*sin(t\*2\*pi/(orbtim\*fps));

else

xFPAedge=xFPALoc-(numpix/2\*pitch) ...

\*sin(t\*2\*pi/(orbtim\*fps));

end

yFPAedge=yFPALoc+(numpix/2\*pitch) ...

\*cos(t\*2\*pi/(orbtim\*fps));

dist=sqrt((xFPAedge-xFPA)^2+(yFPAedge-yFPA)^2);

FPACol=dist/(pitch);

FPARow=zFPAarr(2)/pitch+numpix/2;

After the calculation of the first two points of the HGV, the time parameter t, is decremented until the HGV is at the edge of the sub-tile. If enough frames to be considered a valid case are required for the missile to reach the sub-tile edge, the conditions where the missile is at the edge of the sub-tile are stored and constitute one valid track. If too few frames are used when the missile is at the entrance edge of the sub-tile, the time parameter is increased until the missile reaches the exit edge of the sub-tile. If enough total frames occur, the conditions where the missile enters the sub-tile edge are stored and constitute one valid track. If too few frames occur between the entrance and exit edges, no parameters are store, new random parameters are generated, and the process is repeated. The valid cases are stored in a file and are used in the simulation to test the neural network processing. Note that the conditions where the missile is at the entrance edge of the sub-tile are sufficient to regenerate the full track of the HGV simulated data. An example of the track of missile across the sub-tile is shown in the figure below. As described below, when this track is used in the update program, noise is added to the non-track pixels and the level of the track associated with the vehicle is adjusted based on the SNR chosen for the track.



**Figure 3. Example of an HGV Track Trajectory**

For simplicity, and without loss of generality, only the portions of the HGV track that remain within the first and fourth quadrants in the x-y plane are included in the simulator. The data generated from this simulation program is saved in a succinct form into a matrix, called StartingPoints, where each row corresponds to a specific HGV and includes the information required to recreate its trajectory. The StartingPoints matrix serves as the main input for the program that interfaces with the PNN algorithm. Specifically, the format of the StartingPoints information for a particular HGV is as follows:

%Format of Starting Points:

%StartingPoints(1:3); Great circle parameter u

%StartingPoints(4:6); Great circle parameter v

%StartingPoints(7); Parameter t for the Great Circle equation

%StartingPoints(8); snr for this missile (not in dB)

%StartingPoints(9); missile altitude above Earth in meters

%StartingPoints(10); missile speed in m/s

%StartingPoints(11); FPA row position of missile

%StartingPoints(12); FPA column position of missile

%StartingPoints(13); Missile x location

%StartingPoints(14); Missile y location

%StartingPoints(15); Missile z location

The program interfacing with the PNN is described next. The program begins with loading the StartingPoints matrix, and choosing a random row, corresponding to a particular HGV. The process consists of generating an initial frame including random noise for each pixel and random SNR for each HGV, and then repeatedly updating each HGV’s location in space and on the FPA before generating the next frame (with random noise) as long as there is one HGV still within the FOV. For each pixel, a Gaussian noise with a standard deviation dependent on the SNR. The process of updating the HGV’s location in space and the corresponding image of the HGV on the FPA is similar to the process described earlier in generating the HGV track, and details can be found in the actual interface program included in this report.

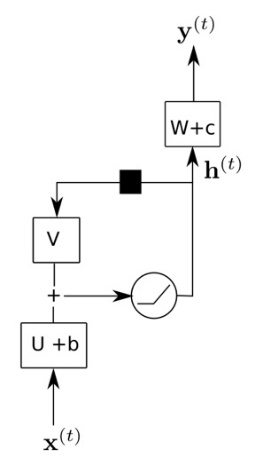
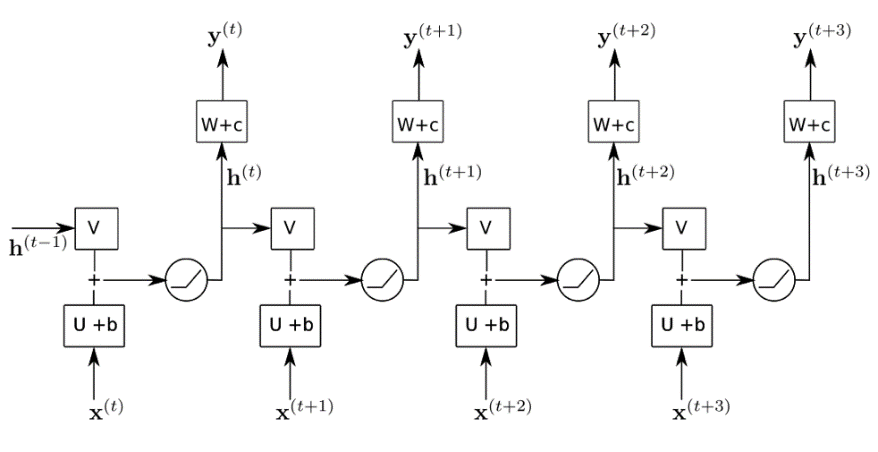
**3. PNN Hardware Operation**

This section explains the operation of the PNN hardware to implement the required processing. Section 3.1 provides a background describing the processing and the PNN hardware. Section 3.2 describes the performance of the hardware based on speed of the components. Included is analysis of the frame rate, latency, and an optional direct feed approach.

**3.1 Background**

**3.1.1 Algorithm Background**

The basic structure of a recurrent CNN is shown in Figure 4 (left). Consider the input as an input raster at time , with channels, rows, and columns of data. For example, in the HGV application, we have assumed channels of infrared sensor data for the following analysis corresponding to two infrared bands such as at 2.8 and 4.2 um. A resolution of pixels of input will be used, corresponding to a tile of a 4k x 4k FPA as described in the previous section.

**Figure 4. (left) Recurrent CNN Unit as Described; (right) Unfolded Network for 4 Time Slices**

A recurrent CNN uses a latent state (which serves the role of a reservoir) to summarize an object’s history. We use the notation to indicate a latent state at time and it has channels and rows and columns of input. The value is a hyperparameter we will tune based on our data, but for the purpose of discussion we assume .

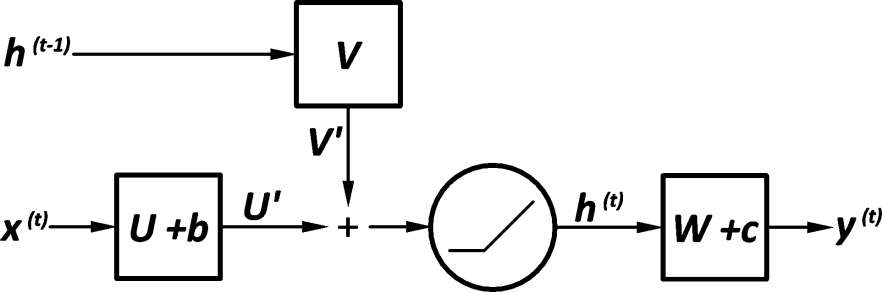
In a recurrent CNN we aim to estimate the convolution weights , , so that

where is a convolution operation and is a non-linear activation function (such as a ReLU). This predicts the next latent state (reservoir) as a function of the input and the previous latent state. The parameter describes a convolution operation with channels of output and channels of input, so there are different 2-D convolutions to perform because each output channel requires computing a sum of convolutions over each input channel. Many CNNs use filters which results in 9 learnable parameters per 2-D convolution; Look Dynamic’s PNN can have up to 127 learnable filters per 2-D convolution. Similarly, the parameter describes convolutions because it maps channels of input (from ) to channels of output.

The output of the network at time is

where are learned weights and is another activation function that depends on the attribute to be predicted: e.g. an identify function for regression or a sigmoid or softmax for classification. A recurrent CNN is trained by backpropagation through time, which amounts to training an unfolded recurrent network over sequences of training data.

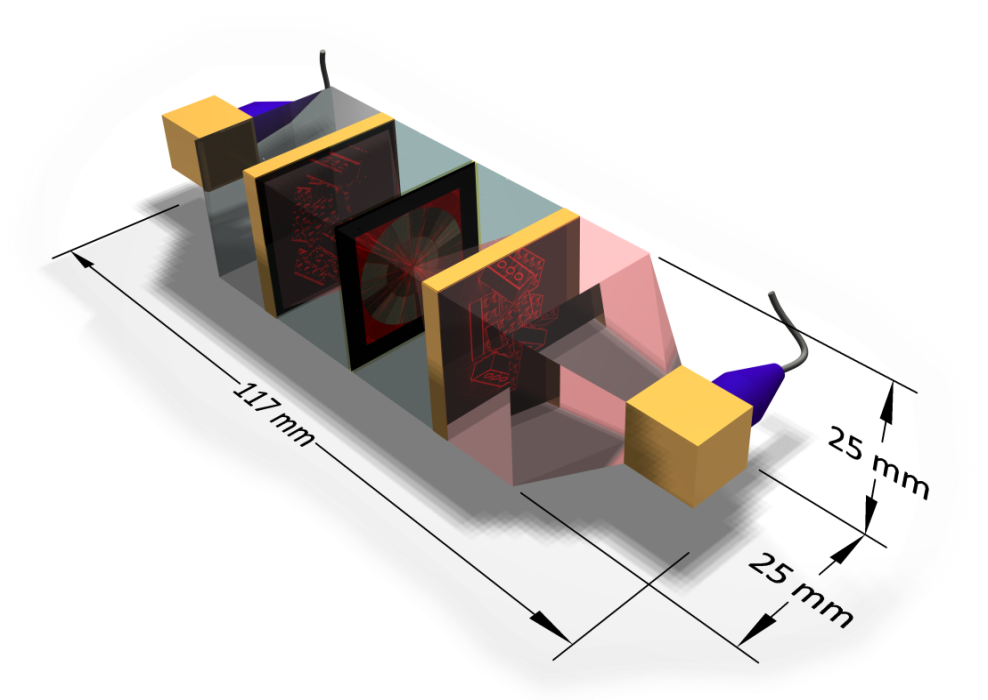
The figure below is a redrawn version of the left side of Figure 4 and shows a block diagram of the neural network algorithm to be implemented in this project. The algorithm consists of three optical convolutions (three squares), a summation, and a non-linear activation function. Each of the functions shown in the block diagram are described in the sections below.

