NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TRACKING HIGH-SPEED PROJECTILES WITH AN EVENT-BASED PIPELINE

by

Andrea Hashimoto and Yolanda E. Gutierrez

March 2022

Thesis Advisor:	Vinnie Monaco
Second Reader:	Mathias N. Kolsch

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2022	3. REPORT TYPE AND DATES COVERED Master's thesis

4. TITLE AND SUBTITLE TRACKING HIGH-SPEED PROJECTILES WITH AN EVENT-BASED PIPELINE	5. FUNDING NUMBERS RCPQB
6. AUTHOR(S) Andrea Hashimoto and Yolanda E. Gutierrez	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Reconnaissance Office, Virginia, 20151	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

The potential accuracy and reliability of event-based pipelines in light of their low-resource and lightweight physical demands make them promising candidates for critical systems with strict environmental constraints. The research done for this paper is intended to expand on event-based pipelines as an optimal means of tracking high-speed projectiles in real time. Time intervals between spikes in a neural network can be implemented in such a way that linear mathematical functions are predictable, as shown by Xavier Lagorce and Ryad Benosman's 2015 paper, "STICK: Spike Time Interval Computational Kernel, A Framework for General Purpose Computation using Neuron, Precise Timing, Delays and Synchrony." However, there has yet to be research on predicting non-linear functions with this method. In this work, an event-based sensor is used to gather high-speed projectile data, which is then processed to determine the optimal parameters for the ballistic equations. The specific spiking neural network is designed and integrated for further implementation in STICK. While smaller components of the ballistic functions are still necessary for the complete functionality of a STICK implementation to be applied to trajectories, this work provides proof of concept that the combination of these two technologies has the capability to allow for trajectory tracking without the current operational cost, constraints, and larger scale requirements of other current tracking techniques.

14. SUBJECT TERMS computer science, neural network, event-based, Dynamic Vision Sensor, STICK	15. NUMBER OF PAGES 95
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

TRACKING HIGH-SPEED PROJECTILES WITH AN EVENT-BASED PIPELINE

Andrea Hashimoto
Civilian, CyberCorps: Scholarship For Service
BS, University of California - Santa Barbara, 2016
BA, University of California - Santa Barbara, 2016

Yolanda E. Gutierrez
Lieutenant Commander, United States Navy
BS, Old Dominion University, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2022

Approved by: Vinnie Monaco
Advisor

Mathias N. Kolsch
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The potential accuracy and reliability of event-based pipelines in light of their low-resource and lightweight physical demands make them promising candidates for critical systems with strict environmental constraints. The research done for this paper is intended to expand on event-based pipelines as an optimal means of tracking high-speed projectiles in real time. Time intervals between spikes in a neural network can be implemented in such a way that linear mathematical functions are predictable, as shown by Xavier Lagorce and Ryad Benosman's 2015 paper, "STICK: Spike Time Interval Computational Kernel, A Framework for General Purpose Computation using Neuron, Precise Timing, Delays and Synchrony." However, there has yet to be research on predicting non-linear functions with this method. In this work, an event-based sensor is used to gather high-speed projectile data, which is then processed to determine the optimal parameters for the ballistic equations. The specific spiking neural network is designed and integrated for further implementation in STICK. While smaller components of the ballistic functions are still necessary for the complete functionality of a STICK implementation to be applied to trajectories, this work provides proof of concept that the combination of these two technologies has the capability to allow for trajectory tracking without the current operational cost, constraints, and larger scale requirements of other current tracking techniques.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**

2 Background **5**

2.1 Event-based Vision Sensors and the Biological Eye 5

2.2 Neurons and SNNs 8

2.3 Current Solutions and Problems 14

2.4 Next Steps 15

2.5 Research Intentions 19

3 Experiment Design **21**

3.1 Materials and Devices 21

3.2 Cleaning Methods 26

4 Methodology **31**

4.1 Programming Environment 31

4.2 Importing Data from AEDAT4 32

4.3 Applying a Moving Median 35

5 Implementation **39**

5.1 Ballistic Equations 39

5.2 Curve Fit 42

5.3 STICK Framework Design 44

6 Results and Analysis **57**

6.1 Results 57

6.2 Proportional Error to Time 61

6.3 Suitability of DVS 63

6.4 Precise Timing Computation 64

7 Conclusion and Future Work **71**
7.1 Overall Conclusions 71
7.2 Continuing Work 72

List of References **75**

Initial Distribution List **77**

List of Figures

Figure 2.1 Light inverted on the retina 6

Figure 2.2 Biological eye anatomy and function 6

Figure 2.3 Biological neuron anatomy 9

Figure 2.4 Membrane potential summation in the soma 10

Figure 2.5 Action potential of a neuron 11

Figure 2.6 Spiking Neural Network (SNN) architecture 13

Figure 2.7 von Neumann Architecture 15

Figure 3.1 Front view of the DAVIS346 camera without the lens. 22

Figure 3.2 Visual comparisons between the Dvsnoisefilter, Knoisefilter, and Ynoisefilter outputs. 23

Figure 3.3 Example of the module workflow in the Dynamic Vision (DV) graphical user interface (GUI). 24

Figure 3.4 Catapult model used for data collection 25

Figure 3.5 Workflow of the Dvsnoisefilter and accumulator module applied . 27

Figure 3.6 Workflow of the Dvsnoisefilter, Ynoisefilter, and accumulator module applied 28

Figure 4.1 Scatter plot: Raw data without noise 33

Figure 4.2 Scatter plot: Adjusted orientation 34

Figure 4.3 Moving median vs. moving average 36

Figure 4.4 Scatter plot: Clean trajectory 37

Figure 5.1 Scatter plot: Horizontal component 40

Figure 5.2 Scatter plot: Vertical component 41
Figure 5.3 Neuron model from Lagorce and Benosman 45
Figure 5.4 Types of synapses listed by Lagorce and Benosman 46
Figure 5.5 Inverted memory network by Lagorce and Benosman 47
Figure 5.6 Synchronizer network by Lagorce and Benosman 48
Figure 5.7 Multiplier network 49
Figure 5.8 Subtractor network 51
Figure 5.9 STICK framework for the horizontal component 55
Figure 5.10 STICK framework for the vertical component 56
Figure 6.1 Comparisons of Error Accumulation in Scatter Plots from Various Trials 58
Figure 6.2 MSE Result for a single trial. 59
Figure 6.3 RMSE Result for a single trial 60
Figure 6.4 Change in RMSE for x over time. 62
Figure 6.5 Change in RMSE for y over time. 63

List of Tables

Table 5.1 Covariance matrix 44

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

CPS cyber physical system

DV Dynamic Vision

DVS Dynamic Vision Sensor

FOV frame of view

GUI graphical user interface

IR infrared

ISS International Space Station

LIF Leak-Integrate-Fire

LRDR Long Range Discrimination Radar

NAN not a number

SciPy Scientific Python

SNN Spiking Neural Network

STDP spike-timing-dependent plasticity

STICK spike time interval computational network

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

To my dear, loving husband, thank you for always supporting my needs and being the camel that rode the waves of military life, navigating the seas at my side. I appreciate your unwavering dedication and your never-ending positivity. To my wonderful, sweet children, I would have never made it through this program without you guys being champions and pulling through to support me in my darkest hours. I never felt alone or unloved throughout this process.

To Dr. Vincent Monaco, thank you for your support and mentorship throughout this process. This project was challenging and required your patience and understanding. Thank you for all the times you explained things to us multiple times.

Last, I would like to thank Andrea Hashimoto. This project and thesis work was challenging to say the least. We would have never made it to the finish line without you playing taskmaster. I hope we end up working together again in the future.

-LCDR Yolanda Gutierrez, United States Navy

A huge thank you to my family who was supportive, patient and encouraging throughout my master's degree. Thank you to the friends who have sparked ideas and kept me persevering. A special thanks to Dr. Vinnie Monaco, who gave endless time and support throughout the process with more patience than I have ever encountered from a professor. This work would not be possible without your guidance and ingenuity.

Yolanda, you will forever hold a special place in my life and carry the title of Thesis Partner, a title that no one else dared to shoulder. Without your fun-loving spirit and can-do mentality, I would not have completed this thesis. Thank you.

- Andrea Hashimoto

This material is based upon activities supported by the National Science Foundation under Agreement No 1565443. Any opinions, findings, and conclusions or recommendations

expressed are those of the author and do not necessarily reflect the views of the National Science Foundation.

CHAPTER 1:
Introduction

The performance of cyber physical systems relies on the ability to be operational and, thus, protected from physical attacks. Limiting adversaries' accessibility and preventing physical attacks on these systems protects the availability and integrity of the data within. One concern is protecting an asset from threats moving at high speeds where fine temporal resolution is imperative to intercepting or avoiding these threats. For instance, spacecraft outside of our atmosphere rely on high-speed surveillance to navigate a path with minimal collisions. Consider satellites that send and receive critical communication signals for our nation in orbit with fragments of debris; the satellites' ability to avoid collisions is a large factor in their overall lifespan.

Additionally, our nation's threat detection networks rely on the ability to accurately detect, process, and track targets in motion. Current hardware and software implementations of tracking systems are weighed down by the computational overhead created by frame-based vision sensors and the processing constraints imposed by traditional computational architecture. The current disconnect between processing and memory creates unmitigated processing delays. This work is designed to show that an event-based Dynamic Vision Sensor (DVS) can be used in place of traditional frame-based sensors for tracking a projectile in motion and the spike time interval computational network (STICK) framework can be utilized in combination with the DVS as a neuromorphic solution to overcome the challenges associated with traditional von Neumann processing methods.

This work provides the building blocks for an event based implementation of a real-time surveillance and response system to ensure the safety and efficacy of cyber physical system (CPS) devices in high-speed environments that are essential to our information and communications. The direction of current development in high-speed tracking techniques is quickly moving toward the use of bio-inspired sensors and processing pipelines. Specifically, the use of frame-less cameras, or event-based sensors, have been found to perform exceptionally well in resource-constrained environments where fine temporal resolution is imperative. While traditional, frame-based cameras record megabytes of redundant, extra-

neous data in every frame, event-based sensors capture only the changes, the pertinent data, during a shot. This selective data capture minimizes unnecessary memory storage as well as power consumption when it comes time to process the input. Mimicking the biological eye, event-based sensors do not record input as frames like traditional cameras do. Rather the events are asynchronously recorded upon demand, only when events occur, which again saves energy, memory, and processing resources.

Consistent with event-based data collection, bio-inspired neural network models are created to mimic the biological neural spikes to compute and process information similarly to the way biological neural systems do. The use of Spiking Neural Network (SNN)s, which mimic the electric pulses propagated across neurons, shows great promise in reducing energy consumption, increasing processing speed, and eliminating the von Neumann bottleneck [1]. More recently, the time interval between spikes has proven to carry data effectively, adding the capability of memory to the spiking neural model [1]. The STICK framework introduced in [1] uses computational units and precise timing to derive a number of both linear and non-linear mathematical operators. The precise timing framework allows the data to be encoded in the time intervals of each spike, thus, enabling the data to be stored and computed by each computational unit on site [1]. This method prevents the necessity of retrieving data from storage entirely, thereby omitting the latency of traditional computer architecture. Not relying on traditional von Neumann architecture all together eliminates the bottleneck created by branching and cache misses, enabling much more parallelism with the data locally available [1]. While Lagorce and Benosman show that time intervals between spikes can be implemented in such a way that linear mathematical functions are predictable with their framework [1], there has yet to be research on predicting non-linear functions with this method. Since the STICK framework is event-based, expressed as spikes, authors suggest effective results of using STICK in combination with event-based vision algorithms [1].

The research done for this paper utilizes an event-based pipeline to show the practicality and efficiencies of processing event data from a DVS using a SNN with a STICK framework. The data collection using a DAVIS346 DVS paired with iniVation's Dynamic Vision (DV) software subsidizes the limited available documentation of its capabilities. Additionally, the models for accurately and consistently predicting the projectile paths are refined for accurate representation using the STICK framework. Subsequently, the logic design for implementing the ballistic models in STICK is proposed for future work.

Aside from the aforementioned benefits of using an event-based sensor, converting events in the world to event-based data is a natural fit for processing with a SNN. Taking event-based input to be processed by an event-based neural network streamlines the complexities of identifying objects in motion and their direction. As such, event-based models are highly energy-efficient and require limited resources, which make SNNs exceptionally suited for space-limited, weight-limited, and power-limited environments. In addition, research [2] shows that the accuracy of SNNs' trajectory predictions compared to those of humans in a similar environment have a consistently lower measure of error. In various lighting conditions, the SNN that Debat et. al. [2] constructed shows more accurate predictions on average as compared to human predictions. The potential accuracy and reliability of event-based pipelines in light of their low-resource demands, make them promising candidates for critical systems with strict environmental constraints.

The structure of this paper is such that the background (Chapter 2) presents the necessary groundwork of event-based sensors and SNNs for a thorough understanding of how these tools were used in this research. The design of the data collection portion of the experiment is outlined in Chapter 3, before the methodology of cleaning and preparing the data for processing is described in Chapter 4. The implementation of curve fitting the ballistics models and defined STICK components of the SNN can be found in Chapter 5. The results of this work and its analysis are discussed in Chapter 6, where the error of the curve fit and integration of the STICK computational units to form the neural network is also covered. Lastly, Chapter 7 elaborates on the expansion and direction of neuromorphic designs in software and hardware alike, where future work is discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

This chapter discusses the foundational research conducted in order to pursue a solution for increasing the speed and accuracy of high-speed projectile detection and tracking. For context, this chapter reviews the basic form and function of the eye and neuron to better understand the similarities between these structures and the bio-inspired technology as well as the benefits that accompany them. More specifically, event-based sensors share many of the same functionalities as the human eye in regard to their ability to capture events asynchronously for processing and their reliance on light. These similarities warrant shared benefits, including maximum efficiency with minimal data collection and transport. Likewise, SNNs are modeled after neurons in the brain that work asynchronously and as sparsely connected networks. Just as the brain receives input from the optic nerve in the eye, SNNs ideally receive input from event-based sensors that inherently send events for processing. Thus, this chapter explains the compatibility between SNNs and event-based sensors for improving tracking of projectiles and how current implementations compare with other projectile tracking techniques.

2.1 Event-based Vision Sensors and the Biological Eye

Bio-inspired vision sensors take on many of the same attributes and most of the same functions as their biological counterpart. The anatomy of the eye provides fundamental similarities with event-based sensors, which results in (three) shared functionalities: (lateral inhibition,) efficient detection of changes and preference for ideal lighting. This section will cover the basic form and mechanisms of the biological eye as well as the advantages that event-based sensors assimilate.

2.1.1 Basic Understanding and Anatomy of the Eye

The convex shape of the lens causes the image of the object to be inverted when light passes through the lens such that the image is projected upside-down when it makes contact with the retina (Figure 2.1). When the light reaches the retina, photoreceptors, known as rods

and cones, are stimulated. Rods and cones, receptors for grey scale in low light and color in ample light, respectively, convert the light stimuli to electric pulses.

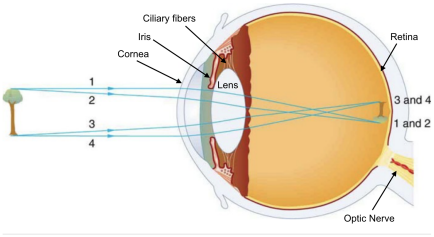


Figure 2.1. Light passes through the cornea, pupil, and lens. The image is inverted as it passes through the lens. Light stream 1 and 2 crosses paths with light streams 3 and 4, which inverts the image when projected on the retina. Source: [3].

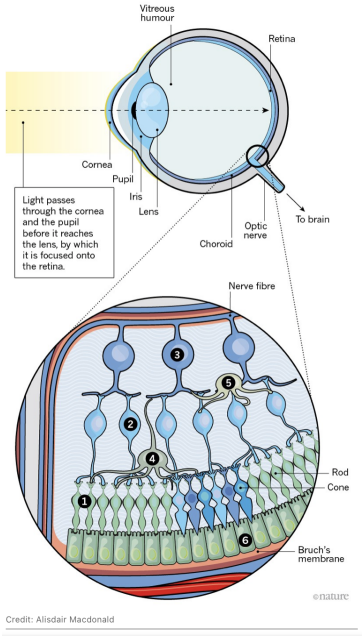


Figure 2.2. The retina includes five types of neurons: photoreceptors (1), bipolar cells (2), ganglion cells (3), horizontal cells (4), and amacrine cells (5). The layer of retinal pigment epithelial cells (6) is not addressed in this paper. Source: [4].

As seen in Figure 2.2, the photoreceptors send the signal to nearby neurons (ganglion, bipolar, horizontal, and amacrine neurons), which consolidate and funnel the signals to the optic nerve, a bundle of 100 million neurons [4]. Given that the electric pulses are sufficient to meet or surpass the thresholds of these neurons, the signal is passed along for the brain to comprehend. Through this relay, a baseball player is able to recognize a ball pitched at 100 mph, to detect, extrapolate, and react to its motion with enough time remaining to fire neurons to trigger muscles to swing the bat appropriately. This chain of events happens so quickly, modern frame-based cameras cannot catch every movement.

2.1.2 Maximum Efficiency

Natural selection has optimized the eye to be only as complex as it has to be to fulfill its function and as efficient as possible to minimize time and resource consumption. By modeling camera technology after this highly optimized structure, similar benefits are reflected in event-based cameras. The photoreceptors in the eye only need to send changes in the environment back to the brain [5] while the static images remain the same and do not require the brain's attention. Similarly, event-based sensors do not record or send static images, as traditional cameras do, but only detect and send new events, events that mark changes in the scene. Unlike frame-based cameras that have set frame rates, each pixel on the event-based sensor is responsible for detecting significant change in its field of view and sending events at the unique rate of that change, not any set rate [5]. This enables dynamic and customized rates of data collection. While the set frame rate of a traditional camera underpolls a high-speed object and overpolls static objects, event-based sensors are triggered by illuminated motion and therefore collect the minimal data required to update the scene.

