# Aimpoint Solutions on Complex Area Targets

**Matthew McLaughlin**
**Fires Battle Lab**
**Fort Sill, Oklahoma**
**matthew.b.mclaughlin4.civ@mail.mil**

## ABSTRACT

In artillery fires, the use of multiple aimpoints can spread damage over a large area when a unitary target location is inaccurate or when a collection of enemy targets span an area. Little research exists for optimizing aimpoint patterns to maximize effects. Current methods include only basic geometric shapes and software solutions that are often hard-coded and scaled for target areas. Since processing is not a major issue in modern computing, dynamic solutions are obtainable for irregular geometries and clustering of enemy detections. This research is a mathematical optimization solution that demonstrates how to tailor aimpoint configurations to complex area targets. Several factors are weighed when generating aimpoint configurations including the target-weight density function, weapon accuracy, type of damage, and radius of effects. The goal of this research is to reach greater effectiveness or preserve effectiveness against complex area targets with fewer rounds—saving the military the extensive logistic transportation requirements and ammunition costs.

## ABOUT THE AUTHOR

**Matthew McLaughlin** is a Lead Computer Scientist at the Fires Battle Lab in Fort Sill, OK. He has a B.A. in Mathematics and a B.S. in Computer Science from Cameron University. He is pursuing a MEng. in Modeling and Simulation from Arizona State University; he expects to graduate May 2020. He applies mathematical solutions to complex modeling problems which supports combat analysis and constructive simulation experiments. He received the Army Team M&S Award in 2016 for work done to advance constructive capabilities. He receive the Army Individual M&S Award in 2017 for his software innovations supporting terrain optimizations.

# Aimpoint Solutions on Complex Area Targets

**Matthew McLaughlin**
**Fires Battle Lab**
**Fort Sill, Oklahoma**
**matthew.b.mclaughlin4.civ@mail.mil**

## INTRODUCTION

Cannons first made their appearance on the battlefield in the 14th century as a means of engaging the enemy at long ranges and with a power that could destroy equipment, break down walls, and kill multiple enemy soldiers (Dastrup, 1992). Rocket fires made their way onto the battlefield in the late twentieth century offering longer range and a wider area for lethal effects on a target. Aimpoints have also been used to spread damage over a target area and can be a cheaper solution if used effectively. Consider an aiming strategy where multiple rounds are aimed at the center of a target. If the first detonation does a great amount of damage, each subsequent round will have less to destroy, therefore having less effective coverage than a strategy that spreads aimpoints over the whole target area. Exactly how aimpoints spread over a target will mostly depend on the target size and shape, target location error, and the number of rounds to deliver. Other factors include the accuracy of a sensor/observer in determining target location and size, firing unit location, weapon and round accuracy, and the radius of effects from each detonation.

The US Army analyzed the ballistic flight path of artillery and determined that five factors affected the ability to provide accurate fires (FM 3-09):

- *Accurate target size and location*. These metrics describe the known accuracy of the radar or observer. Typically azimuthal error (direction of target from sensor) is less in terms of distance than range error (distance to target from sensor). The ability to determine the size of the target area plays a great role in the spreading of aimpoints.
- *Accurate firing unit location*. The firing unit has better accuracy when close to the target. The trajectory (angle and velocity) plays a big role in impact accuracy and the shape of its explosion. At high velocities (direct fire), the damage pattern will become stretched. If a round hits a target from a high angle (indirect fire), the damage pattern will be more circular.
- *Accurate firing unit weapon and munitions information*. The known accuracy of a firing unit and the damage of their munitions also factor into aiming algorithms. If the fire unit's munition destroys a large area, less aimpoints will be needed to cover the full target area.
- *Accurate meteorological information*. This information is a collection of external factors that must be considered to achieve accurate fires including wind direction, wind speed, air temperature, and air pressure.
- *Accurate computational procedures*. The factors that affect accuracy in this category include procedure, training, and discipline. An optimized aimpoint solution will enhance this factor with minimal training.