****

**Figure 5. Block Diagram of the Algorithm Implemented for this Project**

**3.1.1 Hardware Background**

Figure 6 shows the construction of Look Dynamic’s PNN device. The PNN consists of two identical Sensay (a portmanteau of "Sensor" and "Display") devices. Each Sensay device has both a display (modulator) and a sensor at each pixel. In a Sensay, pixels are called Trixels (a "Transmit-Receive pIXEL"). Each Trixel also has analog memory and processing elements as well as the ability to interconnect and exchange data with its neighbors. A third device called Antilles resides between the Sensays. Two light sources are located at each end of the architecture and provide light that is modulated by each Sensay.



Sensay B

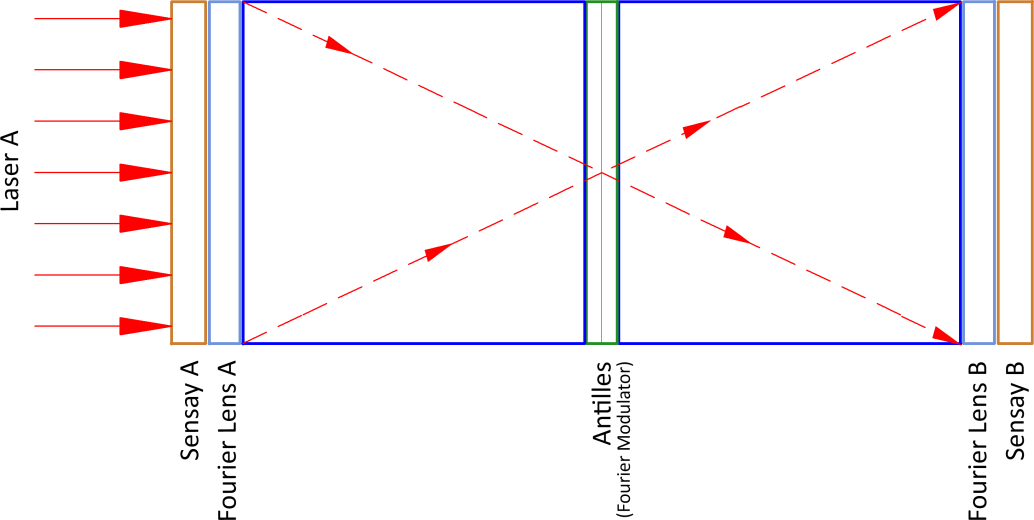
Sensay A

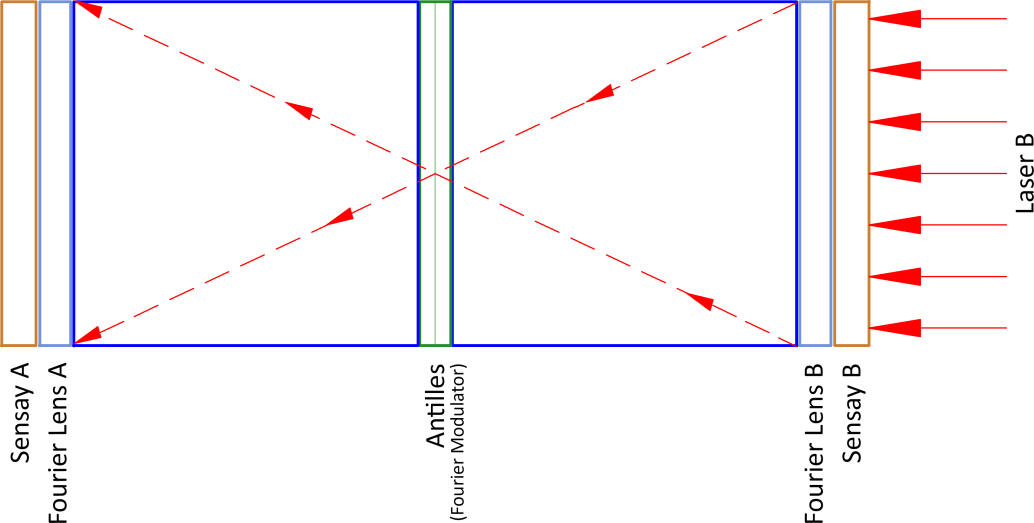
Antilles

**Figure 6. PNN Module Construction**

The Antilles provides filtering at the Fourier plane of each Sensay. Each Sensay has a Fourier lens (not shown) bonded to its surface, so any modulated (displayed) data converts to its Fourier equivalent at the Antilles. It is important to note in the following description of system operation that the explanation applies to all Trixels in a Sensay device; the whole array operates in lockstep and all Trixels operate simultaneously in parallel. The team will be using Look Dynamic’s PNN design to emulate and predict its operation for reservoir processing and application to the HGV application.

Modulating the light which passes through Antilles, using a programmable pattern suitable for a circular geometry, allows virtually any desired convolution. Figure 7 shows a diagram of the system layout. Illuminating light shines through Sensay A (silicon is transparent at the wavelength used), is Fourier-transformed by a lens, is modulated by some pattern at Antilles, inverse-Fourier transformed by a second lens, and is sensed by Sensay B operating in sensor mode. Essentially the system simply focuses an image from a display (Sensay A) to a sensor (Sensay B), imposing a convolution at Antilles as the light naturally passes through a Fourier plane. The hardware is sufficient for almost any Deep Learning architecture including ResNet, Inception, and VGG families. As described, it can also be configured to implement Reservoir Computing.