2.1.3 Illumination Dependency

Since human eyes rely on photons to stimulate the photoreceptors of the retina, they require light for any visual input to be processed. There is an ideal range where objects are fully illuminated and the pupil can adjust to accommodate brightness, but any lower and the cones are no longer able to see color, and any higher all photoreceptors are over stimulated, leaving residual shapes in the field of view (recall looking at the sun and looking away, leaving a residual shape of the sun in your field of view). Analogous to this, event-based

cameras rely heavily on an appropriate amount of light to illuminate moving targets. Without enough light, pixels cannot identify changes in the scene, and with too much light, pixels can be overstimulated, causing them to send false data when a new event is not present. However, as discussed later in Chapter 3, minimal light is required for optimal output. The DVS provides sensitivity settings that can be adjusted to fit the environment, preventing pixels from being overstimulated. Just as photoreceptors in the eye convert light into electric pulses that the brain can easily interpret, the event data from event-based sensors align well with event-oriented processing.

2.2 Neurons and SNNs

In order to better understand the structure of neural networks, and SNNs in particular, basic familiarity of neuron form and function is necessary. An overview of the general role of neurons in the body as well as the more detailed process of signal propagation via action potential are essential to thoroughly understanding the motivations and benefits of using a SNN for processing event-based data.

2.2.1 Neuron Anatomy and Layout

Signals are passed throughout the body by way of the nervous system, which includes the brain, central nervous system, and peripheral nervous system. Neurons are the building blocks of the nervous system and all biological cognition. Receptors, such as photoreceptors for light, chemoreceptors for scents and pheromones, and mechanoreceptors for movements and touch, sense and convert input from the environment to electric impulses that can be propagated through neurons. These neurons are aligned to construct neural pathways. Like a trail system in a forest, the pathways that are used most frequently become dominant [6] and the easiest to use. The less frequented pathways fade with time if not used, causing the organism to forget. Neural pathways are interconnected in patterns throughout the brain that encode information for the brain to recall [6]. Memories, skills, knowledge, associations, all require a specific pattern unique to each brain that retains that information. A pattern of neural pathways lights up in the brain when recalling a face or learning the arc of a ball [6]. The pattern of neural pathways itself encodes the data, much like the spike intervals of the STICK framework does.

As shown in Figure 2.3, there are three basic parts of a neuron: the dendrites, soma, and axon [7]. Dendrites are receivers that grow from the soma; they receive signals from other neurons via a synapse and pass that information to the soma. The soma is the cell body, the core of the neuron where the neuron's structure and cellular functions are maintained. More pertinently, the soma is responsible for the accumulation of membrane potential. The axon is the transmitter of the neuron. Axons can be as long as 1 meter in length, including, for instance, the sciatic nerve that runs down the length of the leg [6]. Synapses, or neural junctions, are the small gaps of space between neurons. The neuron sending the signal across a synapse is called the pre-synaptic neuron, and the neuron on the receiving side of the synapse is called the post-synaptic neuron.

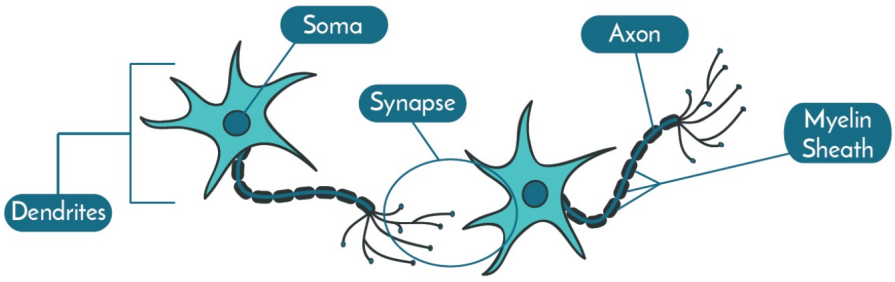


Figure 2.3. Essential anatomy of biological neurons. Source: [7].

2.2.2 Action Potential and Signal Propagation

Since the spiking in SNNs is very closely modeled after the action potential in biological neurons, discussing process of signal propagation through an axon will help to better explain the spiking behavior in neural networks. Starting from the dendrites of the pre-synaptic neuron, we will follow signal propagation down the axon until it is passed to the post-synaptic neuron where the relay continues.

Typical neurons start in a resting state with a membrane potential of -70mV [8]. Each dendrite may receive inhibitory or excitatory signals, which decrease or increase the membrane potential, respectively [9]. Neurotransmitters are chemicals sent across synapses to carry signals from the pre-synaptic neuron to the post-synaptic neuron. As the dendrites receive

neurotransmitters from the axon terminals of other neurons across the synapses, the signals accumulate in the soma. Figure 2.4 shows the summation of excitatory and inhibitory signals in the soma that trigger the action potential. As seen by the voltage graph in Figure 2.4, the action potential, or spike, is not generated until the voltage threshold is met. Once the threshold is met, the action potential is then propagated down the length of the axon to the axon terminal. After the signal travels down the axon to the axon terminal, neurotransmitters are activated to transmit the signal across the synapse to the dendrites of nearby neurons.

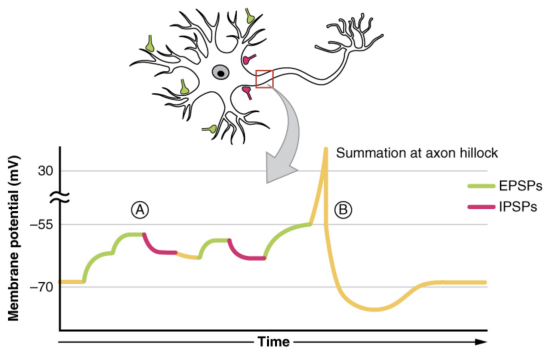


Figure 2.4. The three excitatory signals raise the membrane voltage while the two inhibitory signals lower the voltage. When the sum of these signals in the soma reach -55mV, shown by the threshold, the action potential occurs peaking at +30mV. Source: [10].

The neuron's process of generating an action potential and returning to resting state is simulated by neurons in a neural network. Both follow a cycle of resting state, depolarization, repolarization, hyperpolarization, and return to resting state. Each of these states and the portion of spike at which each occurs is shown in Figure 2.5. Before a neuron fires, the membrane potential at rest is maintained at -70mV. When the sum of the incoming signals reaches the threshold of -55mV, voltage-gated sodium ion channels in the cell membrane are activated to open allowing an influx of sodium ions [9]. The high concentration of sodium ions inside the axon depolarizes the cell creating an action potential that peaks at +30mV [9]. This new voltage activates voltage-gated potassium channels to open, allowing for an outflux of potassium ions, which repolarizes the neuron, bringing the voltage back down [9]. However, while the potassium channels close, the voltage undershoots the initial voltage level, causing hyperpolarization of less than -70mV. It is important to note that during

the repolarization state, there is no amount of stimuli that can induce the neuron to refire [9]; this is called the absolute refractory period. Again, during the hyperpolarization state when the membrane potential is lower than -70mV, the neuron requires higher excitatory stimuli to fire; this is known as the relative refractory period.

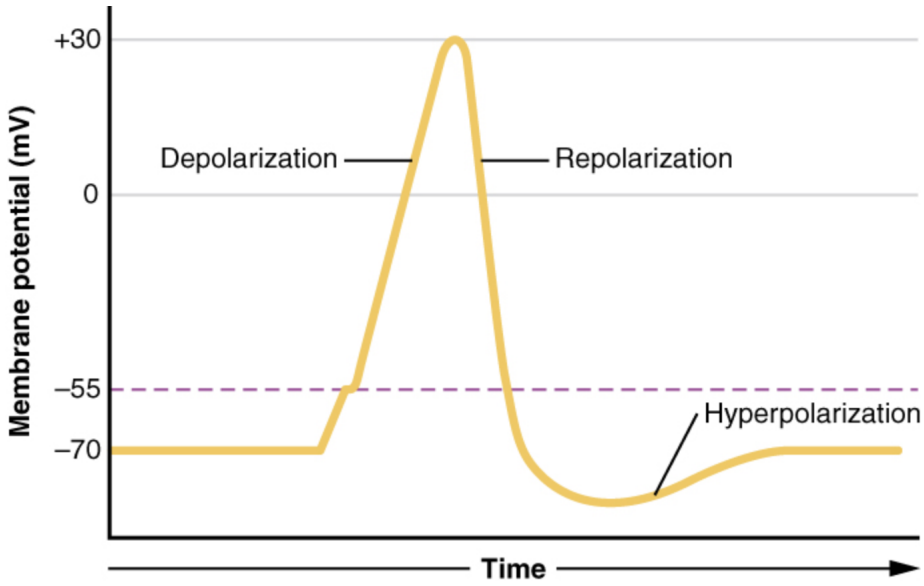


Figure 2.5. The action potential can be considered four states: resting at -70mV, depolarizing from -55mV and 30mV, repolarizing from 30mV to below -70mV, and back to resting. Source: [11].

Similarly, neurons of SNNs have been designed to follow the same depolarization, repolarization, hyperpolarization, and resting state cycle of their organic counterparts. When a SNN neuron is fired, it too has a refractory period where the neuron resets before another spike can fire. Leading researcher in neuromorphic computation Wolgang Maass explains the foundational design of neurons in [8]. Post-synaptic neuron v has an initial resting potential P_v. When pre-synaptic neuron u fires at time s, the membrane potential P_v of neuron v is affected at time t. This contribution can be quantified by the model $w_{u,v}\varepsilon_{u,v}(t-s)$, where $w_{u,v}$ is the weight of the synapse between neuron u and neuron v [8]. The weight and change in time are related by a response function $\varepsilon_{u,v}$. While learning can be expressed as constant function $w_{u,v}(t)$ in a SNN, Maass acknowledges that an accurate reflection of biological

learning would require rapid fluctuation of the value of $w_{u,v}(t)$.

Also modeled after the brain, the STICK framework includes refractory periods after a neuron fires that disables that neuron until its state variables are reset. To represent this disablement during the refractory period, [1] uses gate-synapses that are triggered by the synaptic events, such as the synaptic firing of a neuron, the gate-synapse controls the activation and deactivation of the exponential synapses by use of the gate signal which is represented by a logical 1 to activate the gate-synapse and a 0 to deactivate the synapse. This precise timing is controlled by the use of variables to represent the time a neuron requires to emit the spike after reaching potential and propagation delay between the emitting neuron and the target neurons after the spike occurs. The authors chose $10\mu s$ to represent the time to emit the spike and $1ms$ to represent the propagation delay. After the neuron reaches potential and the predetermined time intervals have occurred, the neuron's state variables, to include the gate, are reset to 0.

2.2.3 Asynchronous Processing

Since the action potential in biological neurons abides by an "All-or-None" Law, the results are binary: same spike each time or no spike at all. The "All-or-None" Law states that a spike occurs only if the threshold is met and the spike always peaks at +30mV. This binary method of spiking conserves energy and resources. Thus, when adapted to neurons in neural networks, the same conservation applies. SNNs, which work asynchronously, are more in line with neurological functions. SNNs maintain the same three components as other neural networks: input layer, hidden layers, and output layer, but operate based on discrete events or spikes in activity. Closely mirroring the behavior of a biological neuron, the neuron in an SNN reaches the threshold determined by a differential equation, the neuron will spike and the neuron's potential will be reset, as shown in Figure 2.5. However, neurons in the network that do not receive spikes that meet their threshold, do not fire, thereby preserving resources.

2.2.4 Sparsely Connected Networks

Artificial neural networks were originally created to mimic the biological activity of the brain. There are three main components of a neural network: the input layer, hidden layers, and the output layer (Figure 2.6). Within an ANN, the neurons are fully connected, taking

continuous values as inputs and outputting continuous values. This does not accurately portray the way the brain's neurons process data as the brain works based on electrical impulses, or spikes. Neurons throughout the brain are not all fully connected as they are in ANNs. They only communicate to neighboring neurons and each neural pathway of linked neurons processes different aspects of the stimulus independently. Since each neuron is only reactive to the neurons nearby, stimuli of different types can initiate signals passed to the brain simultaneously. For instance, when holding an orange, the photoreceptors in the eye send the image and color to the brain at the same time as the mechanoreceptors in the fingertips are getting feedback on the texture of the peel. The input of the texture and weight of the orange is sent to the brain on a separate neural pathway from that of the visual input. Meanwhile, the scent of citrus is picked up by the chemoreceptors in the nose that send the smell over another neural pathway, perhaps a moment later, to the brain to process. These neural pathways working independently more closely reflect the functionality of SNNs.

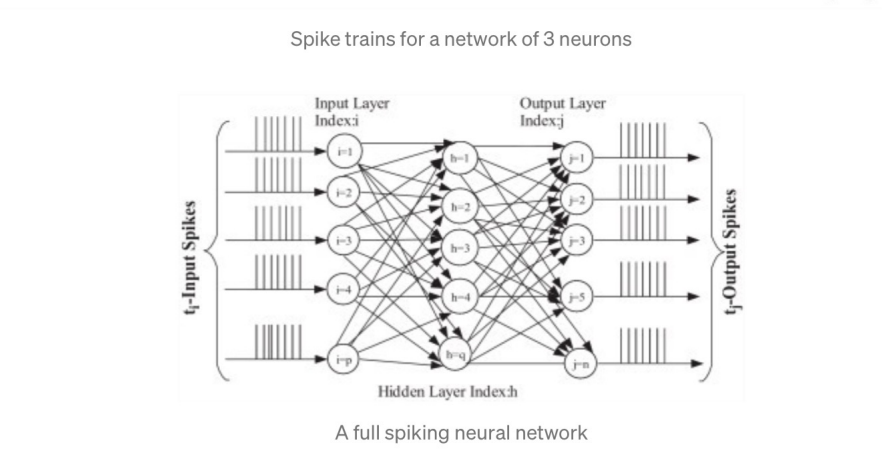


Figure 2.6. General SNN architecture showing fully connected input layer, hidden layers, and output layer, similar to that of a ANN. Source: [12].

Since SNNs are sparsely connected, instead of fully connected, they more closely resemble locally connected neurons in the brain, which allows for an enhanced ability to process spatio-temporal data [12]. Thus, SNNs allow for the natural processing of data without the increased convolution seen with the use of recurrent neural networks where the output

from previous steps is fed as input to the next step and so on. Aside from the biomimetic structure, SNNs are computationally more powerful than other neural network models of similar size [8], which makes them favorable to ANNs.

2.3 Current Solutions and Problems

The majority of commercially available non-radar based tracking systems utilize more traditional style video cameras that represent motion as a series of frame-based camera shots, or stills. These still frames are limited by the frame rate. The current industry standard frame rate is somewhere between 30 - 60 frames per second for high definition capture while the human eye can comprehend as many as 1000 changes per second [5]. When played back, these frames combined create a reproduction of the live events. However, data is missing from all changes that occurred as the camera is blind between frames. This has a significant impact on fast moving objects and is the reason for motion blur. Another disadvantage of frame-based cameras is the majority of the background data in each still frame is not useful for tracking high-speed objects and requires extra processing resources. Each frame from a traditional camera has extraneous and redundant data that must be removed. Not only is this extra processing, the amount of redundant data to store and transmit is substantial. As discussed previously, this is not in line with how the cones and rods interpret data for your brain to process a response to motion [5]. Additionally, these tracking systems and the algorithms ran to detect objects in motion are implemented on commercially available computer systems that are based on von Neumann architecture. The power and time constraints with regard to von Neumann architecture limit the necessary real world usability and increase the bulkiness of these systems.

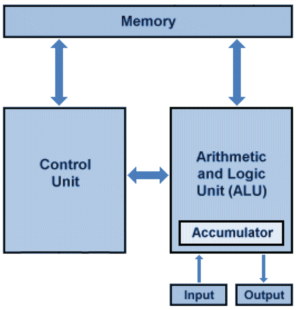


Figure 2.7. The layout of traditional computer architecture retains the expensive I/O between the processor and memory, known as the von Neumann bottleneck. Source: [13].

The von Neumann bottleneck is caused by the design of traditional computer architecture. In von Neumann architecture, memory and processing units are separate entities as shown in Figure 2.7 [13]. Data is stored in memory, but computation occurs in the processing unit. Processing speed is faster than the time required to retrieve data from main memory, thus resulting in excess power consumption and decreased throughput. This bottleneck is especially challenging for complex machine learning algorithms as by their nature they are power and memory intensive. Neural networks require frequent vector matrix computation and process very large amounts of data in order to calculate weights and extract features from the hidden layers. These constraints make it impossible for small devices to run complex machine learning algorithms such as image processing for DVS spatio-temporal data.

2.4 Next Steps

In order for complex machine learning algorithms to progress in terms of practical tracking of projectiles in motion, realistic solutions must be available for moving away from frame-based video capture and circumventing the von Neumann bottleneck. Event-based vision sensors can be used to address the capture of spatio-temporal events. As discussed previously in Section 2.1, the use of event-based sensors to track the leading edge of movements will more closely resemble how natural vision works by recording events as independent pixels. Additionally, event-based sensors are computationally less demanding than frame-based sensors, as they produce up to 1000 times less data and achieve a much higher analogous resolution than frame-based sensors [5].

2.4.1 Importance of Projectile Tracking

Technology across all fields are increasingly relying on automated systems for monitoring, detection, reacting, and, in some cases, resolving various types of problems. Repetitive tasks that do not require human attention can easily be offloaded to machines for higher efficiency and lower costs. The accuracy of a SNN trajectory predictions compared to those of humans in a similar environment can be consistently better. In various lighting conditions, the SNN Debat et. al. constructed had a lower mean absolute error for any given light condition than that of the human prediction results [2]. Furthermore, in lower lighting, invoking lower confidence for both humans and neural network, human predictions vary each time whereas the SNN leans more cautiously toward the statistical average in low lighting and veers more accurately from the average as lighting improves [2]. The potential accuracy and reliability of event-based pipelines in light of their low-resource demands, make them promising candidates for critical systems with strict environmental constraints. As societies become more reliant on computer systems, the vulnerabilities therein are apparent and concerning. From commands on the ground, to aircraft in the atmosphere, to satellites outside the atmosphere, anywhere there are cyber physical systems, there are countless points of failure. Collision avoidance has become a high priority for both DOD and commercial sector. According to the director of command and control at Lockheed Martin's rotary and mission systems division, "the catalogue of what was needed to be tracked was exploding" [14].