These factors will be considered later when generating a model to represent munition accuracy and effects against any target area.

Most current methods to optimize aimpoints focus on geometric shapes with hard coded solutions, *but what about irregular shapes or clusters of enemy detections?* We could arbitrarily spread aimpoints throughout the target area, but if processing time is no longer an issue with modern computing, *why not tailor the aimpoint pattern to the particular target area?* Optimized aimpoint configurations for unitary target (Driels, Zhou, Wang, & Moten, 2016) and simple geometric shapes have been heavily explored. We build on this research by generalizing the shape of the target area to be any surface function. This surface function maps a location to a weight. This weight can quantitatively describe the amount of damage we want at that coordinate relative to any other coordinate. A region of high weight density will position more aimpoints around it than a region with low or no weight density.

## OPTIMIZED SOLUTION

This research demonstrates a possibility of tailoring aimpoints to a specific mission for the purpose of maximizing lethality and efficient use of rounds. This solution focuses on a relatively small collection of factors and relatively simple mathematical models for round-on-target error and damage to the target. The round-on-target error draws from shared and individual probability distributions. These errors are both approximated as bivariate normal distributions throughout this research and based on known capabilities of the weapon system, type of munition and fuse, and the gun-to-target distance (Anderson, 2004). Damage will be assessed from the Carleton distribution, but any damage function (Klopcic, 1990) (Way, 2015) can be used. The last factor is the weighted target area. This could be a polygon target area or disposition of enemy locations. As our algorithm attempts to maximize coverage, aimpoints will be pushed away from one another to minimize overlapping of effects. This will also balance against spilling effects outside the edges of the defined target area. This will cause aimpoints to be packed closer to where effects matter the most. All of these factors are considered when seeking an optimized aimpoint configuration.

### Range and Deflection, Bias and Error

For any collection of aimpoints, all rounds may experience a common force that pushes them off their intended course. These effects could come from unforeseen meteorological effects such as a gust of wind. Long time-of-flight or rounds that are light-weight will be more susceptible to this kind of error. This shared error is the *bias component of error* and pulls the mean point of impact (MPI) away from the aimpoint mean (which is often the center of the physical target). For our solution, we consider this bias to be drawn from a bivariate normal distribution having range and deflection variances ($\sigma_{Br}^2$ and $\sigma_{Bd}^2$, respectively) and no correlation ($\rho_B = 0$) between range and deflection.

Each round will also experience a component of error independent from other rounds. This error provides additional unforeseen effects that are unique to each round. This error can come from manufacturing, imbalance of charge powder, and subtle meteorological perturbations in each round's path not accounted for in the collective bias. For our solution, we consider this *individual component of error* to also be drawn from a bivariate normal distribution having range and deflection variances ($\sigma_r^2$ and $\sigma_d^2$, respectively) and no bivariate correlation ($\rho = 0$).

Detonations deviate from their corresponding aimpoints due to normally distributed shared and individual error. Therefore, the actual location of a detonation, $\vec{D}$, will generally have the following form:

$$\vec{D} = \vec{A} + \vec{X} + \vec{M} \tag{1}$$

To align the range, deflection, aimpoint, and detonation coordinates, we will place the center of the target area at (0,0) and rounds will move in a positive *y* direction (originating from below on a typical *x,y* graph). This will align the range to the *y*-axis and deflection to the *x*-axis. $\vec{A}$ is the location of an aimpoint (and desired point of impact), $\vec{X}$ is the individual component of error sampled from the $\mathcal{N}\left(\vec{0}, diag(\sigma_d^2, \sigma_r^2)\right)$ distribution:

$$f\left(\vec{X}\right) = \frac{1}{2\pi\sigma_d\sigma_r} e^{-\left[\frac{x^2}{2\sigma_d^2}+\frac{y^2}{2\sigma_r^2}\right]}, \text{ where } \vec{X} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

and $\vec{M}$ is the shared bias error for all rounds sampled from the $\mathcal{N}\left(\vec{0}, diag(\sigma_{Bd}^2, \sigma_{Br}^2)\right)$ distribution:

$$f_B\left(\vec{M}\right) = \frac{1}{2\pi\sigma_{Bd}\sigma_{Br}} e^{-\left[\frac{x^2}{2\sigma_{Bd}^2}+\frac{y^2}{2\sigma_{Br}^2}\right]}, \text{ where } \vec{M} = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{3}$$