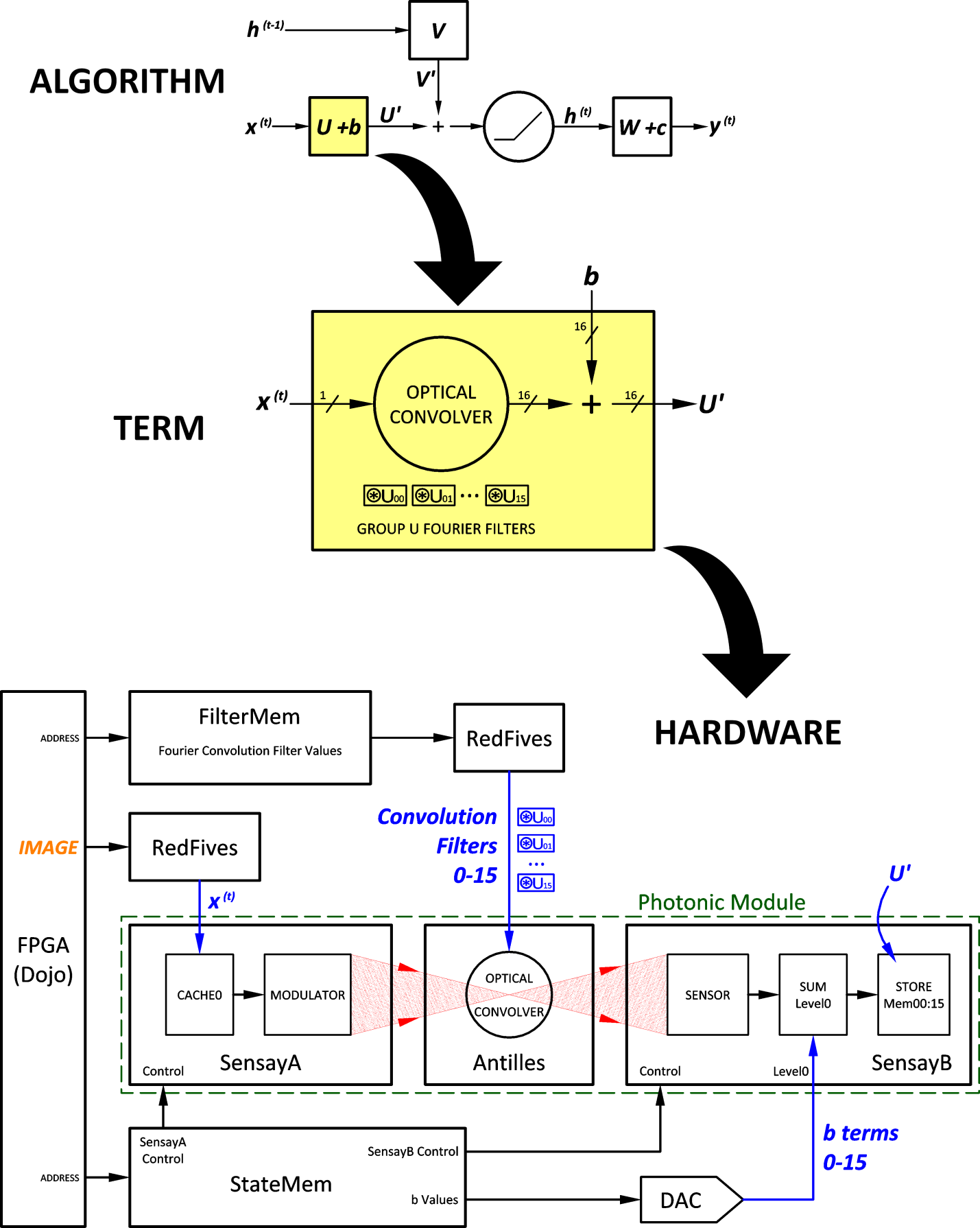




**Figure 7. Bidirectional Convolutional Optical Computing**

**3.1 Hardware Sequences**

Figure 8 shows an expanded view of the U+b operation (entire processing shown in Figure 5), which as described below, is implemented using Look Dynamic’s compact and low power PNN implementation.

****

**Figure 8. Step 1 - Calculate U' Term**

In this step the system ingests a frame of image data, X(t), through an FPGA. The FPGA passes the information to RedFive devices, which convert the digital data from the FPA/FPGA to a PWM signal. This pulse width modulated data is stored in the SensayA device on capacitors. There is a capacitor for each pixel in the image. The charge on each capacitor corresponds to the pixel value received by the FPA. The image data is loaded 48 pixels at a time into the cache capacitors in SensayA. When a full frame of pixels has been loaded, the charge on each capacitor controls the modulator on each Trixel of the SensayA device and lets through an amount of light proportional to the corresponding received pixel value on FPA.

Also during Step 1, data corresponding to the U filters is loaded into the Antilles device. As with the image data, RedFive devices are used to convert the digital data stored in the FPGA to PWM data. The U data are the filters that are convolved with the image data X(t). Figure 8 shows that 16 different convolutional filters, each consisting of 127 Antilles filter coefficients, are loaded. A separate channel is provided for each of the 127 coefficients. For each filter loaded, SensayB records the result of the convolution. During Milestone 3, the team will evaluate if 16 filters is appropriate for the current application and also if 127 filter coefficients are required. The architecture allows either of these parameters to be changed and optimized for an application.

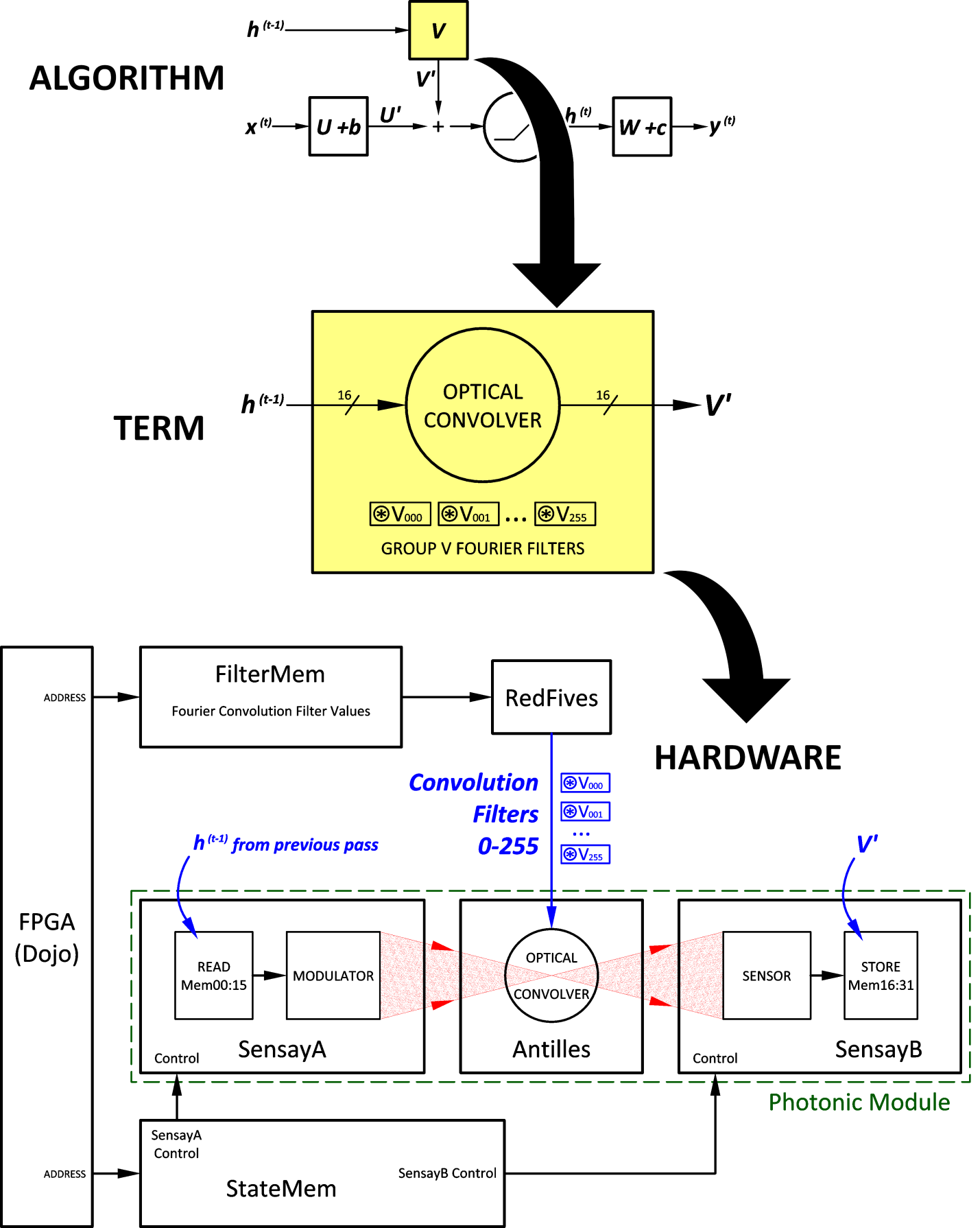
For both the image data and the Antilles coefficients, many systems would choose to use a digital-to-analog converter (DAC) to drive the analog components. However, processing using a PWM approach has the advantages of lower power (no DACs required), more precision can be obtained, and the process is much faster.

As shown in Figure 8, the convolution results are summed to a series of voltages, the "b" terms, and the resulting U' term, a series of full-resolution arrays, are stored in SensayB memory locations 0 through 15 at each Trixel. At the end of this Step, U' resides in SensayB memory locations 0 through 15.