Imminently, the amount of debris orbiting the Earth continues to increase, so much that from 1999 to 2020, the International Space Station (ISS) had to make 29 collision avoidance maneuvers, three of which took place in 2020 [15]. Private and public sectors alike are researching better methods of tracking debris of which there are approximately 27,000 pieces 10cm in diameter or larger traveling around 17,500 mph in orbit [15]. In such an environment where the U.S. has critical communication systems in orbit, "Reliable detection and tracking of high-speed projectiles is crucial in providing modern battlefield protection or to be used as a forensic tool" [16].

2.4.2 Current Research

Interest in the field of event-based pipelines has been growing over the past several years. As more research is done in this area, more uses, support, discoveries, and developments

will arise to expand the capabilities of current implementations.

In May 2021, the authors of [2] applied spike-timing-dependent plasticity (STDP) rules to a SNN to process output from an event-based sensor. Their goal was to use this event-based pipeline of bio-inspired components to quickly and accurately predict the end point of a ball thrown in an arc. For their neural network, they used three layers of Leak-Integrate-Fire (LIF) neurons that learned the weights of feedforward connections through a STDP rule [2]. True to an event-based SNN, neurons only fire if the membrane potential reach the threshold. Once a neuron fires, the signal is propagated to the next layer of the network before the neuron returns to its resting potential. Each layer was trained individually and unsupervised, using seventy percent of the data to train and thirty percent of the data to test. All the trials were collected using a Neurosoc camera from Yumain. This camera was deliberately chosen for its dual functionality of capturing frames and events. Thus, this event-based camera is equipped for spatial and temporal filtering prior to spike generation, which results in reduced noise, edge detection, and increased output sparseness [2].

The results of their research were three-fold: the pipeline was able to output reliable projectile predictions, the filter selectivity was sufficient for differentiating moving objects, and the pipeline outperforms human predictions for ball trajectory [2]. However, their work did not address hardware limitations due to the von Neumann processor constraints. Considering the effectiveness of their system, the authors have begun to patent their work. Future use of their system appears to have practical uses as it succeeds at more complex trajectories, such as bouncing or collisions, and more complex filtering, perhaps on a trajectory with more movement in the background.

2.4.3 Current Tracking Technology

There have been a handful of methods and technologies used for addressing the issue of high-speed projectile tracking. In 2019, a space fence system was deployed for operation by the U.S. Air Force. The space fence, a ground-based Long Range Discrimination Radar (LRDR), works with 300 other sensors that all feed data to their iSpace software for processing and analysis [14]. The system claims to work in real-time for early detection and identification of ballistic missiles or decoys [17]. Although it is difficult to analyze without details on exactly how LRDR operates, most radar systems rely on a scanning mechanism that does not

maintain constant monitoring. This scanning essentially creates the same problem faced by frame-based sensors: the background of non-moving objects must be processed with every sweep, and there are moments of blindness due to the scan rate. Additionally, the space fence operates from Earth to detect threats outside of Earth's atmosphere. Locality alone presents challenges for maintaining clear and accurate detection. While presently one of the most advanced systems for tracking projectiles, the robust system undoubtedly requires high throughput, data storage, processing power, and physical space.

Smaller, lighter systems have been used for high-speed projectile tracking as well. Acoustic techniques are plausible for detecting and tracking supersonic projectiles. Acoustic sensors are setup in arrays that enable them to detect shock waves to compute the projectile's trajectory [16]. Yet this poses a challenge for targeting subsonic or silenced projectiles. infrared (IR) has also been explored as a viable option for tracking high-speed projectiles at close range. IR sensors can pick up the trajectory and velocity of the projectile; however, this technique works best when the projectile is hot from traveling at supersonic speeds. For projectiles at subsonic speeds, no aerodynamic heat is produced creating a challenge for IR tracking [16]. Other methods have used IR in concert with industrial cameras. As discussed previously, even with a shutter speed of 750fps, frame-based cameras are not accurate enough for tracking projectiles traveling at subsonic speeds of 100 to 300m/s [16].

Event-based pipelines are a light-weight, power efficient, and fast alternative to the other tracking techniques. Given the development of neuromorphic hardware becoming more refined, the possibility and probability of full event-based systems, from the hardware to the software, is plausible for general use. Prophesee is a company invested in manufacturing event-based products. Using their own event-based vision sensors for capturing data, event-based algorithms for customized software, and neuromorphic chips on which to run the event-based processing, Prophesee presents viable event-based machine vision products for any high-speed tracking and real-time processing. With growing interest and investment in event-based systems, a real-time solution for a resource constrained environment can become general use.

2.5 Research Intentions

The research done for this paper is intended to elaborate on event-based pipelines as an optimal means of tracking high-speed projectiles in real time. Given the building blocks of the STICK framework, the design of non-linear networks combined with the use of DVS to compute the trajectory of a projectile in motion would contribute to the expansion of previous work. While smaller components of the ballistic functions are still necessary for the complete functionality of a STICK implementation to be applied to trajectories, this work provides proof of concept that the combination of these two technologies has the capability to allow for trajectory tracking without the current operational cost, constraints, and larger scale requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Experiment Design

The experiment design for this study includes collecting data from a projectile with a set angle of trajectory and varied force against a plain black background. This setup allows the angle of trajectory to be set to differing angles for comparison between processing of varied trials. With this controlled environment, the SNN implementation is not intended for real-world application, but as a proof of concept that can be developed in future work for practical applications on a large scale with varying environments. Therefore, the projectile's speed, landing distance, maximum height, size and shape are restricted by the appearance on the DVS's output. The projectile's whole trajectory must be captured by the DVS and the projectile must be easily distinguishable in the shot for the duration of the flight path for most accurate processing.

This chapter first introduces the materials and devices necessary to run clean, consistent trials, then walks through the filtering and data collection portion of the experiment. Chapter 4 goes into the programming details of any and all processing applied after the data is collected.

3.1 Materials and Devices

This section is an overview of the tools used to run all of the trials. The event-based sensor chosen to capture the motion of the projectile is the DAVIS346. Supporting software by iniVation supplies various noise filters to apply to the capture before the data is ever recorded. Lastly, ten identical white marbles were used as the projectile and the physical projectile launcher was a miniature catapult.

3.1.1 The DAVIS346

Unlike traditional frame-based cameras which capture light based on a timing clock, the DAVIS346 sensor captures light based on the dynamics of the scene. This advantage results in fine temporal resolution imagery, that has a very dynamic range, while maintaining low latency, making it perfect for application in object or projectile tracking. The frame

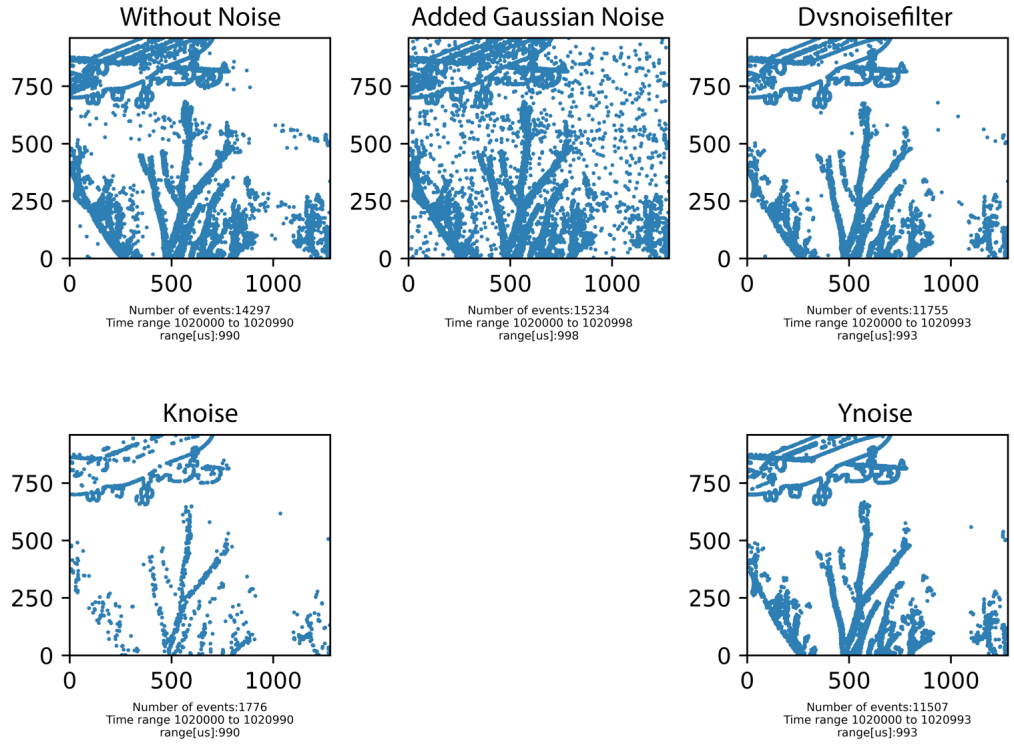
of view (FOV) is dependent upon the focal length of the lens and the width of the pixel array [18]. The pixel array has a width of 346 pixels and height of 260 pixels. The width and height of the array depends on the pitch of each pixel, 18.5μm/pixel for this model [18]. With this pixel array and pixel pitch, the total resolution is 346x260, measuring 6.4mm wide by 4.81mm high [18]. This high pixel resolution results in reduced motion blur as compared to traditional frame-based cameras, which is instrumental in the collection and processing of higher speed projectiles. The lens type is a C mount lens, which mounts directly in front of the chip [18], pictured in Figure 3.1. Since the DAVIS346 can capture events and frames, both versions were saved for comparison during the analysis phase.

Figure 3.1. The DAVIS346 front view shows the chip, which sits in front of where the lens is mounted. Source: [18].

3.1.2 Dynamic Vision Filters

In conjunction with the DVS, we used iniVation's DV software platform to interact with the DAVIS346. The software comes with built-in settings, configurations, and modules that help customize the camera's output. Most importantly, DV has noise filter modules with multiple options for filters including Dvsnoisefilter, Knoisefilter, and Ynoisefilter. While there is overlap in the parameters and ideal contexts for which they are used, each filter has varying strengths and weaknesses. The Dvsnoisefilter is the best option for this use as it is a standard spatio-temporal filter that maps event timestamps and stores the map [19]. In particular, the Dvsnoisefilter has a hot-pixel filter that omits output for "hot pixels", or pixels that are always on. However, as the Dvsnoisefilter is yet one of the more generic filters, the application of the Ynoisefilter combined with the Dvsnoisefilter proved to be the most ideal.

The Ynoisefilter is a more "fine-grain filter"; one of its best uses is image reconstruction [19]. Meanwhile, the Knoisefilter is not ideal for this purpose due to its tendency to incorrectly filter out real data as false negatives [19]. Since it is imperative to maintain as much of the original, true data as possible, we opted against using the Knoisefilter. A visual depiction of the differences is shown in Figure 3.2 from a comparison done by iniVation.



Note: data100k is an artificially generated event file without noise, data_noise100k is that file with added Gaussian noise, the other images are the output of the discussed filters.

Figure 3.2. The Dvsnoisefilter maintains the most noise in the output but does not corrupt real data. The Knoisefilter eliminates noise while also corrupting the image by removing real events, as depicted in the frames with portions of the bush and plane missing. The Ynoisefilter removes much of the noise while preserving the image. Source: [19].

The DV software GUI provides a visual workflow into which custom modules, such as noise filters, can be placed. As seen in Figure 3.3, the GUI allows users to add or remove

modules from the workflow while visually keeping track of the order in which the modules are applied. For the purposes of this evolution, the Dvsnoise filter and Ynoisefilter were used in combination with the black background screen, to allow for the removal of the majority of background noise and any areas where light hit the background but did not represent motion.

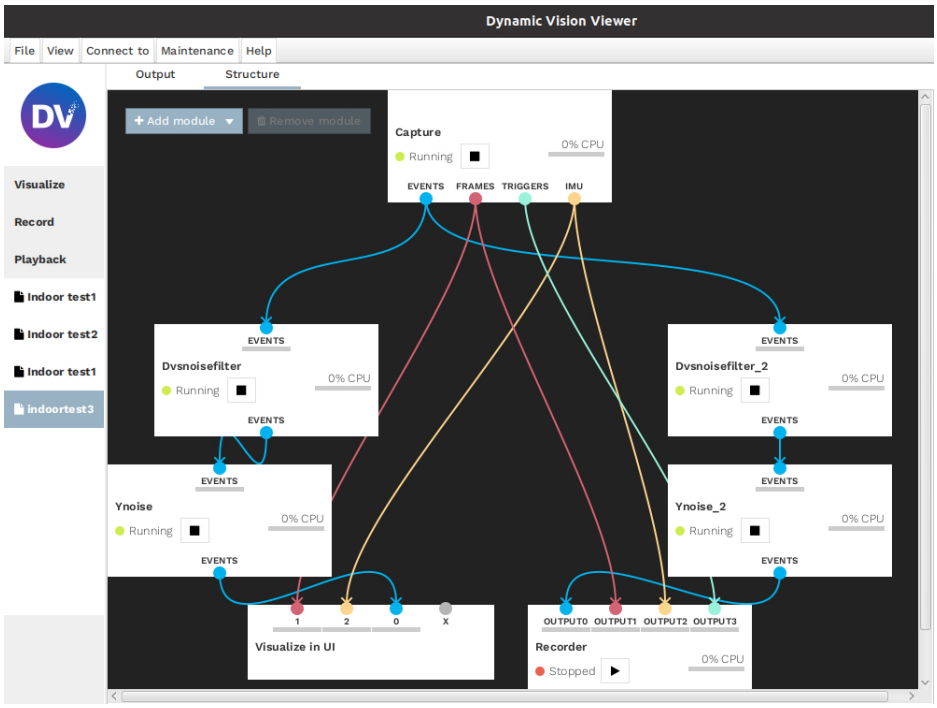


Figure 3.3. The DV GUI representation of a workflow stacking the Dvsnoise-filter and Ynoisefilter modules in a single configuration.

3.1.3 Projectile, Launching Mechanism, and Staging

The projectile had to be large enough for the model to easily detect, but as small as possible to allow for a large portion of the arc to be caught by the DVS field of view and be easily displayed on the output screen. The projectile was a marble, chosen for its uniform shape and ideal weight and size for our particular projectile launcher to toss. The white marbles, measuring 1.95cm in diameter and weighing 3.97 grams, were tossed at 2.4 meters from the DVS and 1.5 meters in front of a black background. A black photography backdrop

measuring ten by twelve feet was suspended evenly with the top edge eight feet from the floor, leaving excess fabric draped on the floor. The backdrop was made out of two millimeter thick polyester fabric and was utilized to prevent additional light reflection and reduce gloss. The black backdrop also provided the most color distinction from the white marble and absorbed excess light, preventing areas of color saturation from being picked up by the DVS. Additionally, the use of a directional studio light helped to minimized as much of the physical noise as possible and brighten the leading edge of the marble in flight.

For the projectile launcher, a small crafts catapult assembled with thin wood and rubber bands (Figure 3.4) ensured consistency between each launch. It throws identical objects with the same trajectory to produce consistent trials and repeatable arcs for future experimentation. The catapult was set on top of a flat platform such that when the marble is released, the release point was 56.388cm from the floor. Given the weight and size of the marble, the catapult launched each marble an average horizontal distance of 3.15m from the starting point.

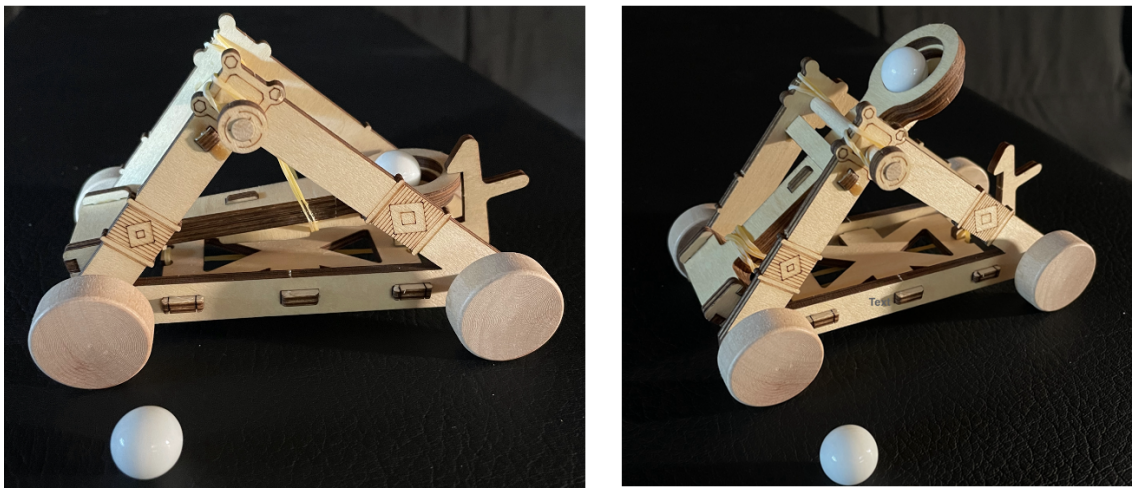


Figure 3.4. Catapult assembled from thin wood and rubber bands for maximum consistency between trials. Marbles included for size reference.

3.2 Cleaning Methods

In order to obtain insight on predicting the position of a high-speed projectile using a SNN, pre-processing of the data requires applying filters through the DV interface to filter the events before it is piped to the screen output and saved to file. With the filters selected, the trials are ran in a methodical manner to ensure clean, consistent shots. This section covers the data collection process as well as the method used for confirming that the quality of each trial is sufficient. The final steps of cleaning and testing the data with our ballistics models are covered in depth in Chapter 4.