To homogenize round aimpoint-detonation variances for all rounds and to align range and deflection to the *x* and *y*-axes, we make the following assumptions (approximations):

- *Rounds move in parallel paths along one gun-target line*---this allows us to approximate factors using one launcher, one type of munition and fuze, one trajectory path, and constant angle and speed of round impact. This also implies that the distance between firing elements is negligible.

- *Flat terrain*---the solution does not use terrain and therefore this solution will perform better for flat or approximately flat areas, thereby allowing normally distributed range/deflection errors. Future research may explore the distortions of impact error and damage distribution caused by mountainous or urban terrain.
- *Rounds are indistinguishable*---having the same properties from round to round. This allows us to treat aimpoints as a set rather than assigning a meaningful order and complicating the solution. In other words, a solution set containing $\{\vec{A}_1, \vec{A}_2\}$ is that same as $\{\vec{A}_2, \vec{A}_1\}$ and therefore not treated as another unique solution.

Given the number of aimpoints, *n*, the probability distribution function that captures all of these error components from Equations (2) and (3) can be written:

$$g(\vec{M}, \boldsymbol{D}, \boldsymbol{A}) = f_B(\vec{M}) \cdot \prod_{i=1}^{n} \left( f(\vec{D}_i - \vec{A}_i - \vec{M}) \right) \tag{4}$$

where:

$$\boldsymbol{D} = \begin{bmatrix} D_{1x} & D_{2x} & \cdots & D_{nx} \\ D_{1y} & D_{2y} & \cdots & D_{ny} \end{bmatrix} \text{ and } \boldsymbol{A} = \begin{bmatrix} A_{1x} & A_{2x} & \cdots & A_{nx} \\ A_{1y} & A_{2y} & \cdots & A_{ny} \end{bmatrix} \tag{5}$$

**Damage Function**

Damage functions describe the area surrounding a detonation as a function of position. Damage is intentionally vague because the desired effect may be to only suppress or disable enemy targets. Other situations may require catastrophic kills across the same target region. The shape of this damage function can change greatly depending on the defined damage threshold, but also the hardened nature of the target type (tank or personnel), target posture (upright or defilade), and the reliability of effects reaching the target (in the open, a building, or a forest). For simplicity, our research will consider the parameters for the damage function to be constant across all enemy targets and locations and defined by a Carleton damage pattern $\langle D_0, R_1, R_2 \rangle$. Carleton damage is a low-fidelity model describing the probability of damage around a detonation and has a bivariate normal shape. The parameters describe peak probability of damage at the center, $D_0$, and spread along the deflection and range axes, $R_1$ and $R_2$, respectively:

$$P_d(\vec{X}) = D_0 \cdot e^{-\left[\frac{(x-u)^2}{R_1^2} + \frac{(y-v)^2}{R_2^2}\right]}, \text{ where } \vec{X} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{6}$$

where $(u, v)$ is the location of the detonation and $(x, y)$ is the point being sampled for damage. When considering a fire mission with multiple detonations, think of a damage function as an image that will be copied and positioned over each detonation.