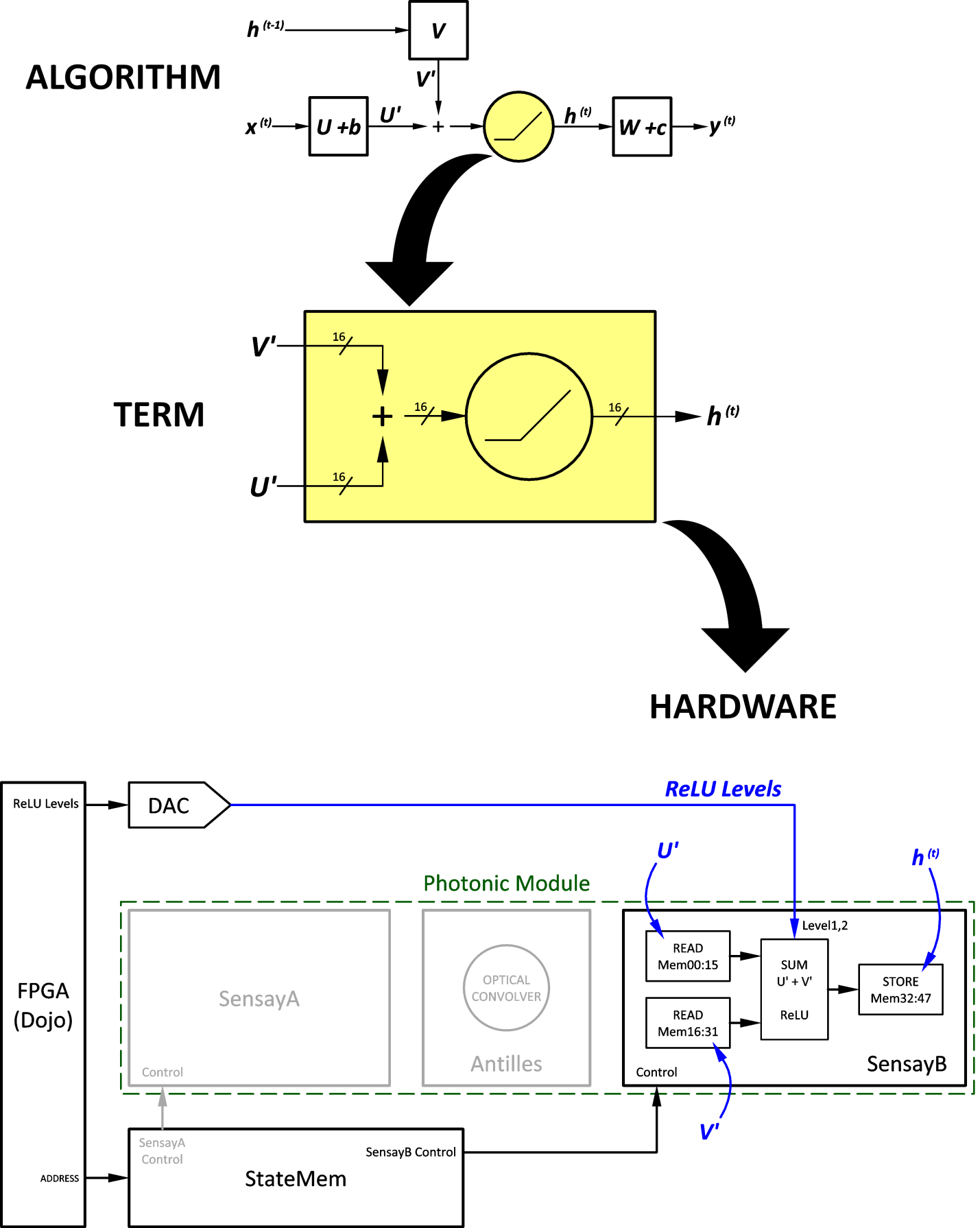
Figure 9 shows the diagram for Step 2, which calculates the V’ term. In this step, the previous h(t-1) terms stored in the previous pass (as described in Step 4), are read from SensayA memory locations 0 through 15 at each Trixel. Each frame of the h(t-1) information is sent to the modulator to produce a Fourier product at Antilles, where it is convolved against a set of sixteen Fourier filters. Since there are sixteen h(t-1) terms and each term is processed by sixteen filters, there are 256 convolutions total.

The resulting inverse-Fourier transformed terms are sensed and summed at SensayB where the results are stored in memory locations 16 through 31. Note that the previous U' terms are in different memory locations and are not affected by the operation of Step 2. At the end of Step2, V' resides in SensayB memory locations 16 through 31.

The approach described above goes back one frame in time, but during Milestone 3 the team will use simulations to determine if there is benefit to going back more steps in time. If there is benefit, the operation of the PNN can be changed without any significant impacts. No hardware changes are required, latency will not be impacted, the only change will be how memory locations in the Sensay devices are utilized.

****

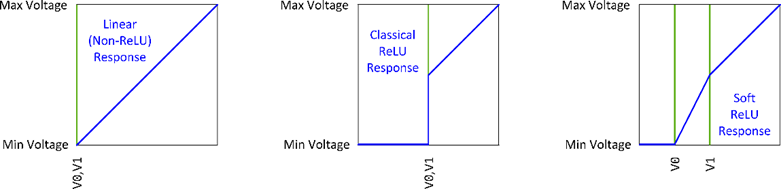
**Figure 9. Step 2 - Calculate V' Term**



**Figure 10. Step 3 - Calculate the h Term**

Figure 10 shows the operation of Step 3, where the U' and V' terms residing in SensayB memory are read, summed, and a non-linear activation function is applied to the sum. The non-linear activation function is commonly called the Rectified Linear Unit (ReLU).

Two external analog levels set the threshold and response of the ReLU function. Rectified Linear Units (ReLU) are often applied to data to suppress weak responses. The Sensay device has a flexible dual-slope ReLU implementation which can result in a variety of responses shown in Figure 11, ranging from no effect (Example A) to a traditional cutoff (Example B) to a variable-slope cutoff (Example C). Two selected analog voltages (typically external voltages driven by DACs) control the transfer function.

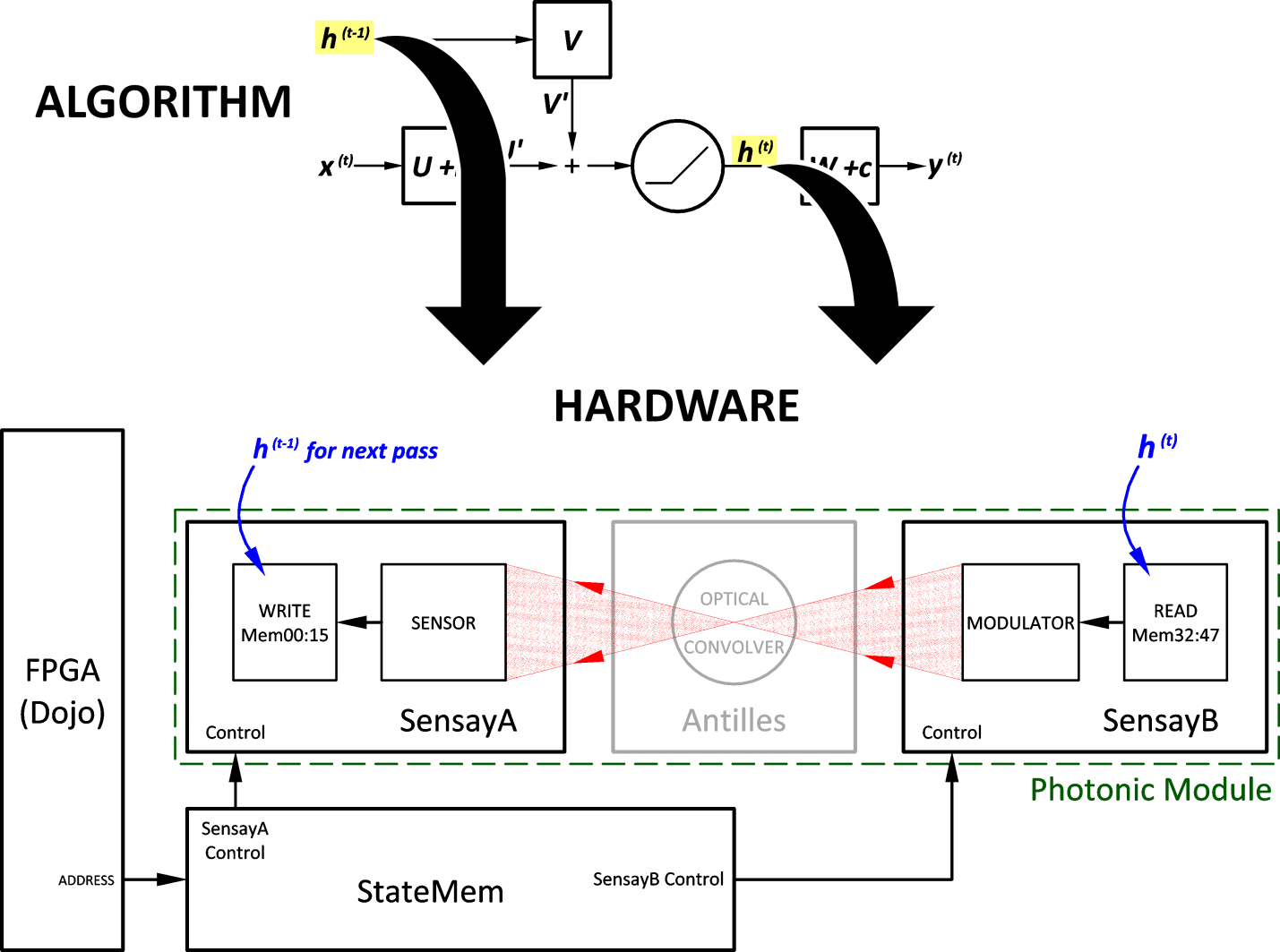


**Figure 11. Examples of ReLU Function Using Two Threshold Levels**

The result, h(t), is written to SensayB memory. At the end of this step, h(t) resides in SensayB memory locations 32 through 47. Note that the operations of summation and ReLU are part of the same dataflow. They are simply wired in series at each trixel and executed in a single analog cycle.

Figure 12 shows the operation of Step 4. In this step, the h(t) terms are copied to SensayA. Note that Antilles is "OFF" so the values are unchanged; this is simply the fastest way to copy the values to SensayA where they are needed for Step 2 on the next pass.

At the end of this Step, h(t-1) resides in SensayA memory locations 0 through 15. As noted above, the team will analyze the number of steps back in time that are appropriate for the HGV detection and tracking application. A larger step backward in time will actually reduce the latency because a smaller number of h(t-1) frames will be utilized for each time step. However, as shown in Section 3.2, the latency is dominated by Step 1, so while the latency will decrease if more steps back in time are used, the decrease will not be significant.