3.2.1 Data Collection

In an effort to conduct a repeatable experiment for any future research, we controlled as many aspects of the trials as we could. The aforementioned catapult was used for consistent force and trajectory, and plain white marbles—all of the same size and weight—for consistent tosses. Thirty trials were recorded with only the Dvsnoisefilter applied (Figure 3.5)and fifty trials with both the Dvsnoisefilter and Ynoisefilter stacked consecutively (Figure 3.6). Of the fifty trials with both filters, thirty of them were done with the catapult at no incline and twenty trials with the catapult at a 2 centimeter incline. The thirty trials without incline are intended to show a baseline for the output while the twenty trials with an incline show the performance of the neural network when variation is introduced in the trajectory angle.

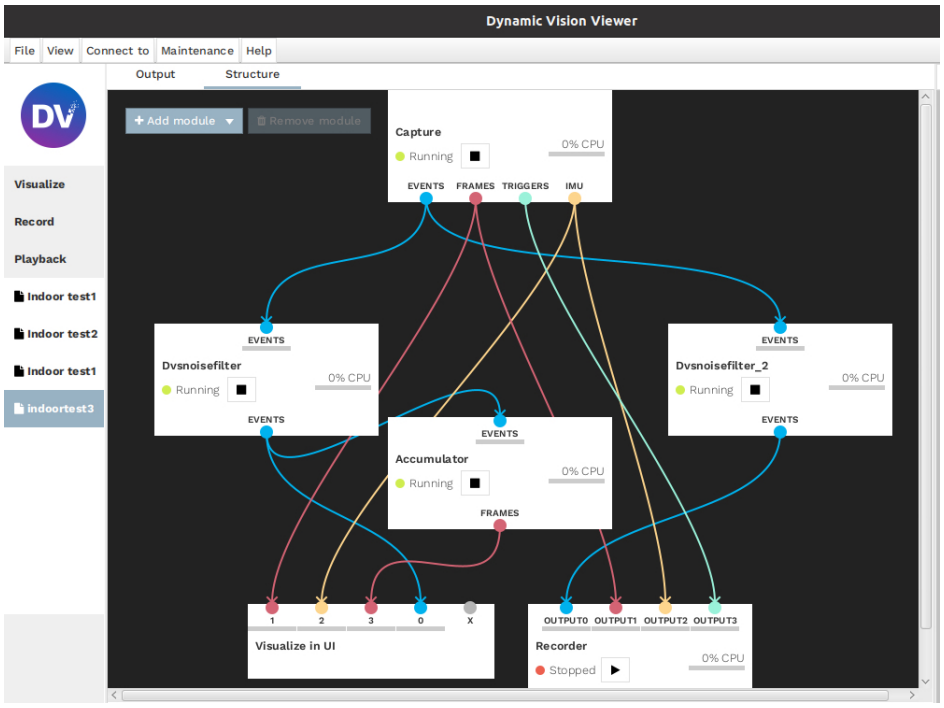


Figure 3.5. The output of the capture is passed to the Dvsnoisefilter, which sends its output through the accumulator for live feedback, before sending that output to the visualizer to be seen on screen during captures and then recorded to be saved as a file.

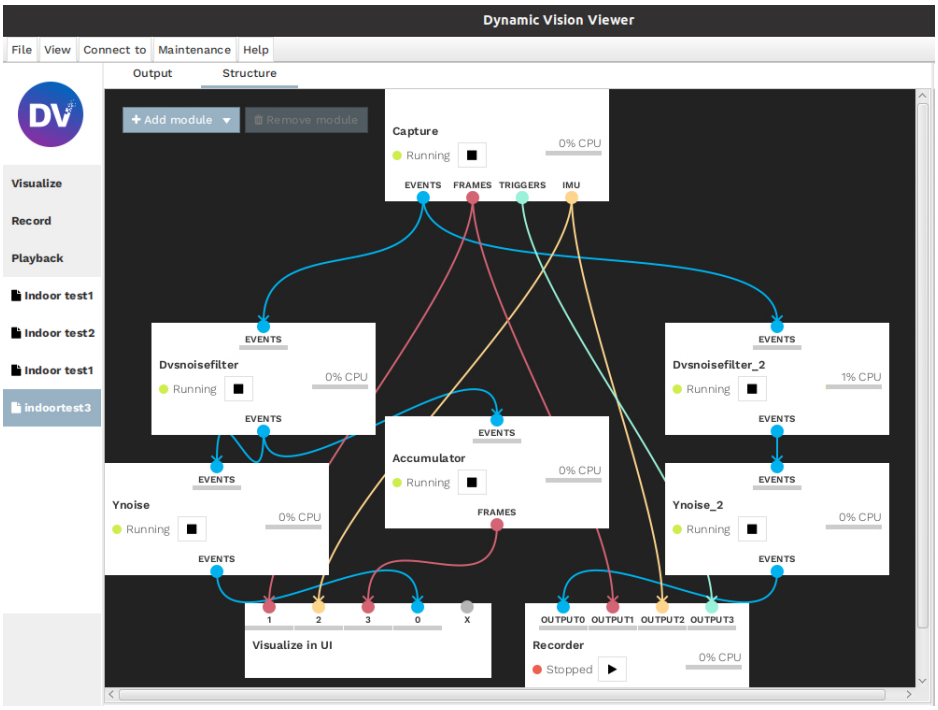


Figure 3.6. The output of the capture is passed to the Dvsnoisefilter, which sends its output through both the accumulator for live feedback and the Ynoisefilter. All outputs are sent to the visualizer to be seen on screen during captures and the recorder to be saved as a file.

3.2.2 Quality Confirmation

Plotting the Aedat4 files in Python showed a rough evaluation of the quality of the output from the trials. Based on some of the first takes, the Dvsnoisefilter and Ynoisefilter removed much of the noise from our scatter plot. To increase the number of events the DVS picked up, setting up a studio light on the right side of the set so the marble was moving in the direction of the light enabled more pixels to be lit up as it brightened the leading edge of the marble. Based on the visual output of the DVS the projectile's trajectory is covered better. This is verified against the output of the scatter plot. After several iterations of using different data collection techniques, this method worked best for collecting the most events for each trial.

It is worth mentioning that the original set of trials used a black marble against a white background with top lighting from the ceiling. This setup was specifically disadvantageous because of the light reflecting off of the white background and the shadows cast by the lights from above. Reversing the setup to a black-on-white scheme, limiting the ambient lighting, and controlling the direction from which the main light source came drastically improved the results of the scatter plot.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Methodology

Prior to processing the data, it first needs to be cleaned and checked for validity and usability within the traditional ballistics model. There are several methods available that can be used to predict trajectories, the data needs to be verified against these different methods of implementation to determine if the data meets the usability threshold and can accurately predict the projectile's path. This allows for the best data and model combination to be used later in the SNN. This chapter will discuss the programming environment and techniques used for preparing and processing the data collected in Chapter 3.

4.1 Programming Environment

For the programming portion Python provides the versatility, functionality, and supporting packages and documentation that is necessary for processing DVS output with Aedat files, prediction models, and substantial amounts of data. The most up-to-date version, 3.9.7, supports all necessary packages for the required data processing and analysis components to enable visualization of the events from our iniVitation DVS camera, more specifically the output format Address Event Data 4 (Aedat4). Visual Studio Code was chosen as the code editor due to it being open-source, its vast library of extensions, syntax support, and user friendliness.

Required packages include:

- pandas - to organize and manipulate the data in dataframes
- numpy - to handle and manipulate data in arrays
- matplotlib.pyplot - to plot the data graphically
- math - for arithmetic calculations for ballistics equations
- curve_fit from scipy.optimize - to model the ballistics equations and find the best fit curve for a selected number of data points
- AedatFile from dv - to load data from the Aedat4 file to Numpy arrays

4.2 Importing Data from AEDAT4

The DAVIS346 exports the collected input as an Aedat file, specifically version 4, which uses the extension ".aedat4". AEDAT4 makes use of Google Flatbuffers in order to convert the data objects that represent visualized movement of the projectile while in motion into a more accessible format, a series of bytes that are organized in an easy to import, process, and manipulate. It also creates the necessary support files for Python readability and interpretation (iniVation software user guide). The data is grouped by three fields, compression which consists of an algorithm to compress all data in the file, dataTablePosition which provides the offset information for each piece of data, enabling quick reference and non-sequential memory access, and the infoNode which is a descriptive informational field, allowing for the tagging of each stream.

DVS data was processed utilizing the AedatFile package imported from the DV System for Python library which allows for the processing of AEDAT4 data into Python in the raw or processed forms. This was utilized to port the data into Numpy arrays and Pandas dataframes. The data was broken into columns of timestamps, x-values, y-values, and polarities. Timestamps represent the time of capture in microseconds (μs), increasing from the initial start of recording and progressing until the end of recording occurs. The x-values represent the horizontal position of the projectile during its flight; the y-values represent the vertical position of the projectile during its flight. Polarity represents the presence of light detected by the DVS camera. A polarity of zero correlates to detection of movement, while polarity of one correlates to the absence of light.

4.2.1 AEDAT4 File to Dataframes

The trials are recorded as one continuous file, without an organic way to distinguish where one trial ends and the next trial begins. Additionally, each trial contains all pixels regardless of the presence of movement from the leading edge of the projectile. Therefore, the data is initially scrubbed of all event frames consisting of a polarity of one, which indicates there is not a movement detected. Then, the indices are reset to reflect only the leading edge pixels. Trials are labeled by number by separating the event data where the x-values are reset to a lower number. A new column called trial_number was added to the dataframe in order to capture this change. The file containing all trials was then split apart by masking the trials by number and storing them each as a dataframe in a list of separate dataframes.

As shown in figure 4.1, the initial scatter plot of the data has the y-values mirrored about the approximated line $y = 137$, the inverse of the projectile's true motion. Additionally, the initial data set has undesirable noise, which impacts the success of the predictive model.

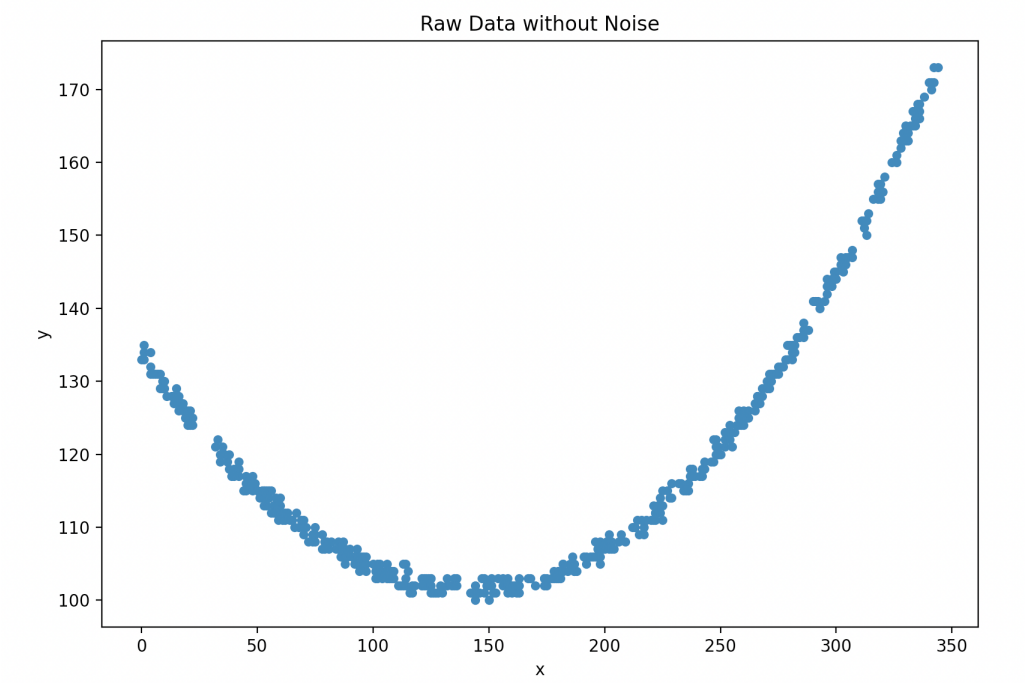


Figure 4.1. Plotted raw data points before y-values are adjusted and rolling median is taken. However, all noise has been filtered using the DV software discussed in 3.2.1.

4.2.2 Axis Inversion and Timestamp Adjustment

Data cleaning must be conducted to ensure the integrity of the results. The origin of the axes for DVS output is consistent with other video and imaging software which begin at the top left corner as opposed to the bottom left where mathematical axes begin. This adjustment only affects the y-axis, as the x-axis data remains on the left. We made a function to translate the data to begin at the mathematical origin. This is done by subtracting each individual y-value from the last y-value in the trial. The resulting curve is displayed in Figure 4.2.

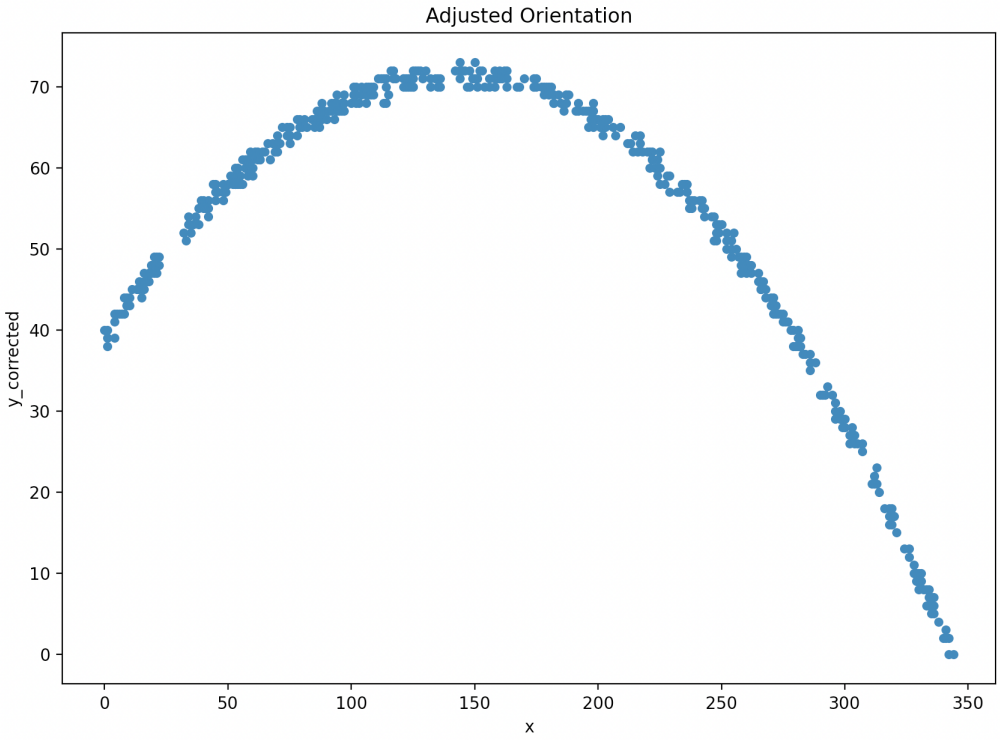


Figure 4.2. Data plotted after the function to translate the data has been applied. The translated data more accurately starts with the origin at the bottom left.

In the same function, the timestamps are adjusted for ease of computing. The original timestamps were sixteen digits long, which quickly outgrows Python's limit of working with twenty-digit long integers. On a 64-bit processor, the largest integer value Python 3.8 can handle is $2 \exp 63 - 1 = 9223372036854775807$ [20]. In order to maintain precision, for every trial r, this function subtracts the first timestamp of trial r from each timestamp in trial r. By subtracting off the value of the first timestamp, the differences between each timestamp remain the same but the numbers used in calculation are only six digits in length. The products of values in the thousands are short enough that Python does not truncate them, and in doing so preserves precision.

4.3 Applying a Moving Median

As discussed previously, the noise filters are responsible for the majority of the data preparation. In order to remove any remaining noise from the data, a moving, rolling median is applied to the dataframes. The moving median is robust against anomalies and works to smooth out the curve. A moving median was selected over the use of a moving average in order to ensure the presence of anomalies would not impact the trend calculations. The moving median is considered a robust statistic and is not affected by inconsistencies such as outliers and data anomalies [21]. Figure 4.3 shows the contrast between applying a moving average vice a rolling median for a raw data set with an obvious outlier. When running the trials for our data set, the leading edge of the marble lit up multiple pixels throughout the arc, causing the progression of pixels to appear to move backwards on the x-axis in some instances. This backward progress causes an issue for the models' predictive abilities. To avoid this issue, rows where this anomaly occurred were purged from the dataframe. The resultant data is clean and appropriate to be utilized as input for the models to produce valid outputs.

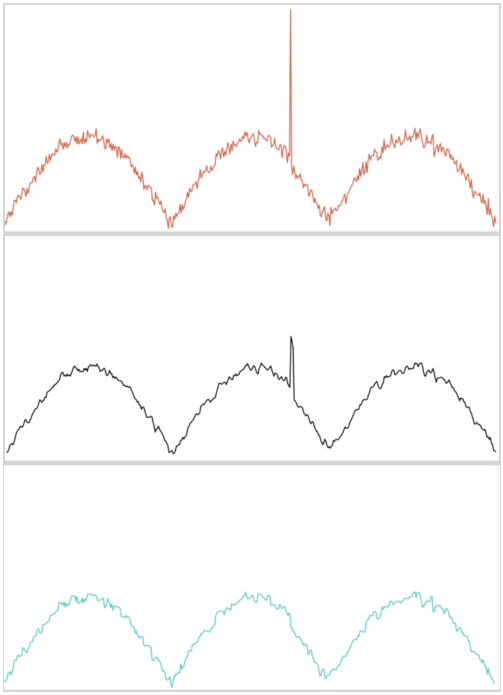


Figure 4.3. Top: Raw data set with apparent outlier. Middle: Moving average applied to the raw data smooths the outlier but is not as robust as the moving median. Bottom: Moving median applied to the data set removes the outlier completely, making the moving median a robust statistic. Source: [21].