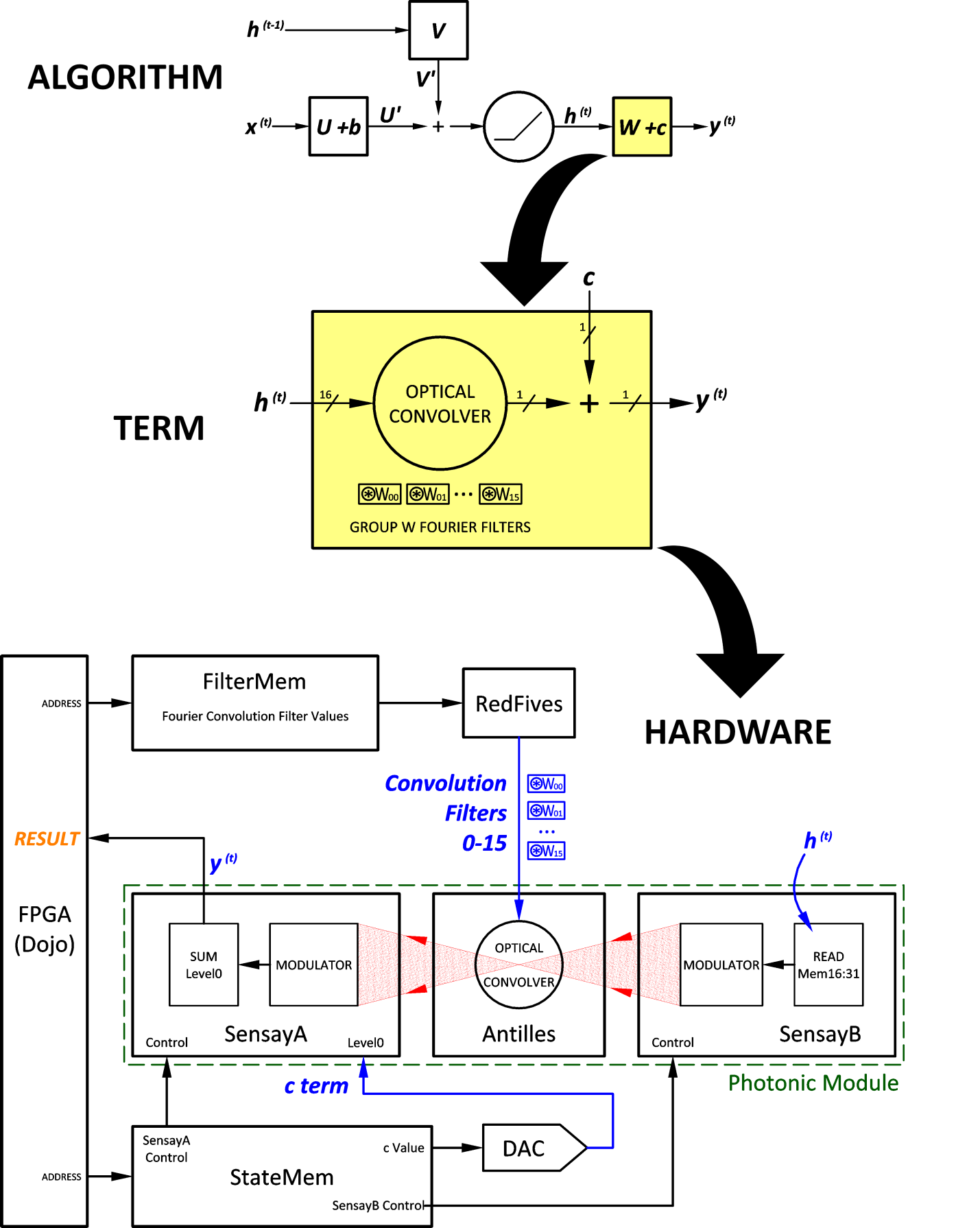
**Figure 12. Step 4 – Copy h Terms to SensayA**

Figure 13 shows the operation of Step 5. During Step 5, the final output frame, y(t), is created.

The array h(t) is sent by SensayB, now operating as the modulator with LaserA off and LaserB on, through a lens, producing a Fourier transform at Antilles. Sixteen filter sets, denoted W, are applied and the convolved result is sensed by SensayA.

The results are then summed to the "c" term, a series of sixteen analog voltages, and the resulting frame, y(t), is output from the digital level interface back to the FPGA.

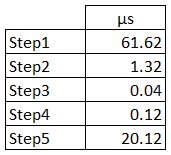
At this point, another frame of data is input to the PNN and the operation begins over again at Step 1.



**Figure 13. Step 5 – Calculate the y Term, the Result**

**3.2 Performance**

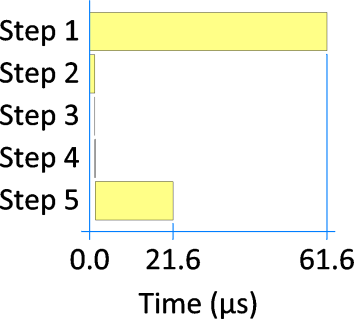
The LD-MU team has analyzed the time required for the steps described above. The approximate duration for each step is shown in the table below.



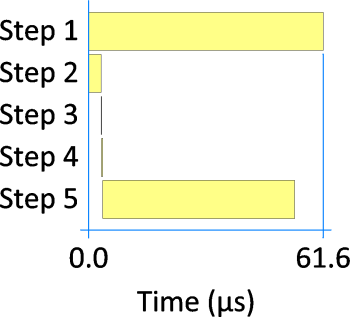
Step 1, Image Load, dominates the timing. Note that because Step 1 just loads values into the SensayA cache, the Sensay and Antilles devices are not used during this step and therefore Steps 2-5 can be run concurrently with Step 1.

**3.2.1 Fastest Frame Rate**

Run concurrently, the full-speed timing diagram is:



Clearly Steps 2 through 5 spend the majority of their time waiting for the next image. Since these steps do not need to run at full speed and since the PNN sections are independent, a power-optimized setting can be used where Steps 2-5 are run slower to reduce power consumption. When Steps 2-5 are run at 40% of the maximum possible speed, the timing is:



Optimized Power Fastest Frame Rate Mode

Steps 2-5 still finish before Step 1, but the overall power use is now about 70% of that used at the full-speed setting while the overall cycle time remains the same.

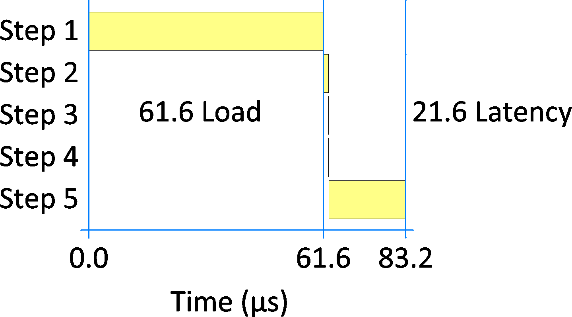
The ingest rate for 768x768 pixel monochromatic images for this system is about 62 μs or 16 kHz. This frame rate is much faster than the expected frame rate for the HGV application, but the processing speed of Look Dynamic’s PNN could be used to advantage in many other applications.

Latency in this mode is two frame times, or about 124 μs, for the PNN module. In addition, there will be two more frame times involved in the system digital upload and download, so the latency is four frame times total, or about 248 μs is expected. This low latency is expected to not be significant for almost all applications of this technology. However, an even lower latency mode is described in Section 3.2.2.

Power for the PNN module and its interface in this mode, including the FPGA but not including the external electronics needed to upload and download images, is expected to be well under 1 W.

**3.2.2 Lowest Latency Mode**

The fastest frame rate does not offer the lowest latency since there is a minimum two-frame pipeline in the PNN. In most cases the data source will be much slower than 16,000 frames/second, so a different operating mode might be preferable. The figure below shows the timing for the lowest latency mode.



Lowest Latency Mode

This is still a frame rate of 12 kHz so it is likely overkill.

However the system can work in "burst mode". For instance, suppose a camera feeding images at 120 Hz. Every 8.3 ms a new frame can be introduced and results can be retrieved 21.6 μs after the load is finished. The average duty cycle in this mode is approximately 1%, so the power usage will drop to about ten milliwatts.

**3.2.3 Direct Feed**

A further consideration regarding rates and power is the idea of direct feed. In some embedded applications, the data from a camera might be in analog form. In this case, it might be possible to forgo the digitization of each frame and feed the analog data directly into the PNN inputs. Although this does not factor into the PNN power usage per se, it does potentially facilitate a lower latency and lower power total solution.

**4. Physical Implementation**

Initial plans for the Antilles modulator employed electro-optic polymers to alter the intensity through different wedges. However, electro-optic polymers require tens of volts to drive the modulation, and exhibit an undesirable temperature dependence, especially relevant during the construction of the Antilles. An alternate approach using graphene has been investigated by the LD-MU team. Recent research in graphene suggests that a graphene-based modulator could use a lower drive voltage and would be relatively insensitive to temperature.

**4.1 Modulator Construction**

The design of the modulator is based on the work presented by the Grigorenko group (Optics Express 2017) with modifications to allow transmissive operation rather than reflective as shown in Fig. X. The bottom layer of the modulator is a wafer of conductive silicon with a thickness of 400 nm. A layer of hafnium oxide is deposited directly on the silicon with a thickness of 200 nm. A single-atom-thick sheet of graphene is deposited on the top of the hafnium oxide layer. Contacts are added to the graphene and conductive silicon layers so that a gating voltage may be applied to drive the modulation. A second layer of hafnium oxide is added on top of the graphene to protect it from damage or wear.

**A picture containing accessory, map

Description automatically generated**

**Figure 14. Achilles Modulator Construction Using Graphene**

**4.2 Modulator Operation**

A single layer of graphene naturally absorbs 2.3% of incident visible and infrared light. The conductive silicon and hafnium oxide layers of the modulator are transparent for 1550 nm wavelength light. With no gating voltage, this absorption will remain unchanged. When a gating voltage is applied, the graphene becomes transparent to the 1550 nm wavelength, where the transparency scales by the voltage applied, typically up to 2-3 volts. This is much more reasonable voltage to achieve in the circuit of the device while still providing an easily tunable grayscale.

Graphene will sometimes show impurities in the form of hydrocarbons, rather than continuous carbon. However, even with impurities in the graphene layer, we expect a minimal effect on the modulation depth. The surrounding controls for each wedge of the Antilles filter can adapt to faulty wedges so that the modulation of all wedges can be normalized to the worst case.

While a 2.3% difference in transmitted intensity seems minimal for modulation, this may be sufficient for the convolution needs of the Antilles. By setting a bias on the sensor to normalize the signal to the intensity of the absorbing graphene, we create a system that effectively tunes from 0% to 100% brightness.

In the circumstance that the fidelity of the sensor-modulator pair is not sufficient, we can increase the absorption of the modulator simply by stacking individual modulators with an insulating layer between them. The Grigorenko group had shown that with each additional layer of graphene ther is approximately a 2.3% increase in absorption (Science 2008). The complexity of the surrounding control circuit does not change; each common component is grouped together.

**4.3 Future work**

One point not yet understood is whether multiple layers of graphene could be used immediately together or if they must remain separated. Being able to use five layers of graphene between the hafnium oxide wafers would not fundamentally change the operation of the modulator but would make fabrication of the Antilles filter simpler.

There are also other possibilities for a modulator that are being explored. Diana Aznakayeva in her doctoral thesis (University of Manchester, 2018) also presented a Fabry-Perot design with an absorption of nearly 40% with a single layer of graphene. This design requires multiple reflections from reflective surfaces and a particular angle that light must enter the modulator, which is not as simple as the normal-incidence modulator presented earlier.

Exploration of these options and other advantageous uses of graphene will continue to be explored and described in more detail in Milestone 3.

**Appendix 1. Matlab Code for Generating HGV Random Tracks**

clear all;

close all;

%Initialization

%FPA and Satellite Parameters

numpix=768; %number of pixels in a tile of the FPA

FPApix=4096; %4096 pixels along side of full FPA

frame=zeros(768,768);

pitch=10E-6; %Pixel pitch size

FOV=120; %Field of view in degrees for a full 4096x4096 array

satalt=800E3; %Satellite altitude

fps=30; %30 frames per second

orbtim=90\*60; %orbit time, 90 minutes times 60 seconds

%Geometry Parameters

Erad=6378E3; %Earth Radius

%Missile Parameters

mach=343; %Speed of sound, 343 m/s

missaltmin=50E3; %Minimum missile altitude

missaltmax=80E3; %Maximum missile altitude

missmachmin=5; %Minimum missile speed in mach

missmachmax=25; %Maximum missile speed in mach

%Initialize Random number generator

rng(4321812); %Seed the random number generator

flen=(FPApix/2)\*pitch/tan((FOV/2)\*pi/180);

misssp=(rand(1,1)\*(missmachmax-missmachmin)+missmachmin)\*mach; %missile speed in m/s

misssp=25\*mach; %set to Mach 10

missalt=rand(1,1)\*(missaltmax-missaltmin)+missaltmin; %miss altitude

missalt=80E3; %set to 80 km above Earth

FPARow=rand(1,1)\*numpix; %Random row on FPA, varies between 0 and numpix

FPACol=rand(1,1)\*numpix; %Random column on FPA

%Start record of FPA row and column positions

FPARowarr(1)=FPARow;

FPAColarr(1)=FPACol;

%Calculate the initial coordinates of the randomly chosen point on the FPA

yFPA=(numpix/2-FPACol)\*pitch; % Note that if one were at the center

% of the Earth looking at the FPA, column 0

% would be on the left side of the FPA

zFPA=(FPARow-numpix/2)\*pitch; % Row zero is at the bottom of the FPA

xFPA=Erad+satalt+flen; %Distance to center of FPA

xFPAarr(1)=xFPA;

yFPAarr(1)=yFPA;

zFPAarr(1)=zFPA;

%Calculate the angle by which the satellite rotates in the x/y plane for

%each increment of time

anginc=(2\*pi/orbtim)\*(1/fps); %in radians

%Start an array of center of FPA location points. Note z is always zero

xFPAlocarr(1)=Erad+satalt+flen;

yFPAlocarr(1)=0;

%Start an array of center of lens location points. Note z is always zero

xLenslocarr(1)=Erad+satalt;

yLenslocarr(1)=0;

%Calculate distance from center of Earth to the edge of the FPA.

%\*\*\*\*\* Note the line below is the same as to center of FPA due to round off

%error \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%OrgFPAEdge=sqrt((Erad+satalt+flen)^2+((numpix/2)\*pitch)^2);

%\*\*\*\*\*\*

%Calculate angle between line from origin to center of FPA and line from

%origin to edge of FPA

FPAAng=atan2((numpix/2)\*pitch,Erad+satalt+flen);

a=(xFPA-(Erad+satalt))^2+(yFPA-0)^2+(zFPA-0)^2;

b=2\*((xFPA-(Erad+satalt))\*(Erad+satalt));

c=(Erad+satalt)^2-(Erad+missalt)^2;

t1=(-b-sqrt(b^2-4\*a\*c))/(2\*a);

t2=(-b+sqrt(b^2-4\*a\*c))/(2\*a);

x1=(Erad+satalt)+t1\*(xFPA-(Erad+satalt));

x2=(Erad+satalt)+t2\*(xFPA-(Erad+satalt));

if x1>x2

t=t1;

else

t=t2;

end

%Now calculate the point on the sphere

xMiss=(Erad+satalt)+t\*(xFPA-(Erad+satalt));

yMiss=t\*(yFPA);

zMiss=t\*(zFPA);

%start building an array of missile locations

xMissarr(1)=xMiss;

yMissarr(1)=yMiss;

zMissarr(1)=zMiss;

%Find theta and phi of spherical coordinates

missPhi=atan2(sqrt(xMiss^2+yMiss^2),zMiss);

missTheta=atan2(yMiss,xMiss);

out\_bound=0;

k=2;

P3R=Erad+missalt;

P3theta=rand(1,1)\*2\*pi;

P3phi=(misssp/fps)/(Erad+missalt);

% Forward Propagation

while out\_bound==0

%Transform to Cartesian coordinates prior to rotation

%See http://tutorial.math.lamar.edu/Classes/CalcIII/SphericalCoords.aspx

xP3=P3R\*sin(P3phi)\*cos(P3theta);

yP3=P3R\*sin(P3phi)\*sin(P3theta);

zP3=P3R\*cos(P3phi);

P3byphi=[cos(missPhi) 0 sin(missPhi); 0 1 0; -sin(missPhi) 0 cos(missPhi)] ...

\*[xP3; yP3; zP3];

%Now rotate by theta about the z-axis

P4=[cos(missTheta) -sin(missTheta) 0; sin(missTheta) cos(missTheta) 0; ...

0 0 1]\*P3byphi;

%P4 is the second missile location ins Cartesian coordinates. Append this to

%the array of missile locations

xMissarr(k)=P4(1);

yMissarr(k)=P4(2);

zMissarr(k)=P4(3);

xMiss=xMissarr(k);

yMiss=yMissarr(k);

zMiss=zMissarr(k);

%Find theta and phi of spherical coordinates

missPhi=atan2(sqrt(xMiss^2+yMiss^2),zMiss);

missTheta=atan2(yMiss,xMiss);

%Now calculate the intersection of the line passing through the new missile

%location and the center of the lens with the plane in which the focal

%plane is located at the second point in time. The following reference

%gives the solution to this problem. http://paulbourke.net/geometry/pointlineplane/

%From this paper, Solution 1 is used and note that N and P3 can be the same

%because the line from the origin to the center of the FPA is normal to the

%FPA. Therefore P3 can be the point where the line from the origin

%intersects the middle of the FPA, and this line is normal to the FPA.

%x/y/zMissarr(2) is the missile location.

%Calculate the Lens and FPA center positions at the second point in time.