Pandas.series.rolling function was used independently on the x- and y-values in order to achieve this objective. The window size, or number of distinct observations, utilized to calculate the resultant figure is varied based on the number of data points per trial in an effort to achieve the best outcome. The "center" parameter is set to true for the rolling function in order to center align the values. This prevents the resulting column of values from developing a right skew. Additionally, the parameter for min_periods was set to the value of 1, in order to preserve the array size and keep the dataframes aligned, without the addition of NAN values to the end. The resulting curve is displayed in Figure 4.4.

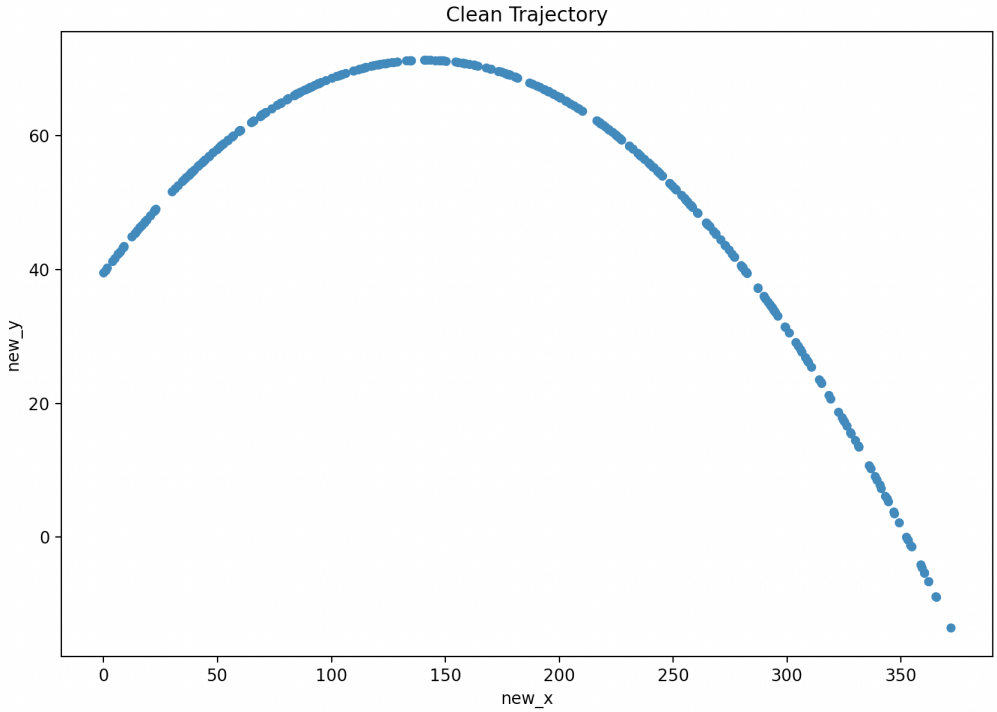


Figure 4.4. Clean data, ready to be used as input to the models.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Implementation

After the data collection and pre-processing is complete, the next step is to verify that the ballistics equations are accurately fit to the data, without overfitting, before creating the implementation with the STICK framework. The horizontal and vertical components of the trajectory are discussed separately as they are modeled by two different equations and are implemented independently. This chapter walks through how the ballistic models are set up and how to estimate the parameters used to determine initial height, initial velocity, and initial launch angle through the use of a curve fitting tool. This chapter further discusses the implementation of STICK framework to create the ballistics equations as a neuromorphic computing solution and defines the specifics of each component required for the final implementation. Finally, it covers how these pieces work together to produce estimated projectile trajectories through the event-based network.

5.1 Ballistic Equations

When looking at projectile motion, Newtonian physics separates the forces working on the projectile into two vectors: vertical and horizontal. The forces at play in the vertical direction are gravity pulling down towards the Earth, air resistance opposing the projectile's vertical motion, and the catapult's thrust upward. The horizontal forces at play are the catapult's thrust horizontally on the projectile and air resistance opposing the projectile's horizontal motion. In a vacuum, air resistance is non-existent; however, in this set up, air resistance is present, but considered negligible. Therefore, in Section 5.1.1 the omission of air resistance from all ballistic equations and models is further discussed.

Horizontal Component of Ballistics Model

Considering the separate horizontal and vertical components of a projectile launched in an arc, the ballistics equations of each component is used as a separate model for the predictions. The horizontal component requires a linear equation (Equation 5.1) taking two

parameters: time t and initial velocity v_{0x}.

$$x = v_{0x}t, \tag{5.1}$$

where x is the predicted x-coordinate, v_{0x} is the initial velocity in the x direction, and t is the timestamp at which the x-coordinate is predicted. This linear function is expected to output a straight line with a positive slope, which expresses the horizontal movement of the projectile independent of the vertical component. At every consecutive timestamp, the projectile is expected to move farther from its initial position, thus, farther from the x-axis, while also moving away from the y-axis as time moves away from the starting time. The horizontal component of the projectile motion is shown in Figure 5.1.

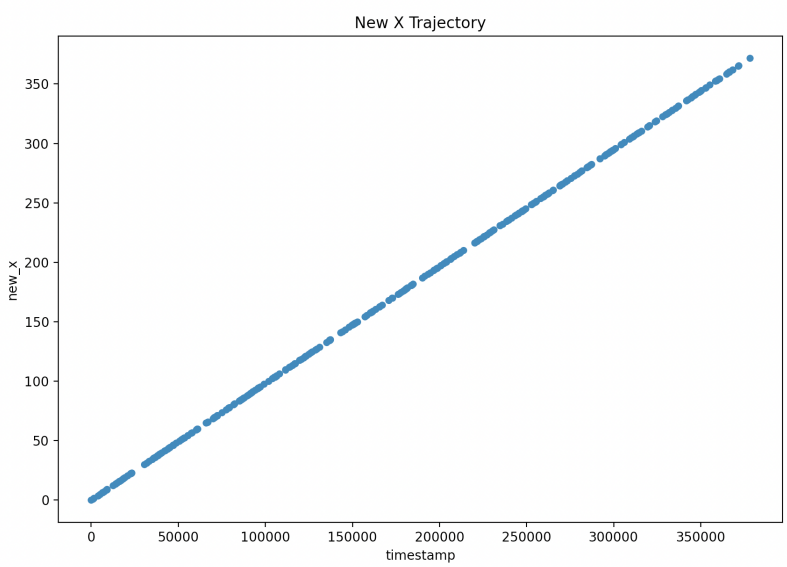


Figure 5.1. Plotting the x-values versus time shows the horizontal component graphed without the vertical component results in a line starting at the origin with a positive slope.

Vertical Component of Ballistics Model

Independent of the horizontal component, the vertical component is derived from the classic parabolic equation in the form of $y = ax^2 + bx + c$. Equation 5.2 takes three parameters:

time t, initial vertical velocity v_{0_y}, and initial height h, as seen in the following equation:

$$y = h + v_{0_y}t - \frac{1}{2}gt^2, \tag{5.2}$$

where h is the initial height of the projectile's motion, v_0 is the initial vertical velocity, t is the time at which the position occurs, and g is the gravitational acceleration.

When graphed, the expected path of the vertical motion is a parabola with respect to time where the initial angle and initial height are consistent with those from Equation 5.2, as seen in Figure 5.2. The y-values move farther from the x-axis as the position of the projectile increases to an apex before decreasing toward the x-axis as time continues to move away from the y-axis as time passes.

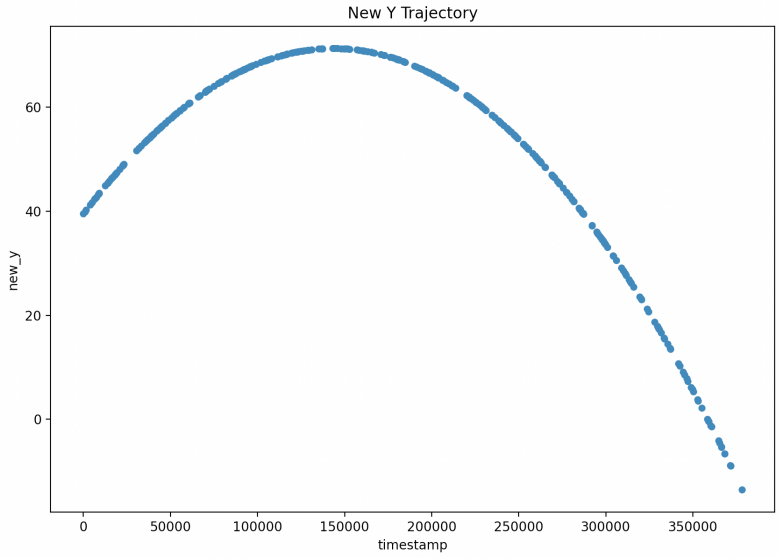


Figure 5.2. Plotting the y-values versus time shows the vertical component graphed without the horizontal component.

5.1.1 Air Resistance

Since air resistance changes depending upon the surface texture of the sphere, the air density—dependent on air temperature, and the instantaneous velocity of the sphere [22],

the air resistance coefficient is estimated as a constant for this model. In practicality, air resistance is dynamic and changes with the environment and velocity of the projectile. These ballistic equations do not directly address air resistance because, when curve fitting the model, air resistance is taken into consideration in the coefficient estimations.

5.2 Curve Fit

For each model, several variables are unknown and must be estimated. The initial launch angle, while known at time of projectile launch by the experimenters, is unknown to the program. Thus, the initial launch angle is one of the variables that must be estimated. Additionally, the initial velocity of the projectile is unknown. The initial velocity must be predicted as the projectile equation depends on the constant velocity of the horizontal component and the updating velocity of the vertical component. This requires a method for determining estimated values of the variables for the model. Curve fitting is a known reliable way to determine best fit for projectile motion, as it works to fit the data points to a curve. The resultant output from curve fitting should provide the most optimal parameters for the ballistics equations used in the model. The Scientific Python (SciPy) library contains a built in function, which was used here, to fit a curve to the projectile's data points.

SciPy.optimize.curve_fit works by using a mapping function, or basis function, before searching for the most accurate weights to produce the minimum error possible. It does this by finding the lowest sum of the squared residuals, a method called non-linear least squares which confirms that the error with curve_fit is as minimal as possible. The key to curve_fit is the form of the mapping function. As discussed previously in Section 5.1, the horizontal component and vertical component variables are independent. Therefore, the mapping functions for the projectile's vertical and horizontal components of motion will be calculated with two separate mapping functions. In order to preserve a measure of training and test data, twenty percent of the data is used with the curve_fit function to train, with eighty percent of the data used to test for each individual trial.

5.2.1 Curve Fitting the Horizontal Component

For the horizontal component of the projectile's motion, the mapping function used with curve_fit is shown in Equation 5.1, which is the standard linear equation for predicting

locations given time and the initial velocity of x. The velocity is multiplied by the amount of time that has passed since the initial leading edge of the projectile was detected. The training portion of data is fed to the curve_fit function in order to best predict the values for estimated parameters. In this case, the intent is to create a best fit estimate of the initial velocity. curve_fit returns two pieces of data: $popt$ and $pcov$. $popt$ is an array containing the optimal values of the parameters, in the case for the horizontal component of motion this is just one variable: the initial velocity of x. $pcov$ is the estimated covariance matrix of $popt$ parameters; the covariance for x is 3.0e-12, this is trivial factor for the horizontal component. The initial velocity value returned by the curve_fit function is then used along with the testing data in the linear ballistics equation for predicting the horizontal trajectory. The outcome of running this model is an array of predicted x-values based on the real-time values, plotted in Figure 5.1. The resultant plot is a smooth, consistent line, as expected when predicting the horizontal trajectory.

5.2.2 Curve Fitting the Vertical Component

For the vertical component of the projectile's motion, the mapping function used with curve_fit must be non-linear since the vertical motion of the projectile is non-linear. Like the horizontal component, the vertical component is a function of the amount of time that has passed since the leading edge of the projectile was detected. However, gravity and the initial angle of trajectory are also incorporated in the parabolic function for the most accurate flight path. Since the vertical velocity is constantly changing throughout the arc of motion, both the launch angle and initial y velocity are parameters that must be calculated. Since the projectile was launched from an elevated position, the initial height must be estimated as well. The ballistics equation adds the height to the parabolic ballistics equation, as seen in Equation 5.2. The curve_fit function again returns two types of data for non-linear functions: $popt$ and $pcov$. This time, $popt$ returns the optimal values for initial height and the initial y-velocity. $pcov$ returns the covariance matrix for these parameters, as shown in Table 5.1. Since the initial height and initial velocity are independent variables with no effect on each other, the results of covariance support this relationship, as $pcov$ is consistently a trivial value. While the covariances for each trial vary from values between -9.8e-07 to 5.0e-02, they all are low enough to show the trivial relationship between the $popt$ values.

The estimated weights, determined by training the ballistics equation model, are fed back

Covariance	Initial height	Initial velocity
Initial height	1.02102872e-02	-1.65505610e-07
Initial velocity	-1.20928535e-07	1.82734806e-12

Table 5.1. The $pcov$ matrix of a single trial shows the $pcov$ values for the estimated parameters.The pcov matrix does not calculate covariance of a variable against itself.

into the ballistics equation function along with the timestamps of the testing data. This results in the creation of an array of y-values corresponding to the timestamps where real y-values were recorded. The data for both the y-component and x-component of the projectile motion are now ready to be analyzed to determine their level of accuracy. These weights will be found for each individual trial separately, as each trial has a unique launch angle and velocity.

5.3 STICK Framework Design

The STICK framework was developed as a solution to the von Neumann bottleneck seen in traditional computer hardware. This solution does not separate memory from computation, rather it inherently stores the data and uses the data for computation simultaneously [1]. Lagorce and Benosman use the time intervals between spikes in a SNN to encode the values used as input for the neural network. In this way, the data is processed in memory, without experiencing the latency created by conventional architecture. Their set up is such that, every input has two spikes and two input neurons: $first$ and $last$. The duration between these spikes can be encoded using various methods, including logarithmic, linear, and luminance time encoding; however, they use the following linear encoding for their research to account for the refractory period:

$$\Delta t = f(x) = T_{min} + x.T_{cod} \tag{5.3}$$

where $x \epsilon [0, 1]$, $T_{min} = 10ms$ and $T_{cod} = 100ms$ is the smallest time step.

Lagorce and Benosman's work shows that the use of neuromorphic networks allows for the creation of a Turing complete framework that can calculate all known mathematical functions using non-von Neumann architecture. [1]. Therefore, STICK allows for the derivation of all mathematical operators, linear and non-linear [1]. With this foundation, the networks developed in the STICK research paper were modified and networks joined or combined to fit the ballistic models for the flight path of a projectile.

STICK Neuron Model

The neuron model shown as Figure 5.3 is utilized as the foundation for the creation of the computational units. It is assumed to be non-leaking, meaning there is no leakage of the membrane voltage, V, or leakage is of a negligible amount. g_e is the constant input current that is only changed through synaptic events. g_f is the exponential input synaptic events, these are controlled by the synaptic gate signal. τ_m is the time constant of the membrane, set to 100 milliseconds by the authors. τ_f is the exponential time constant, set to 10 milliseconds by the authors [1].

$$\begin{cases} \tau_m . \frac{\mathrm{d}V}{dt} &= g_e + gate.g_f \\ \frac{\mathrm{d}g_e}{dt} &= 0 \\ \tau_f . \frac{\mathrm{d}g_f}{dt} &= -g_f \end{cases}$$

Figure 5.3. The neuron model used by Lagorce and Benosman where V is the membrane potential. Source: [1].

STICK Synaptic Model

There are four synapses as shown below in Figure 5.4. These synapses represent the connection between neurons in the network and handle the propagation of signals. These synaptic inputs directly change the respective neurons. w is the weight of each synapse. Each computational unit synapse simultaneously stores and uses the transmitted data [1].

- $V-$ synapses directly modify the membrane potential value: $V \leftarrow V + w$,
- g_e- synapses directly modify the constant input current: $g_e \leftarrow g_e + w$,
- g_f- synapses directly modify exponential input current: $g_f \leftarrow g_f + w$,
- $gate-$ synapses : $w = 1$ activate the exponential synapses by setting $gate \leftarrow 1$; $w = -1$ deactivate the exponential synapses by setting $gate \leftarrow 0$.

Figure 5.4. The four types of synapses used with weight w for each. Source: [1].

STICK Memory Storage

In order to store and recall values without modification, a memory capability was created as shown in Figure 5.5. The network is composed of two input neurons, three internal neurons, and an output neuron. The $first$ and $last$ neurons are used to sort spikes as each pair of spikes reach the input neuron. Synaptic connections from $first$ to $last$ cause $first$ to post-synaptically spike only upon receiving the $first$ spike (at time t_{in}^1) and $last$ to post-synaptically spike only upon receiving the second spike (at time t_{in}^2) [1]. The membrane potential of acc after both input spikes is the difference in t_{in}^2 and t_{in}^1. The $recall$ neuron is the other input neuron and reads out the value stored in the membrane potential of the acc neuron when it is triggered. When the $recall$ neuron fires, the integration process begins again. As a result, an output spike equal to the represented value minus the encoding time is produced, thereby enabling memory recall. Two inverted memory networks can be linked to store and recall the non-modified value [1].

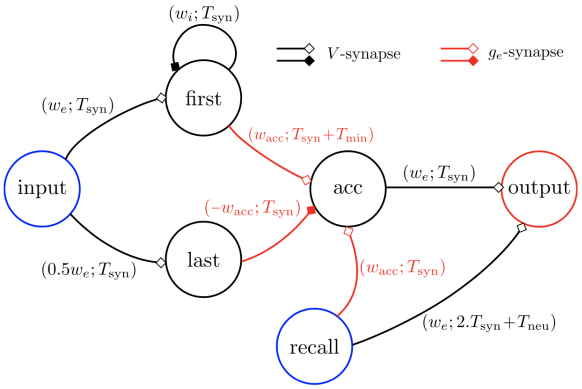


Figure 5.5. The inverted memory network stores a value until it is recalled and outputs the inversion of the input value. Input neurons are shown in blue, internal neurons are shown in black, and the output neuron is shown in red. Source: [1].

5.3.1 Computational Units Needed

To apply the STICK framework to the ballistic models described in Sections 5.1 and 5.2, four main computational units are necessary to carry out computations for both the horizontal trajectory prediction and vertical trajectory prediction. Each of the components is described in detail to fully explain its structure and functionality.

Synchronizer Network

Figure 5.6 displays the STICK synchronizer network. The network receives a number of inputs, represented as N and stores their values through the use of individual STICK memory storage devices discussed in Section 5.3. Once the synchronizer receives the last pair of input spikes, the synchronizer neuron will fire, causing the stored values to be read out simultaneously [1].

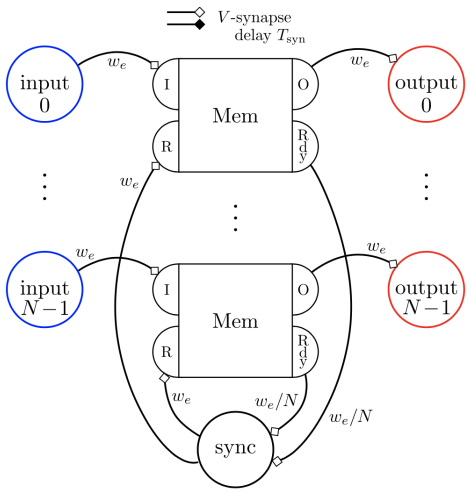


Figure 5.6. Two memory networks store values until the *sync* neuron recalls. The *sync* neuron only recalls these values once the last value is stored. In this way, all output spikes occur simultaneously. Source: [1].

Accumulator Neuron

The accumulator neuron is where the difference in time ΔT_{in}, between the first spike of $input1$ and the second spike of $input1$, is stored. This ΔT_{in} is the value necessary for further processing or future operations. Values are stored in the accumulator's membrane potential by the same method as described in the memory network discussed in 5.3. When the recall neuron is triggered, the value stored in the accumulator neuron is read out [1].

Multiplier Network

The multiplier network calculates the product of two inputs by taking the exponent of the sum of the natural logs of each input, as follows: $exp(log(x_1) + log(x_2))$. Each input pair is comprised of a $first$ and $last$ spike. It uses two computational neurons, acc_log1 and acc_log2 to calculate the natural logarithm of the input using the g_f dynamics of neuron acc. The input value is first stored in the membrane potential of neuron acc. Once the second encoding spike appears, the delay function of the log is calculated [1]. The completion of the logarithmic computation done in acc_log1 activates acc_log2, which sends the results

to the exponential circuit. The resulting calculation of the final product is sent to the *output* neuron, as shown in Lagorce and Benosman's multiplier network (Figure 5.7):

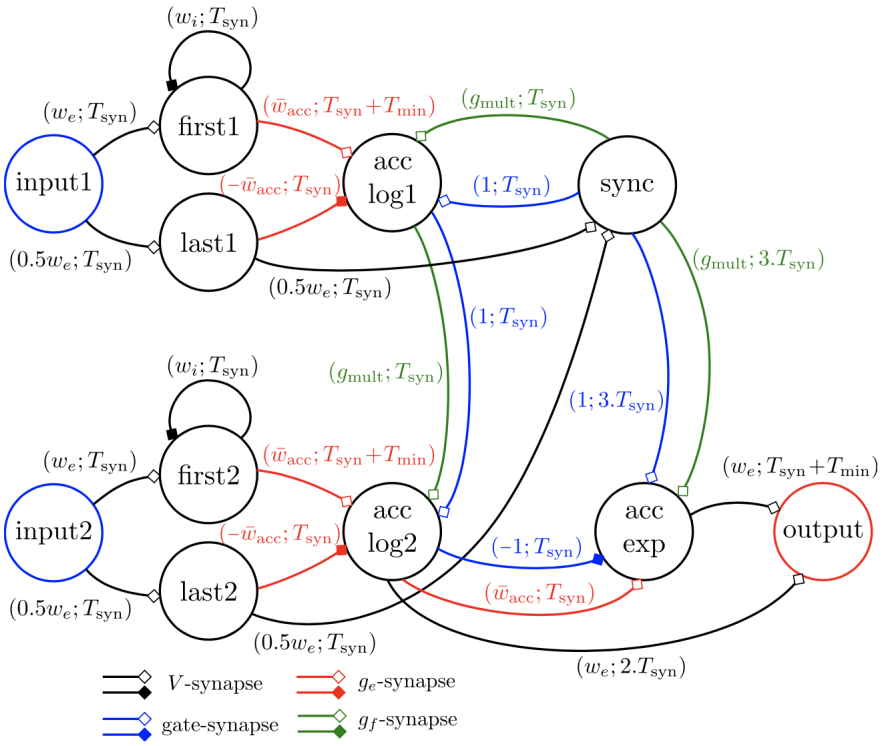


Figure 5.7. The multiplier network calculates the product of two inputs by summing their natural logs and taking the exponential of the sum. Source: [1].

Subtractor Network

The subtractor network subtracts its two inputs. Due to the synchronization of the inputs, the difference is found by subtracting the time between action potentials of the second spike of each input [1]. A positive difference is output to the *output*+ neuron and a negative difference is output to the *output*− neuron, as shown in Figure 5.8. Activation of both input neurons triggers *sync*1 and *sync*2 neurons to set their membrane potentials to $\frac{1}{2}V_t$. The *sync* neurons signal the inhibitor neurons, such that *sync*1 excites *inb*1 and inhibits *inb*2 while *sync*2 excites *inb*2 and inhibits *inb*1. Each inhibitor neuron inhibits its output neuron

allowing the opposite output neuron to spike first. The job of the $inb1$ neuron is to inhibit the $output+$ neuron, while the job of $inb2$ is to inhibit the $output-$ neuron. The network takes advantage of the sequential nature of time by ensuring that if the first input is smaller than the second, then the $output-$ neuron is activated first, whereas, if the second input is smaller than the first, then the $output+$ neuron is activated first. This set up ensures that the appropriate parity is assigned to the output.

The $zero$ neuron detects the case where the subtraction results in zero and outputs a zero on the $output+$ neuron. The authors chose to count zero as a positive output to alleviate the system spiking both positive and negative neurons on a zero [1]. After the first spike of each input is synchronized, the last spike at the *sync* neuron excites its inhibitor, inhibits the opposite inhibitor, then excites its output neuron, and inhibits the opposite output. At the same time, it signals the $zero$ neuron. If the spikes from both $sync1$ and $sync2$ arrive at the $zero$ neuron at the same time, the $zero$ neuron inhibits $inb2$ and inhibits the negative output neuron with double the weight to ensure that the signal will only propagate to the positive output neuron.

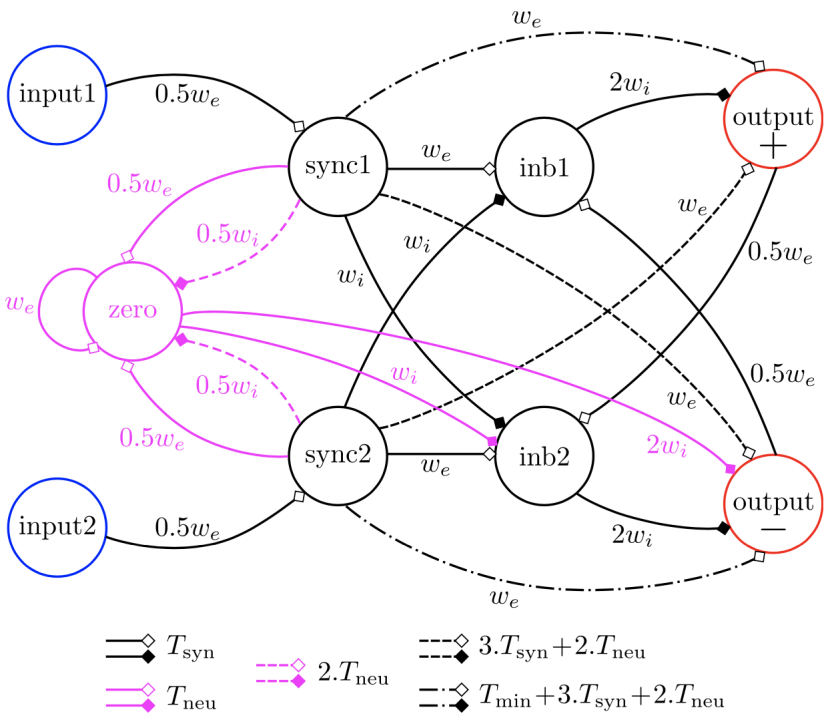


Figure 5.8. The difference between two inputs yields a single output in either the positive or negative output neurons. The *zero* neuron, shown in fuchsia, prevents both output neurons from spiking. Source: [1].

5.3.2 Putting It All Together

As stated previously in 5.1, the linear model yields the x-coordinates by taking the product of the instantaneous horizontal velocity v_{0x} and the time t. The non-linear polynomial regression yields the y-coordinate by taking the sum of the products of the defined parameters and time t. In order for these values to be used by STICK, each value must be converted to a time interval, implying that two precisely timed spikes must be passed to the network. The beauty of piping the input directly from an event-based sensor is that such time encoding can be simple. Each event is already correlated with a single timestamp. Given a list of these times $e_n(i)$, the signal $u(t)$ can be accurately converted to a series of spiking events

for neuron n with the encoding function also used in [1], as follows:

$$u(t) = u(e_n(i)) = f^{-1}(e_n(i+1) - e_n(i)), \forall i = 2.p, p \in \mathbb{N} \tag{5.4}$$

where i is an even index of $e_n(i)$. The inverse function used in 5.4 is the inverse of the chosen encoding function. For proof of concept, linear encoding will suffice; however, in practicality, any of the aforementioned encodings (5.3) may be more appropriate depending on the size of the input values and precision requirements. The linear encoding function f for this implementation is the same as that used in Lagorce and Benosman, as it takes into account the refractory period of a neuron and satisfies our precision requirements:

$$\Delta t = f(x) = T_{min} + x.T_{cod}, \forall x \in [0, 1] \tag{5.5}$$

where Δt is the time interval between spikes, $T_{min} = 10ms$ allocates the minimum time between spikes to allow the neuron to reset, and $T_{cod} = 100ms$ as the smallest time step.

Both the horizontal motion prediction and vertical motion prediction rely on this encoding scheme for accurate data processing. The STICK implementation for each dimension of projectile motion is detailed in the following sections.

Processing the Horizontal Component of Projectile Motion

The horizontal position x of a projectile in motion is composed of the initial velocity v_x of the projectile and the time t at which the position occurs. The initial velocity of x, as calculated by the classic equation $velocity = \frac{distance}{time}$, which can be estimated for this use by

$$v_{n,n+1} = \frac{x_{n+1} - x_n}{t_{n+1} - t_n}, \tag{5.6}$$

where $v_{n,n+1}$ is the instantaneous velocity for any two consecutive x positions, x_n and x_{n+1} over the time span from t_n to t_{n+1}.

As shown in Figure 5.9, the STICK multiplier network is applied to the application of the ballistic model for the horizontal projectile motion. Using the two previous functions (5.4, 5.5), the initial horizontal velocity v_x is encoded as the time interval between two spikes,

Δt_{v_x}. The first neuron, $first1$, spikes, then after Δt_{v_x} has passed the second neuron, $last1$, spikes. The signal is fed to the STICK multiplication network, representing the v_x parameter of the ballistics equation. The second input to the multiplier component, $Input2$, is the timestamp t from the event-based sensor linearly encoded to the interspike time Δt_t between $first2$ and $last2$, just as we did for v_x. This network will be iterated through until the series of timestamps necessary for calculating future projected locations of the horizontal motion of the projectiles is complete. Since the projectile motion in the horizontal direction only advances in a positive direction, the output of the multiplier will always have a positive output. However, for more variable applications where horizontal movement may have backward or forward progress, the signed multiplier network allows for both negative and positive outputs, as shown by $Output+$ and $Output-$ in Figure 5.9. This functionality may be elaborated upon in future work.

Processing the Vertical Component of Projectile Motion

Consistent with the ballistic models of each direction of motion, the STICK implementation of the vertical component of the projectile's motion is more involved than that of the horizontal component's. The ballistic model of the vertical component described by Equation 5.2 can be broken down into more singular operations, each of which is computed by the respective computational network.

Since the two mathematical operator networks are the subtractor and multiplier, the ballistics model requires two minor adjustments. Rather than use t^2 for the second-degree polynomial, multiplication can be substituted for the square. Instead of summing the initial height h and the product of the initial velocity v_{0_y}, subtracting the negative value of v_{0_y} from h yields the same mathematical results. Thus, the adjusted ballistics model for the vertical component of the flight is:

$$y = h - (-(v_{0_y}t)) - (\frac{1}{2}gt)t \tag{5.7}$$

Next, the model must be subdivided into individual expressions of like operators in accordance with the mathematical order of operations. This is done by first computing the multiplication expressions: $v_{0_y}t$ and $(\frac{1}{2}gt)t$ Since the multiplier network only takes two inputs, the second expression is further broken down to $\frac{1}{2}gt$ and t. Because g is a predefined constant, $\frac{1}{2}g$ is a single known value that can be treated as a single input. The products from these subdivisions are then strategically computed using two subtractor networks. Figure

5.10 diagrams how the computational units are chained together to output the estimated y-position of the projectile. This displays the combination of three multiplier networks and two subtractor networks. As signals are passed throughout the fully integrated network, the synchronizers will be used to store the updated variables until each individual network has received the requisite spiking pair inputs.

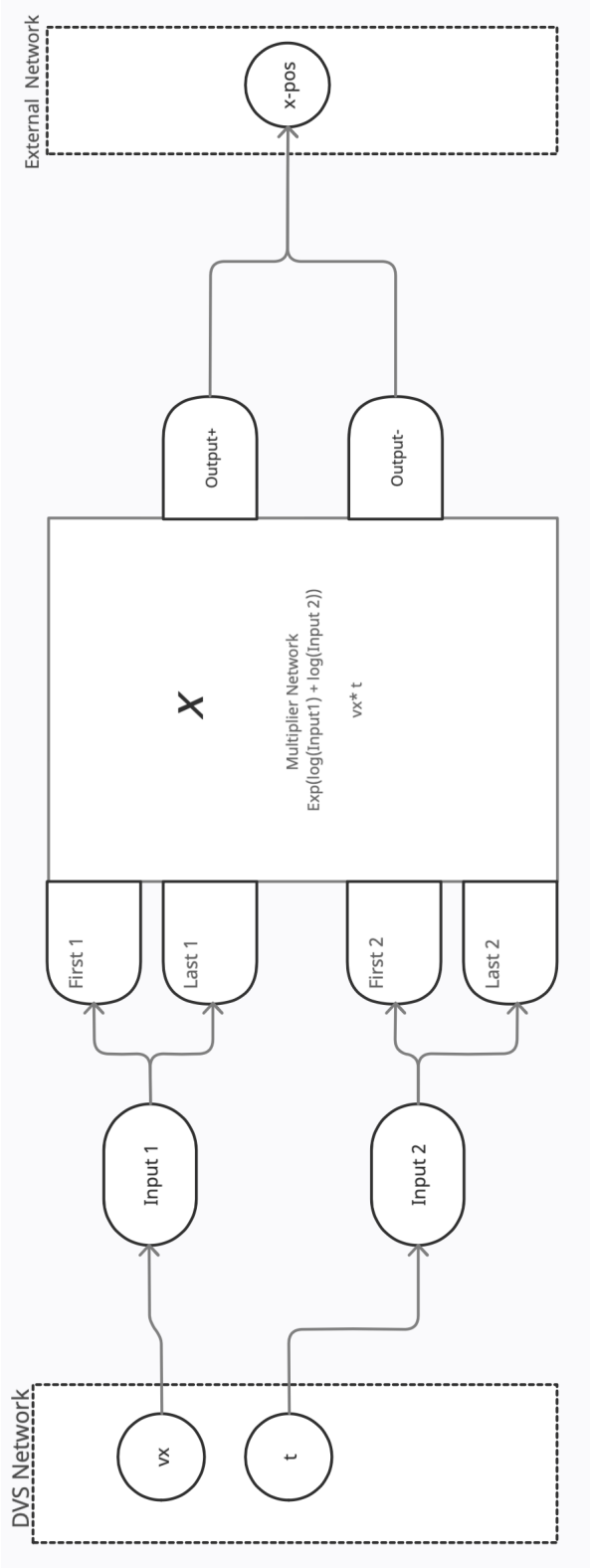


Figure 5.9. The multiplier network adapted for processing the horizontal component of the projectile's path.