%Both of these stay %in the x/y plane, but rotates from the x-axis toward

%the positive y-axis

%online high accuracy calculator: https://keisan.casio.com/calculator

xLenslocarr(k)=(Erad+satalt)\*cos(anginc); %Using online high precision calculator

%answer should be 7177999.994601126323586

%answer obtained is 7177999.994601126

yLenslocarr(k)=(Erad+satalt)\*sin(anginc); %Using online high precision calculator,

%answer should be 278.399408170541340397

%answer obtained is 2.783994081705413

xFPAlocarr(k)=(Erad+satalt+flen)\*cos(anginc); %Using online high precision calculator

%answer should be 7178000.00642525982767

%answer obtained is 7.178000006425260e+06

yFPAlocarr(k)=(Erad+satalt+flen)\*sin(anginc); %Using online high precision calculator

%answer should be 278.3994086291414758

%answer obtained is 2.783994086291415e+02

u2=dot([xFPAlocarr(k) yFPAlocarr(k)],[xFPAlocarr(k)-xMissarr(k) ...

yFPAlocarr(k)-yMissarr(k)])/dot([xFPAlocarr(k) ...

yFPAlocarr(k)],[xLenslocarr(k)-xMissarr(k) ...

yLenslocarr(k)-yMissarr(k)]);

%Using online high precision calculator

%answer should be 1.0000000164121146757917

%answer obtained is 1.000000016412115

%Now calculate the location of second missile location on the FPA

xFPAarr(k)=xMissarr(k)+u2\*(xLenslocarr(k)-xMissarr(k));

%Using online high precision calculator

%answer should be 7178000.006425281828543

%answer obtained is 7.178000006425282e+06

yFPAarr(k)=double(vpa(yMissarr(k))+vpa(u2)\*(vpa(yLenslocarr(k))-vpa(yMissarr(k))));

%Using online high precision calculator

%answer should be 278.3988409197992733505

%answer obtained is 2.783988409197773e+02

zFPAarr(k)=zMissarr(k)+u2\*(0-zMissarr(k));

%Using online high precision calculator

%answer should be -0.00111741285047880422

%answer obtained is -0.001117412888561

%Calculate corresponding row and column on FPA

%Calculate distance between edge of FPA near column 1 and the current point

%on the FPA corresponding to the missile. Calculate coordinates of the

%edge of the FPA near column 1. Note the FPA coordinates

%go from 0 to %numpix, where 0 is one edge and 768 is the other. Pixels

%are centered at 0.5, 1.5, etc.

%The vpa's may not be required

xFPAedge=double(vpa(xFPAlocarr(2))-(vpa(768)/vpa(2)\*vpa(10E-6)) ...

\*sin(vpa(2)\*vpa(pi)/(vpa(90)\*vpa(60)\*vpa(30))));

yFPAedge=double(vpa(yFPAlocarr(2))+(vpa(768)/vpa(2)\*vpa(10E-6)) ...

\*cos(vpa(2)\*vpa(pi)/(vpa(90)\*vpa(60)\*vpa(30))));

% %will need to put check here to see if missile is off edge of FPA

dist=sqrt((xFPAedge-xFPAarr(k))^2+(yFPAedge-yFPAarr(k))^2);

FPAColarr(k)=dist/(pitch);

FPARowarr(k)=zFPAarr(k)/pitch+numpix/2;

if (FPAColarr(k)>=numpix) || (FPARowarr(k)>=numpix)||(FPAColarr(k)<=0) || (FPARowarr(k)<=0)

out\_bound=1;

else

k=k+1;

end

end

t1=(0:(length(FPAColarr)-1))/30;

figure

subplot(211)

plot(t1,FPAColarr,'\*')

title('Forward Propagation');

xlabel('Time (sec)')

ylabel('Column position');

subplot(212)

plot(t1,FPARowarr,'\*')

xlabel('Time (sec)')

ylabel('Row position');

% Back Propagation

xMissarr\_back(1)=xMissarr(1);

yMissarr\_back(1)=yMissarr(1);

zMissarr\_back(1)=zMissarr(1);

xMiss=xMissarr\_back(1);

yMiss=yMissarr\_back(1);

zMiss=zMissarr\_back(1);

xLenslocarr\_back(1)=xLenslocarr(1);

yLenslocarr\_back(1)=yLenslocarr(1);

xFPAlocarr\_back(1)=xFPAlocarr(1);

yFPAlocarr\_back(1)=yFPAlocarr(1);

FPAColarr\_back(1)=FPAColarr(1);

FPARowarr\_back(1)=FPARowarr(1);

xFPAarr\_back(1)=xFPAarr(1);

yFPAarr\_back(k)=yFPAarr(1);

zFPAarr\_back(k)=zFPAarr(1);

out\_bound=0;

k=2;

P3R=Erad+missalt;

P3theta\_back=P3theta;

P3phi\_back=-1\*P3phi;

missPhi=atan2(sqrt(xMiss^2+yMiss^2),zMiss);

missTheta=atan2(yMiss,xMiss);

% Forward Propagation

while out\_bound==0

%Transform to Cartesian coordinates prior to rotation

%See http://tutorial.math.lamar.edu/Classes/CalcIII/SphericalCoords.aspx

xP3=P3R\*sin(P3phi\_back)\*cos(P3theta\_back);

yP3=P3R\*sin(P3phi\_back)\*sin(P3theta\_back);

zP3=P3R\*cos(P3phi\_back);

P3byphi=[cos(missPhi) 0 sin(missPhi); 0 1 0; -sin(missPhi) 0 cos(missPhi)] ...

\*[xP3; yP3; zP3];

%Now rotate by theta about the z-axis

P4=[cos(missTheta) -sin(missTheta) 0; sin(missTheta) cos(missTheta) 0; ...

0 0 1]\*P3byphi;

%P4 is the second missile location ins Cartesian coordinates. Append this to

%the array of missile locations

xMissarr\_back(k)=P4(1);

yMissarr\_back(k)=P4(2);

zMissarr\_back(k)=P4(3);

xMiss=xMissarr\_back(k);

yMiss=yMissarr\_back(k);

zMiss=zMissarr\_back(k);

%Find theta and phi of spherical coordinates

missPhi=atan2(sqrt(xMiss^2+yMiss^2),zMiss);

missTheta=atan2(yMiss,xMiss);

%Now calculate the intersection of the line passing through the new missile

%location and the center of the lens with the plane in which the focal

%plane is located at the second point in time. The following reference

%gives the solution to this problem. http://paulbourke.net/geometry/pointlineplane/

%From this paper, Solution 1 is used and note that N and P3 can be the same

%because the line from the origin to the center of the FPA is normal to the

%FPA. Therefore P3 can be the point where the line from the origin

%intersects the middle of the FPA, and this line is normal to the FPA.

%x/y/zMissarr(2) is the missile location.

%Calculate the Lens and FPA center positions at the second point in time.

%Both of these stay %in the x/y plane, but rotates from the x-axis toward

%the positive y-axis

%online high accuracy calculator: https://keisan.casio.com/calculator

xLenslocarr\_back(k)=(Erad+satalt)\*cos(-anginc); %Using online high precision calculator

%answer should be 7177999.994601126323586

%answer obtained is 7177999.994601126

yLenslocarr\_back(k)=(Erad+satalt)\*sin(-anginc); %Using online high precision calculator,

%answer should be 278.399408170541340397

%answer obtained is 2.783994081705413

xFPAlocarr\_back(k)=(Erad+satalt+flen)\*cos(-anginc); %Using online high precision calculator

%answer should be 7178000.00642525982767

%answer obtained is 7.178000006425260e+06

yFPAlocarr\_back(k)=(Erad+satalt+flen)\*sin(-anginc); %Using online high precision calculator

%answer should be 278.3994086291414758

%answer obtained is 2.783994086291415e+02

u2=dot([xFPAlocarr\_back(k) yFPAlocarr\_back(k)],[xFPAlocarr\_back(k)-xMissarr\_back(k) ...

yFPAlocarr\_back(k)-yMissarr\_back(k)])/dot([xFPAlocarr\_back(k) ...

yFPAlocarr\_back(k)],[xLenslocarr\_back(k)-xMissarr\_back(k) ...

yLenslocarr\_back(k)-yMissarr\_back(k)]);

%Using online high precision calculator

%answer should be 1.0000000164121146757917

%answer obtained is 1.000000016412115

%Now calculate the location of second missile location on the FPA

xFPAarr\_back(k)=xMissarr\_back(k)+u2\*(xLenslocarr\_back(k)-xMissarr\_back(k));

yFPAarr\_back(k)=double(vpa(yMissarr\_back(k))+vpa(u2)\*(vpa(yLenslocarr\_back(k))-vpa(yMissarr\_back(k))));

zFPAarr\_back(k)=zMissarr\_back(k)+u2\*(0-zMissarr\_back(k));

xFPAedge=double(vpa(xFPAlocarr\_back(2))-(vpa(768)/vpa(2)\*vpa(10E-6)) ...

\*sin(vpa(2)\*vpa(pi)/(vpa(90)\*vpa(60)\*vpa(30))));

yFPAedge=double(vpa(yFPAlocarr\_back(2))+(vpa(768)/vpa(2)\*vpa(10E-6)) ...