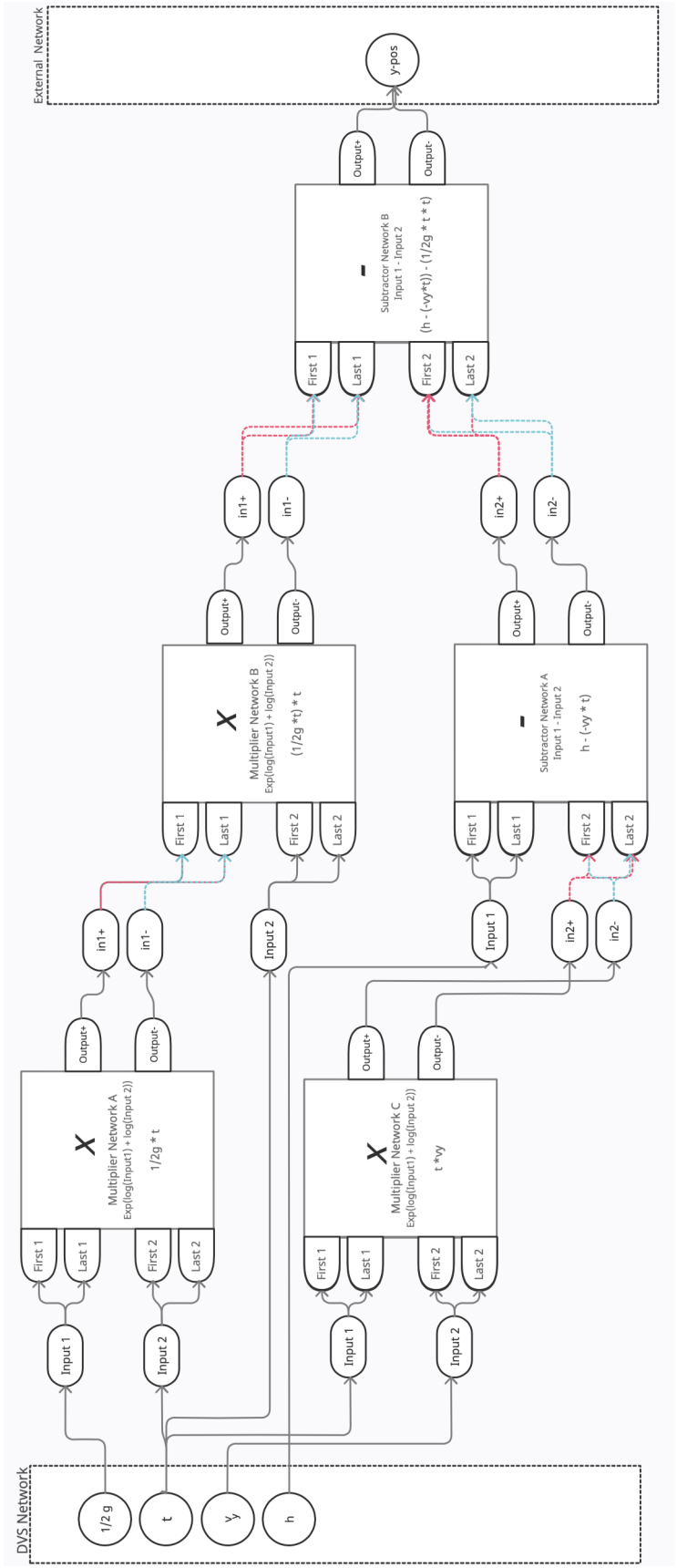


Figure 5.10. All computational units chained together for a fully integrated design for the predicted y-position of the projectile in flight.

CHAPTER 6:
Results and Analysis

After processing the data with the horizontal and vertical models, scatter plots of each component and the overlaid estimated data on the original data reveal the accuracy and error of the predictions. As expected, error accumulates with time resulting in the most inconsistencies appearing at the end of the flight path. In order to quantify the error for each trial, the Root Mean Squared Error (RMSE) of each is calculated and compared. Because error accumulates over time, comparing the error over various intervals of time assists in determining the largest time interval that maintains the highest accuracy. For practical applications, these statistics indicate how reliable a prediction may be at any given time from the first reading.

6.1 Results

The visual results of the predicted curves based on the ballistic models are reviewed in this section as well as the results of the calculated error for each the ballistic models. In an effort to quantify the error for thorough analysis, Mean Squared Error (MSE) and RMSE results are introduced.

6.1.1 Error Accumulation

Given a training portion of the data set, approximately twenty percent as mentioned previously, the estimated variables will inevitably result in the predicted curve being most accurate at the start of each trial, diminishing in accuracy farther along the curve. The error steadily accumulates as time increases, although the rate of accumulation varies from trial to trial, as seen in Figure 6.1. In the first trial (a), the error is not apparent until the curve surpasses the last real data point, whereas in trials (d), (e), and (f) significant error is apparent at or after the apex of the arc. These variations and their causes will be discussed in depth in the analysis section.

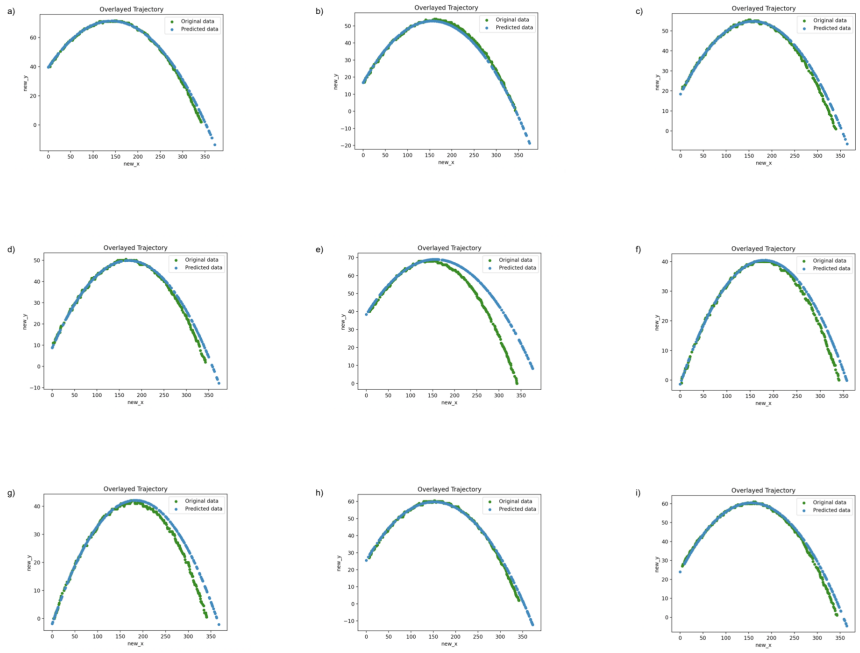


Figure 6.1. The new estimated positions are plotted as horizontal location versus vertical location. The rate at which error accumulates in each trial varies from the apex of the path to the very end of the trial.

6.1.2 Measurements of Error

Since using the curve_fit function to predict data points is the central aspect of predicting the projectile's flight path, the standard deviation of the predicted points from the original data points is an appropriate measure of error for regression models. To quantify the error accumulation, the RMSE best calculates the fit of a curve to a given set of points while presenting the error in the same units of as the data. To calculate the RMSE, first MSE is needed. The MSE finds the vertical distance r from each data point to the curve and squares that distance r to ensure all measurements for comparison are positive, thereby preventing negative values from negating the positive values [23]. All of these squared distances are averaged to determine the MSE for the data set and curve [23]. The RMSE is found by taking the square root of the MSE to better relate the error to the data. Therefore, the lower the MSE is, the lower the RMSE, and the better the fit of the curve to the data. However,

it should be noted that an RMSE of 0 indicates an perfect prediction, in which case the prediction modeling problem is trivial.

Mean Squared Error per Trial

For each trial, the MSE is calculated for the entire curve over all the real data points for that trial. To observe the change in the MSE as time increases, each trial is split into twelve sections: eleven parts of equal length and the last part the length remaining from the end of the eleventh to the completion of the trial. For each of these twelve parts, the MSE is calculated and appended to a text file for further analysis. One run of a single trial is displayed in the following Figure 6.2.

```
Estimated Mean Square Error for (x,y):
 81.36086481699441, 11.97502517972326
Trial split 0/6: Estimated Mean Square Error for (x,y)
 1.6640527795602897, 0.19572499336663535
Trial split 1/6: Estimated Mean Square Error for (x,y)
 2.890163183996484, 0.2636927957628132
Trial split 2/6: Estimated Mean Square Error for (x,y)
 1.8309240735404981, 0.2638949334191527
Trial split 3/6: Estimated Mean Square Error for (x,y)
 2.4327603040229016, 0.13954543666123068
Trial split 4/6: Estimated Mean Square Error for (x,y)
 4.659015599923483, 0.13772772906880015
Trial split 5/6: Estimated Mean Square Error for (x,y)
 4.950843188084174, 0.20072059188573005
Trial split 6/6: Estimated Mean Square Error for (x,y)
 7.737624805322158, 0.5967846266230387
Trial split 7/6: Estimated Mean Square Error for (x,y)
 34.814472911349576, 2.489039005053475
Trial split 8/6: Estimated Mean Square Error for (x,y)
 69.94577623937633, 5.64018483402984
Trial split 9/6: Estimated Mean Square Error for (x,y)
 135.51370992947378, 15.541615199064937
Trial split 10/6: Estimated Mean Square Error for (x,y)
 246.91940742669314, 34.88052275022183
Trial split 11/6: Estimated Mean Square Error for (x,y)
 477.0190347963019, 85.45351140474801
_____________________________Trial Complete 1_____________________________
```

Figure 6.2. The overall MSE for the trial is shown in the first two lines followed by the twelve splits, each with the calculated MSEs of the x and y components separately.

Root Mean Squared Error per Trial

Once the MSE is known, the RMSE is calculated to provide a more favorable measure for goodness of fit [23]. As seen in Figure 6.3, the RMSE for both x and y components start

small, much smaller than the overall RMSE, and climb to a higher value, measuring more than five times that of the overall RMSE. Depending on the trial, the splits begin to rise around the fifth or sixth time interval.

```
Estimated Root Mean Square Error for (x,y):
 9.020025762367014, 4.0646488538280146
Trial split 0/6: Estimated RMSE for (x,y)
 1.2899816974032525, 0.4442190413184962
Trial split 1/6: Estimated RMSE for (x,y)
 1.7000479954021368, 0.5054023966937493
Trial split 2/6: Estimated RMSE for (x,y)
 1.3531164305063703, 0.5200256332692947
Trial split 3/6: Estimated RMSE for (x,y)
 1.5597308447993423, 0.3728646695634239
Trial split 4/6: Estimated RMSE for (x,y)
 2.158475293577173, 0.3580390338607214
Trial split 5/6: Estimated RMSE for (x,y)
 2.22504902813309, 0.5691463427272219
Trial split 6/6: Estimated RMSE for (x,y)
 2.7816586398472243, 1.0028616229827383
Trial split 7/6: Estimated RMSE for (x,y)
 5.9003790439442785, 1.9985370076366793
Trial split 8/6: Estimated RMSE for (x,y)
 8.363359143166528, 2.991842401890915
Trial split 9/6: Estimated RMSE for (x,y)
 11.641035598441087, 4.7533969336466235
Trial split 10/6: Estimated RMSE for (x,y)
 15.713669438940018, 6.990106801106517
Trial split 11/6: Estimated RMSE for (x,y)
 21.840765428415295, 10.698112084872463
_____________________________Trial Complete 1_____________________________
```

Figure 6.3. The overall RMSE for the trial is shown in the first two lines followed by the twelve splits, each with the calculated RMSEs of the x and y components separately. While these values represent the same amount of error for each split, the value is significantly lower than the MSE values.

Considering that the linear model for the x-component only estimates one parameter while the parabolic model for the y-component estimates three parameters, the RMSE is affected based on the number of estimated parameters. This will have a slight impact on the fit of the y-component, but is negligible. The variation of the overall RMSE for the x-component for all trials ranges from 6.12 to 14.35. Meanwhile, the overall y-component RMSE ranges from 2.15 to a maximum of 3.93. The difference in ranges between the x and y components shows the x-component range to be 4.63 times larger that of the y-component's range. While the RMSE for the y-component is consistently much lower than that of the x-component for each time interval, this scales appropriately as x-component data has a wider range of values than the y-component does.

Scattered Index

In order to gauge the meaning of the RMSE values, a scattered index can be used to normalize the RMSE to the mean of the original data set. Additionally, the RMSE was used in calculating the scatter index of the error. The scattered index is calculated by dividing the RMSE by the mean of the original data, then multiplying the result by 100 to represent the metric as a percentage, as shown here:

$$SI = \frac{RMSE}{\bar{X}} * 100 \tag{6.1}$$

When calculating the scattered index for the predictions made by the curve_fit function, the overall scattered index is 4.158% for the x predictions and 6.949% for the y predictions. Within a given trial, the scattered index can vary from 0.46% to 14.22%, the last measurement showing the most error.

Therefore, these metrics show that the RMSEs, when normalized by the mean of the original data, show the model is 4.15% accurate for x and 6.95% accurate for y for the overall trial. Moreover, the model has an average of (0.46%, 0.97%) accuracy for the first two time blocks and can be used to predict the projectile's positions. While this model does not account for live positional updates to the location of the projectile, the overall RMSE and scattered index of each trial run provides an acceptable baseline of error for the implementation of this model.

6.2 Proportional Error to Time

As expected, the analysis of the RMSE results reveals that predicted x,y positions vary from the true data points the farther from the initial position as time increases. With each trial varying from 317 to 354 usable data points, splitting the trials into twelve time chunks provided appropriately-sized intervals for comparing the rate of increasing error. Each successive interval shows an accumulation of error until the final calculation shows the highest RMSE of that trial. As shown by Figures 6.4 and 6.5, the error tends to accumulate at an exponential rate.

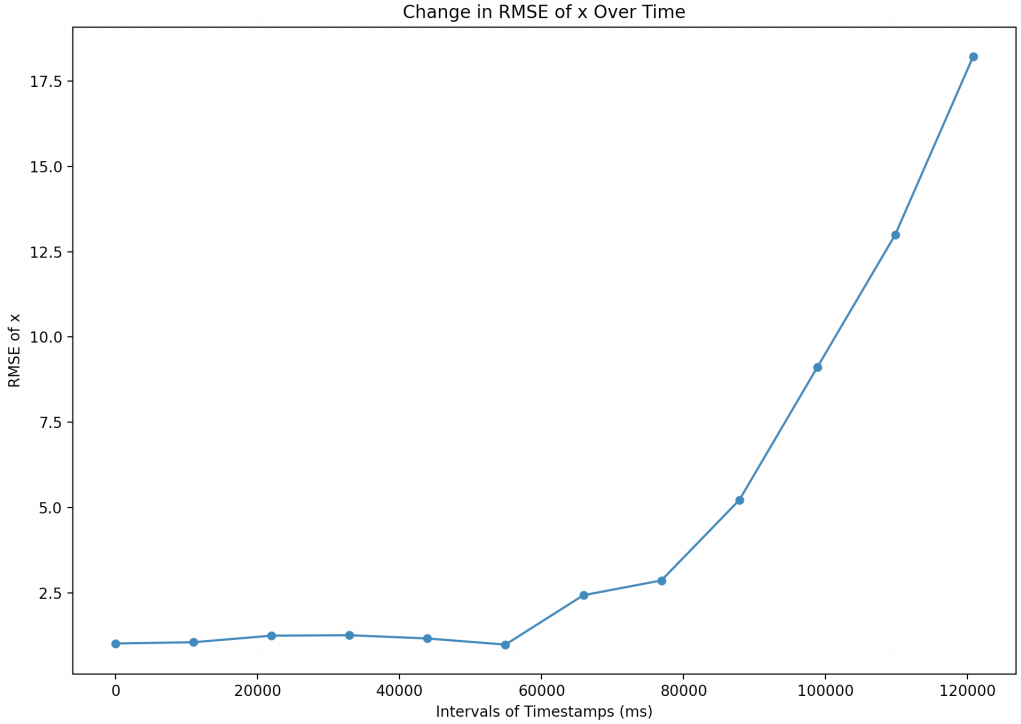


Figure 6.4. RMSE values computed for the estimated x-positions are plotted at each of the 12 time intervals. The error for the horizontal component grows nearly exponentially with time.

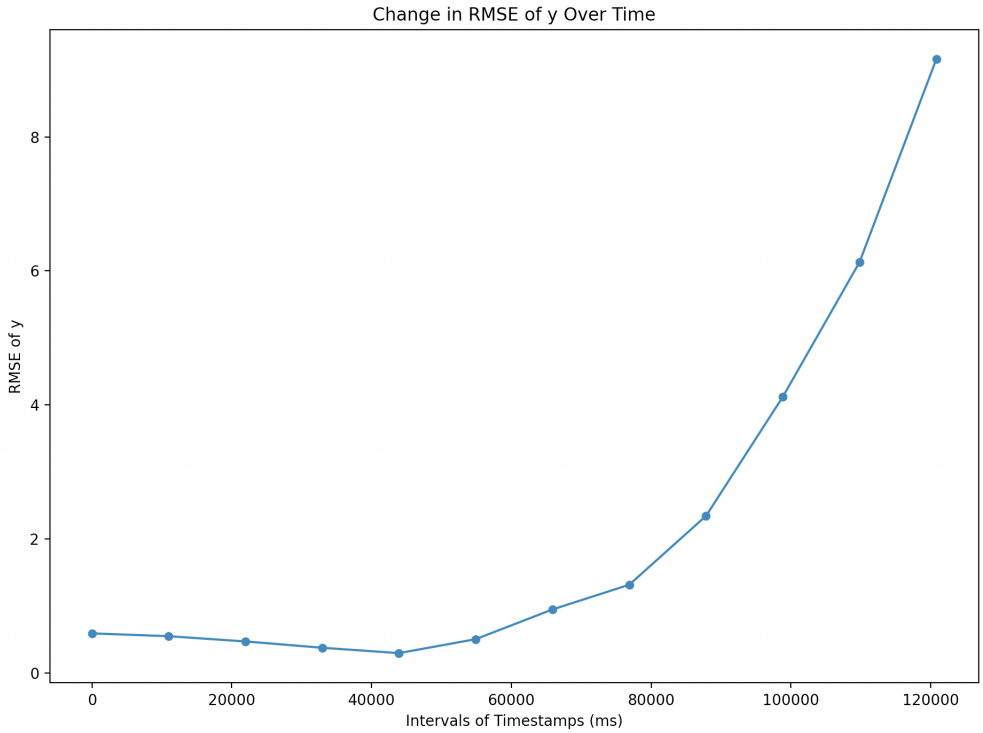


Figure 6.5. RMSE values computed for the estimated y-positions are plotted at each of the 12 time intervals. As expected, the error for the vertical component increases exponentially with time.

6.3 Suitability of DVS

As shown by the data collected with the DVS and then processed using the methods described in Chapter 5, this event-based pipeline is adequate for use in predictive models. By utilizing the curve_fit function, the values of parameters initial velocity, initial height, and initial angle trajectory can accurately be estimated and predictions can be made with a scatter index of 4.16% and 6.95% for x and y, respectively. This is consistent with an overall RMSE for x,y of 6.754, 2.786 for the same trial. Since this is measured over the entire trial, the scatter index is much lower for predictions made closer to the initial reading. The average over the first two time blocks is 0.46%, 0.97% for x and y respectively. The model is expected to receive updated readings before the projectile travels outside of an

acceptable threshold for the environment. These results support the original hypothesis that DVS sensors are suited to tasks like projectile tracking, omitting all of the computational overhead created by the processing of background data from frame-based recording devices. The leading edge of the selected projectile, a round white marble, is generously estimated to average four pixels in size in a shot. Since only the leading edge of the projectile is changing, roughly 0.00333% of the shot is being recorded as an event, compared to 100% of the frame being captured by frame-based sensors. Additionally, since frame-based sensors do not collect data continuously, data is missing from the projectile's path. An event-based camera captures events continuously, the data created by each capture of the projectile, 0.00333%, portion of the frame updating continuously over a two second period, does not amount to the data captured by the roughly 30 to 60 frames per second of a frame-based sensor.

6.4 Precise Timing Computation

In order to address the STICK pipelines laid out in Sections 5.3.2 and 5.3.2, we recognize that the ballistic model for the horizontal component ($v_0 t$) of the projectile trajectory is a subcomponent of the ballistic model for the vertical component ($h + v_0 t - \frac{1}{2} g t^2$). As such, the congruent portion of the vertical component's STICK pipeline (Figure 5.10) is the Multiplier Network C. Thus, the vertical component's STICK pipeline will be addressed first, and the reasoning applied to Multiplier Network C, by transitivity. The only difference between the horizontal component's pipeline and the Multiplier Network C component is the outputs are not swapped after the multiplier finishes, as they are for the Multiplier Network C, because no sign inversion is necessary for the horizontal component.

The fully implemented networks from Section 5.3.2 requires at minimum three multiplier networks and two subtractor networks. Each network has a synchronizer, which allows for each input to arrive at separate times. The accumulator is the component that stores input values until the synchronizer signals. When the last input is received, the synchronizer network is triggered, which in turn recalls stored values from the accumulator. Considering the sensitive timing of the spiking in each of the networks, the full implementation of the integrated computational units relies on the underlying individual proofs done by Lagorce and Benosman, as follows:

- Multiplier Network A handles the $\frac{1}{2} g t$ portion of the equation by taking two inputs:

the constant $\frac{1}{2}g$ and time t. Each input is encoded to a time interval Δt_{in1} and Δt_{in2} at $Input1$ and $Input2$, respectively. It follows that $Input1$ has a $first$ spike at time:

$$t^1_{first1} = t^1_{in1} + T_{syn} + T_{neur} \tag{6.2}$$

Similarly, the time of the $last$ spike is at:

$$t^1_{last1} = t^2_{in1} + T_{syn} + T_{neur} \tag{6.3}$$

The same applies to $Input2$, resulting in $first$ and $last$ spikes occurring respectively at times t^1_{first2} and t^2_{last2}. Once the $last$ spike of each input is propagated, the $sync$ neuron spikes at t_{sync}.

As the multiplier network computes the log of each input, it has two acc_log neurons, which the $sync$ neuron activates to trigger the readout of the values. The acc_log1 neuron has been integrating from time:

$$t^1_{acc1} = t^1_{first1} + T_{syn} + T_{neur} \tag{6.4}$$

] to time:

$$t^2_{acc1} = t^1_{last1} + T_{syn} \tag{6.5}$$

After t^2_{acc1} has passed, the membrane potential of acc_log1 is (from proof in [1]):

$$V_{sto1} = V_t \frac{\Delta T_{cod1}}{T_{cod}} \tag{6.6}$$

The same occurs in acc_log2, resulting in acc_log2 having an membrane potential of:

$$V_{sto1} = V_t \frac{\Delta T_{cod1}}{T_{cod}} \tag{6.7}$$

after time t^2_{acc1} passes.

To extract the sum of the outputs from acc_log1 and acc_log2, the activation of the acc_log1 activates the acc_log2. Specifically, the readout of acc_log1 at time $t^1_{st1} = t_{sync} + T_{syn}$ emits a spike at time t^1_{log1} (determined in [1]), triggering the readout

of acc_log2 at time $t^1_{st2} = t^2_{log1} + T_{syn}$, followed by a spike at time:

$$t^1_{log2} = -\tau_f log(\frac{\Delta T_{cod1}}{T_{cod}}) + t^1_{st} \tag{6.8}$$

This produces the first output spike at time:

$$t^1_{out} = t^1_{log2} + 2T_{syn} \tag{6.9}$$

which will be the input to the exponential network.
At the time that the synchronizer activates acc_log1, it simultaneously activates the acc_exp neuron by firing the exponential's $first$ neuron at time:

$$t^1_{st3} = t_{sync} + 3T_{syn}$$

The output spike from acc_log2 triggers the $last$ neuron of the exponential network at time:

$$t^1_{end3} = t^1_{log2} + T_{syn}$$

At this time, the acc_log2 neuron completes its integrating triggered by acc_log1, with a spike at time:

$$t^1_{exp} = t^1_{end3} + T_{cod}.e^{-\frac{1}{\tau_f}(t^1_{end3} - t^1_{st3})}$$

This spike stimulates the second and final output spike a time:

$$t^2_{out} = t^1_{exp} + T_{syn} + T_{min}$$

Thus, the output value from the network is encoded as ΔT_{out}, where:

$$\begin{aligned}
\Delta T_{out} &= t^2_{out} - t^1_{out} \\
&= (t^1_{exp} + T_{syn} + T_{min}) - (t^1_{log2} + 2T_{syn}) \\
&= (t^1_{end3} + T_{cod}.e^{-\frac{t^1_{end3} - t^1_{st3}}{\tau_f}}) + T_{min} - T_{syn} + (-\tau_f log(\frac{\Delta T_{cod1}}{T_{cod}}) + t^2_{st}) \\
&= T_{min} + T_{cod}.e^{-\frac{1}{\tau_f}(t^1_{end3} - t^1_{st3})}
\end{aligned} \tag{6.10}$$

- Multiplier Network B handles the $(\frac{1}{2}gt)t$ portion of the ballistics equation. It receives two inputs: the first, the output from Multiplier Network A, and the second, another timestamp value encoded as the time interval between the $first2$ and $last2$ neurons. Identical to the process in Multiplier Network A, the last action potential from inputs trigger the synchronizer to begin the cascade of activations. Each neuron is activated at exact the right timing such that the network outputs the product of the two input values accurately. This value is encoded as an interspike time interval and sent to Subtractor Network B.

- Multiplier Network C follows the same process as that of Multiplier Network A, the only differences are the portion of the ballistic equation that it handles and sign inversion on the output of the network. Multiplier Network C handles the $v_{0_y}t$ portion of the equation, thus, $Input1$ encodes the vertical initial velocity v_{0_y} and $Input2$ encodes the timestamp. The internals follow the same precise timing computations, resulting in a ΔT_{out} value encoded as the time interval between t^1_{out} and t^2_{out}. However, after the output action potentials fire, the $output-$ spike is sent to the $in2+$ neuron and the $output+$ spike is sent to the $in2-$ neuron. This accounts for the sign inversion necessary for this expression to be used in a subtractor network. Any positive value v_{0_y} will be inverted and subtracted from h, producing the equivalent to adding the original positive value. Any negative value v_{0_y} will be inverted to a positive value to be subtracted from h—the equivalent of adding the original negative value.

- Subtractor Network A handles the subtraction of the initial height h and the inverted output of Multiplier Network C, $-v_{0_y}t$, allotting to the subdivision, $h - (-v_{0_y}t)$, of the ballistics equation. Like the multiplier network, the subtractor network starts with the encoding of each input value as the time interval, the time between the $first$ and $last$ spikes for each input. $\Delta T_{in1} = t^2_{in1} - t^1_{in1}$ and $\Delta T_{in2} = t^2_{in2} - t^1_{in2}$ for $in1$ and $in2$, respectively. Relying on the synchronizer to start aligning the first spikes, the initial start time for both inputs is:

$$t^1_{in1} = t^1_{in2} \tag{6.11}$$

  Since $t^1_{in1} = t^1_{in2}$, the smaller value, encoded to the shorter ΔT_{in}, will inhibit the inhibitor neuron for the other input prior to that output being triggered. In the case

that the $input1$ is less than $input2$, $input1$ triggers $sync1$ at time:

$$t^1_{sync} = t^2_{in} + T_{syn} + T_{neur} \tag{6.12}$$

The $inb2$ neuron is then set to $-V_t$ after time T_{syn}, which triggers the $output-$ neuron at time: The spike from $sync1$ also excites the $output+$ neuron at time:

$$t^1_{out+} = T^1_{sync1} + 3T_{syn} + 3T_{neur} \tag{6.13}$$

However, slightly before the excitatory spike arrives at $output+$ at time:

$$t^1_{inb1} = t_{sync} + T_{syn} + T_{neur} \tag{6.14}$$

the inhibitory spike of weight $2w_i$ from $inhib1$ drops the $output+$ neuron's potential, preventing it from spiking.
By this time, the last spike of the larger input, $input2$, has triggered a spike at $sync2$ at time:

$$t^1_{sync2} = t^2_{in1} + T_{syn} + T_{neur}$$

as well as sets the $inhib1$ potential to $-V_t$ and resets $inb2$ and $output-$ back to resting potential. The second output spike is triggered at time:

$$\begin{aligned} t^2_{out-} &= t_{sync2} + T_{min} + 3.T_{syn} + 2.T_{neur} + T_{neur} \\ &= t^2_{in2} + T_{min} + 4.T_{syn} + 4.T_{neur} \end{aligned} \tag{6.15}$$

Thus, the correct negative output value is encoded as the time:

$$\begin{aligned} \Delta T_{out} &= t^2_{out-} - t^1_{out} \\ &= T_{min} + t^2_{in1} - t^2_{in2} \\ &= T_{min} + (t^2_{in1} - t^1_{in1}) - (t^2_{in2} - t^1_{in2}) \\ &= T_{min} + (\Delta T_{in1} - \Delta T_{in2}) \\ &= T_{min} + (\Delta T_{in1} - T_{min}) - (\Delta T_{in2} - T_{min}) \end{aligned} \tag{6.16}$$

- Subtractor Network B handles the final subtraction of the output of Multiplier Network B, $\frac{1}{2}gtt$, and Subtractor Network A, $h - (-(v_{0y}t))$, as first and second inputs,

respectively. This completes the computation using the same internal logic as that of Subtractor Network A to output the y-value for the predicted position of the projectile at time t.

This proof of concept demonstrates that a fully event-based pipeline can be created to track projectile motion. A pipeline that can simultaneously computationally process and store information in memory, thereby subverting the von Neumann architectural bottleneck.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7:
Conclusion and Future Work

7.1 Overall Conclusions

Utilizing an event-based pipeline, comprised of DVS and SNNs shows great promise in reducing resource requirements, increasing energy efficiency, and improving the processing speed and accuracy of the automated find, fix, and track components of the targeting kill-chain. The potential accuracy and reliability of event-based pipelines in light of their low-resource demands, make them especially well-suited for systems with high fidelity requirements. Building off of the STICK research conducted by Lagorce and Benosman, which demonstrated a framework with the capability to utilize a SNN to transmit data, this research shows that:

- The data produced by DVS with proper noise-filtering and data cleaning is of a suitable quality to be utilized in neuromorphic computing solutions as shown by the (0.46%, 0.97%) RMSE values for the x and y components of motion achieved in modeling the first and second time blocks, it is easily represented as spike pairs, and more closely mimics natural biological processes.
- The STICK framework can be utilized in combination with an event-based sensor, such as DVS, to create an event-based pipeline capable of real-time tracking. This pipeline is ideal for capturing objects moving at very high speeds that traditional frame-based sensors are known to portray inaccurately due to motion blur.
- The design and proof of this event-based pipeline created to mimic neurological processes is uniquely suited to automatic target detection and tracking. It is well suited to this application because it reduces the data used for identification and tracking by roughly 99%. This is especially useful to development of systems that must operate in constrained environments.

This research is fundamental to the future development of automatic detection and tracking of targets for critical systems with strict environmental constraints. This type of event-based pipeline is especially ideal for space-based projectile tracking. For instance, spacecraft out-

side of our atmosphere rely on high-speed surveillance to navigate a path with minimal collisions. Consider satellites that send and receive critical communication signals for our nation in orbit with fragments of debris — the satellites' abilities to avoid collisions is a large factor in their overall lifespan. The satellites must be designed in such a way as to minimize weight, while maximizing payload capability. The emergence of third generation AI technological capabilities have the potential to assist the U.S. in maintaining its competitive edge and assuring U.S. national security in the future. The reduction of processing requirements, coupled with the reduction of power requirements will allow for the development of systems with a smaller footprint that are capable of increased processing speeds and improved accuracy and precision. While this particular work is a proof of concept for future development, the underlying technology has various applications, including CPS.

7.2 Continuing Work

Building upon the research conducted to implement an event-based pipeline, the most imminent work carries forth the implementation of DVS event-based data processing and the diagrammed logic of the non-linear function with the STICK framework. Aside from this, suggested future work for other aspects of this pipeline includes:

- Testing DVS output for susceptibility to cybersecurity threats and attacks. DVS events rely on leading-edge indications, seen through the detection and movement of light. Attacks that alter the output of pixels, represented by x and y components, could have a significant impact on the data filtering, cleaning, and processing which could result in inaccurate or incomplete tracking data.
- Implementing a function in STICK to closely estimate the parameters, i.e. initial height and initial velocity of the projectile.
- Implementing event-based pipelines for other non-linear ballistics models that include varying air resistance, rebounds, collisions, and self-propulsion.
- Tracking multiple projectiles simultaneously, moving at varying speeds.
- Deploying multiple DVSs to capture various angles allowing for multi-dimensional tracking.
- Conducting a comparison of radar-based object tracking systems, such as the new space fence, to tracking capabilities with an event-based pipeline. Space fence relies on LRDR capabilities that are limited by scan rate; however, whether an event-based

pipeline can outperform these capabilities should be tested.

- Implementing this event-based pipeline and testing on recently developed neuromorphic computing solutions. Examples include IBM TrueNorth, Neurogrid, BrainScaleS system by the Kirchhoff Institute for Physics, and SpiNNaker at the University of Manchester. These systems are specifically designed to emulate biological processing capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] X. Lagorce and R. Benosman, "STICK: Spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony," *Neural Computation*, vol. 27, no. 11, Nov. 2015 [Online]. Available: doi:10.1162/NECO_a_00783

[2] G. Debat, T. Chauhan, B. R. Cottereau, T. Masquelier, M. Paindavoine, and R. Baures, "Event-based trajectory prediction using spiking neural networks," *Frontiers in Computational Neuroscience*, Mar. 2021 [Online]. Available: https://doi.org/10.3389/fncom.2021.6587642

[3] P. P. Urone and R. Hinrichs, *College Physics*. Houston, Texas: OpenStax, June 2012, ch. 26.1.

[4] D. Holmes, "Reconstructing the retina," Sep. 2018 [Online]. Available: https://www.nature.com/articles/d41586-018-06111-y

[5] Prophesee, "Event-based sensing enables a new generation of machine vision solutions," Nov. 2021 [Online], last accessed 2 February 2022. Available: https://www.prophesee.ai/wp-content/uploads/2021/11/PROPHESEE_Event_Based_Vision_White_Paper_November_2021.pdf

[6] lifeXchange, "Neural pathways: How your mind stores the info and thoughts that affect your behaviour," Jan. 2022 [Online]. Available: https://lifexchangesolutions.com/neural-pathways/

[7] E. A. Weaver II and H. H. Doyle, "Cells of the brain," Aug. 2019 [Online]. Available: https://www.dana.org/article/cells-of-the-brain/

[8] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, Dec. 1997 [Online]. Available: doi:10.1016/S0893-6080(97)00011-7

[9] B. Sciences, "Action potential," May 2016. Available: https://www.youtube.com/watch?v=HYLyhXRp298

[10] OpenStax, "Post synaptic potential summation," May 2016 [Online]. Available: https://cnx.org/contents/FPtK1zmh@8.25:fEI3C8Ot@10/Preface

[11] OpenStax, "Action potential," May 2016. Available: https://cnx.org/contents/FPtK1zmh@8.25:fEI3C8Ot@10/Preface

[12] D. Soni, "Spiking neural networks, the next generation of machine learning," Jan. 2018 [Online]. Available: https://towardsdatascience.com/spiking-neural-networks-the-next-generation-of-machine-learning-84e167f4eb2b

[13] NCLab, "John von Neumann and modern computer architecture," Feb. 2017 [Online]. Available: https://nclab.com/john-von-neumann-and-modern-computer-architecture/

[14] S. Magnuson, "News from space symposium: Tracking objects in space both easier, more complicated," Nov. 2019 [Online]. Available: https://www.nationaldefensemagazine.org/articles/2019/4/11/tracking-objects-in-space-both-easier-more-complicated

[15] NASA, "Space debris and human spacecraft," May 2021 [Online]. Available: https://www.nasa.gov/mission_pages/station/news/orbital_debris.html

[16] I. Stancic, M. Bugaric, and T. Perkovic, "Active IR system for projectile detection and tracking," *Advances in Electrical and Computer Engineering*, vol. 17, Nov. 2017 [Online]. Available: doi:10.4316/AECE.2017.04015

[17] L. Martin, "A new breed of radar," 2022 [Online]. Available: https://www.lockheedmartin.com/en-us/products/long-range-discrimination-radar.html

[18] iniVation, *User Guide - DAVIS346*, 2021 [Online], last accessed 1 November 2021. Available: https://inivation.github.io/inivation-docs/Hardware%20user%20guides/User_guide_-_DAVIS346.html

[19] iniVation, *Noise Filters*, 2021, last accessed 1 November 2021 [Online]. Available: https://inivation.gitlab.io/dv/dv-docs/docs/noise-filters/

[20] P. S. Company, *sys—System-specific*, Feb 2022 [Online]. Available: https://docs.python.org/3/library/sys.html#sys.maxsize

[21] Anomaly.io, "Moving median is robust to anomalies," Nov. 2015 [Online]. Available: https://anomaly.io/moving-median-robust-anomaly/index.html

[22] P. P. Urone and R. Hinrichs, *College Physics*. Houston, Texas: OpenStax, June 2012, ch. 8.6.

[23] J. Brownlee, "Regression metrics for machine learning," Jan. 2021 [Online]. Available: https://machinelearningmastery.com/regression-metrics-for-machine-learning/

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California