\*cos(vpa(2)\*vpa(pi)/(vpa(90)\*vpa(60)\*vpa(30))));

% %will need to put check here to see if missile is off edge of FPA

dist=sqrt((xFPAedge-xFPAarr\_back(k))^2+(yFPAedge-yFPAarr\_back(k))^2);

FPAColarr\_back(k)=dist/(pitch);

FPARowarr\_back(k)=zFPAarr\_back(k)/pitch+numpix/2;

if (FPAColarr\_back(k)>=numpix) || (FPARowarr\_back(k)>=numpix)||(FPAColarr\_back(k)<=0) || (FPARowarr\_back(k)<=0)

out\_bound=1;

else

k=k+1;

end

end

t2=((1-length(FPAColarr\_back)):0)/30;

figure

subplot(211)

plot(t2,fliplr(FPAColarr\_back),'\*')

title('Back Propagation');

xlabel('Time (sec)')

ylabel('Column position');

subplot(212)

plot(t2,fliplr(FPARowarr\_back),'\*')

xlabel('Time (sec)')

ylabel('Row position');

t=[t2,t1(2:end)];

figure

subplot(211)

plot(t,[fliplr(FPAColarr\_back),FPAColarr(2:end)],'\*');

title('Whole Path');

xlabel('Time (sec)')

ylabel('Column position');

subplot(212)

plot(t,[fliplr(FPARowarr\_back),FPARowarr(2:end)],'\*');

xlabel('Time (sec)')

ylabel('Row position');

panel=uint8(zeros(numpix,numpix));

row\_index=round([fliplr(FPARowarr\_back),FPARowarr(2:end)]);

column\_index=round([fliplr(FPAColarr\_back),FPAColarr(2:end)]);

for k=1:length(column\_index)

panel(row\_index(k)+1,column\_index(k)+1)=1000;

end

figure

image(panel)

colormap(gray(256));

title('Glider Trajectory');

**Appendix 2. Matlab Code for Updating Tracks in the HGV Detection and Tracking Simulation**

clear all

close all

[numpix, FPApix, pitch, FOV, fps, orbtim,...

Erad, flen, satalt, anginc]=constants;

%Peach Simulation Using Functions

load('StartingPoints.mat');

%Format of Starting Points:

%StartingPoints(1:3); Great circle parameter u

%StartingPoints(4:6); Great circle parameter v

%StartingPoints(7); Parameter t for the Great Circle equation

%StartingPoints(8); snr for this missile (not in dB)

%StartingPoints(9); missile altitude above Earth in meters

%StartingPoints(10); missile speed in m/s

%StartingPoints(11); FPA row position of missile

%StartingPoints(12); FPA column position of missile

%StartingPoints(13); Missile x location

%StartingPoints(14); Missile y location

%StartingPoints(15); Missile z location

miss=[];

%

% %Choose a random row of the starting points (initially there is only 1 row)

% row1=randi(1,length(StartingPoints(:,1)));

% miss=StartingPoints(row1,:);

%

% % Get initial frame

% frame=noiseinFPA(numpix);

% frame=addMissilestoFPA(miss, frame);

%Begin iteration

% while length(miss(:,1)>0) %make sure one missile is still in FOV

% for i2=1:3

% miss=updatemiss(miss,fps,Erad,satalt,orbtim,numpix,pitch,anginc,flen); %Update missile location in space and on FPA

% frame=noiseinFPA(numpix); %Generate noise for next frame

% frame=addMissilestoFPA(miss, frame); %Add missiles to frame

% end

% end

function [frame, missout]=update

%missout will be empty at the end of the simulation

global miss numpix pitch fps orbtim ...

Erad flen satalt anginc

if isempty(miss)

% reset and return initial miss and frame

%Choose a random row of the starting points (initially there is only 1 row)

row1=randi(1,length(StartingPoints(:,1)));

miss=StartingPoints(row1,:);

% Get initial frame

frame=noiseinFPA(numpix);

frame=addMissilestoFPA(miss, frame);

missout=miss;

else

miss=updatemiss(miss,fps,Erad,satalt,orbtim,numpix,pitch,anginc,flen); %Update missile location in space and on FPA

frame=noiseinFPA(numpix); %Generate noise for next frame

frame=addMissilestoFPA(miss, frame); %Add missiles to frame

missout=miss;

end

end

function [numpix, FPApix, pitch, FOV, fps, orbtim,...

Erad, flen, satalt, anginc]=constants

%Initialization

%FPA and Satellite Parameters

numpix=768; %number of pixels in a tile of the FPA

FPApix=4096; %4096 pixels along side of full FPA

pitch=10E-6; %Pixel pitch

FOV=120; %Field of view in degrees for a full 4096x4096 array

satalt=800E3; %Satellite altitude

fps=30; %30 frames per second

orbtim=90\*60; %orbit time, 90 minutes times 60 seconds

%Geometry Parameters

Erad=6378E3; %Earth Radius

%Missile Parameters

% mach=343; %Speed of sound, 343 m/s

% missaltmin=50E3; %Minimum missile altitude

% missaltmax=80E3; %Maximum missile altitude

% missmachmin=5; %Minimum missile speed in mach

% missmachmax=25; %Maximum missile speed in mach

%Initialize Random number generator

rng(4321812); %Seed the random number generator

%Calculated Parameters

%Initially assume a tile in the middle of the FPA

%Find the distance from the center of the lens, which is assumed to be

%along the x-axis, to the center of the FPA. This is the focal length

%of the lens. Note that the center of the lens is at the altitude of the

%satellite.

flen=(FPApix/2)\*pitch/tan((FOV/2)\*pi/180);

% misssp=(rand(1,1)\*(missmachmax-missmachmin)+missmachmin)\*mach; %missile speed in m/s

% misssp=10\*mach; %set to Mach 10

% missalt=rand(1,1)\*(missaltmax-missaltmin)+missaltmin; %miss altitude

% missalt=80E3; %set to 80 km above Earth

anginc=(2\*pi/orbtim)\*(1/fps);

end

function frame=noiseinFPA(numpix)

frame=1000\*randn(numpix,numpix)+2^15; %Standard Deviation of 1000, Mean 32678

end

function frame=addMissilestoFPA(StartingPoints,frame)

for i1=1:length(StartingPoints(:,1))

frame(round(StartingPoints(i1,11)),round(StartingPoints(i1,12)))= ...

frame(round(StartingPoints(i1,11)),round(StartingPoints(i1,12)))+ ...

sqrt(StartingPoints(i1,8)\*1000)\*randn(1,1)+StartingPoints(i1,8)\*1000;

end

end

function miss=updatemiss(miss,fps,Erad,satalt,orbtim,numpix,pitch,anginc,flen)

for i1=1:length(miss(:,1)) %Loop to include all missiles

%Update t (corresponds to frame number, with frame zero at originally

%chosen random point

miss(i1,7)=miss(i1,7)+1;

%Move missile

miss(i1,13:15)=(Erad+miss(i1,9))\*(miss(i1,1:3)\* ...

cos(miss(i1,7)\*(miss(i1,10)/fps)/(Erad+miss(i1,9))) ...

-miss(i1,4:6)\*sin(miss(i1,7)\*(miss(i1,10)/fps)/(Erad+miss(i1,9))));

%Calculate the Lens and FPA center positions

xLensLoc=(Erad+satalt)\*cos(miss(i1,7)\*anginc);

yLensLoc=(Erad+satalt)\*sin(miss(i1,7)\*anginc);

xFPALoc=(Erad+satalt+flen)\*cos(miss(i1,7)\*anginc);

yFPALoc=(Erad+satalt+flen)\*sin(miss(i1,7)\*anginc);

%Calculate missile position on FPA in x, y, z coordinates

%xFPALoc

%yFPALoc

u2=dot([xFPALoc yFPALoc],[xFPALoc-miss(i1,13) ...

yFPALoc-miss(i1,14)])/dot([xFPALoc ...

yFPALoc],[xLensLoc-miss(i1,13) ...

yLensLoc-miss(i1,14)]);

xFPA=miss(i1,13)+u2\*(xLensLoc-miss(i1,13));

yFPA=miss(i1,14)+u2\*(yLensLoc-miss(i1,14));

zFPA=miss(i1,15)+u2\*(0-miss(i1,15));

%Calculate corresponding row and column on FPA

if miss(i1,7)<0

xFPAedge=xFPALoc+(numpix/2\*pitch) ...

\*sin(miss(i1,7)\*2\*pi/(orbtim\*fps));

else

xFPAedge=xFPALoc-(numpix/2\*pitch) ...

\*sin(miss(i1,7)\*2\*pi/(orbtim\*fps));

end

yFPAedge=yFPALoc+(numpix/2\*pitch) ...

\*cos(miss(i1,7)\*2\*pi/(orbtim\*fps)); % This is the upper edge while the

%FPA is in quadrants I and IV of the x and y plane.

%The simulation is constrained to not allow the FPA

%to propagate out of these quadrants.

yFPAedgelow=yFPALoc-(numpix/2\*pitch) ...

\*cos(miss(i1,7)\*2\*pi/(orbtim\*fps)); %This is the lower FPA edge in the x/y plane

if yFPA>yFPAedge || yFPA<yFPAedgelow %check to see if the

%missile is off the edge of the FPA

miss=miss([1:i1-1 i1+1:length(miss(1,:))],:); %Eliminate current row

else

dist=sqrt((xFPAedge-xFPA)^2+(yFPAedge-yFPA)^2);

miss(i1,12)=dist/(pitch);

miss(i1,11)=zFPA/pitch+numpix/2

temp1row=miss(i1,11)

temp2col=miss(i1,12)

keyboard

end

end

end