To standardize the desired effectiveness, we will only consider the probability of kill, $P_k$, as our damage threshold (everything else will scale) with the detonation centered at (0,0). The probability of kill for *n* detonations is one minus the probability that an enemy element at location, $\vec{X} = (x, y)$, survived all detonations, $\boldsymbol{D} = \left[\vec{D}_1, \vec{D}_2, \cdots, \vec{D}_n\right]$. This can be expressed as:

$$P_k^n(\vec{X}, \boldsymbol{D}) = 1 - \prod_{i=1}^{n} \left[1 - P_k(\vec{X} - \vec{D}_i)\right] \tag{7}$$

**Lethal Area**

We can combine Equations (4) and (7) and integrate over all possible detonation locations and shared error to generate an expected probability of kill at the location $\vec{X}$ given the selection of aimpoints, $\boldsymbol{A} = \left[\vec{A}_1, \vec{A}_2, \cdots, \vec{A}_n\right]$:

$$E[P_k^n] = K_n(\vec{X}, \boldsymbol{A}) = \iint \cdots \int_{\mathbb{R}^{2n+2}} P_k^n(\vec{X}, \boldsymbol{D}) \cdot g(\vec{M}, \boldsymbol{D}, \boldsymbol{A}) \, d\{\boldsymbol{D}\} d\{\vec{M}\} \tag{8}$$

Let $K_n$ be the *lethal area function* given *n* aimpoints and their locations, $\boldsymbol{A}$. Note that the integration expands over all $\mathbb{R}^{2n+2}$ space, this is because there is an *x* and *y* component of individual error for each aimpoint and for the collective bias error.

Since $P_k^n(\vec{X}, \boldsymbol{D})$ has two terms (one minus a product) as shown in Equation (7), we can distribute $g(\vec{M}, \boldsymbol{D}, \boldsymbol{A})$ into it and integrate each term separately. The fully integrated $\iint g \cdot d\{\boldsymbol{D}\} d\{\vec{M}\}$ term over all real space is 1, because it is a probability density function. We can also pair up the products from $P_k^n(\vec{X}, \boldsymbol{D})$ with their corresponding individual error terms $f(\vec{D}_i - \vec{A}_i - \vec{M})$ from Equation (2):

$$K_n(\vec{X}, \boldsymbol{A}) = 1 - \iint_{\mathbb{R}^2} f_B(\vec{M}) \cdot \left( \iint \cdots \int_{\mathbb{R}^{2n}} \prod_{i=1}^{n} \left[ \left(1 - P_k(\vec{X} - \vec{D}_i)\right) \cdot f(\vec{D}_i - \vec{A}_i - \vec{M}) \right] d\{\boldsymbol{D}\} \right) d\{\vec{M}\} \tag{9}$$

Each component of the product above depends only on its corresponding $\vec{D}_i$. We can expect this because each round has an *individual* error component that affects only its location of impact. Therefore, instead of having nested integrals, we can represent it by a product of integrals:

$$K_n(\vec{X}, \boldsymbol{A}) = 1 - \iint_{\mathbb{R}^2} f_B(\vec{M}) \cdot \prod_{i=1}^{n} \left[ \iint_{\mathbb{R}^2} \left(1 - P_k(\vec{X} - \vec{D}_i)\right) \cdot f(\vec{D}_i - \vec{A}_i - \vec{M})\, d\{\vec{D}_i\} \right] d\{\vec{M}\} \tag{10}$$

We can distribute again to pull out the one-minus term, but also the variables of integration, $\vec{D}_1, \vec{D}_2, \cdots, \vec{D}_n$, no longer need to be uniquely named. Thus we can apply the substitution, $\vec{D}_i = \vec{\omega}$:

$$K_n(\vec{X}, \boldsymbol{A}) = 1 - \iint_{\mathbb{R}^2} f_B(\vec{M}) \cdot \prod_{i=1}^{n} \left[ 1 - \iint_{\mathbb{R}^2} \left(P_k(\vec{X} - \vec{\omega})\right) \cdot f(\vec{\omega} - \vec{A}_i - \vec{M})\, d\{\vec{\omega}\} \right] d\{\vec{M}\} \tag{11}$$

Note that the deepest integrals represent a convolution. Now consider the case where $\vec{\sigma}_B$ approaches $\vec{0}$. This forces $f_B(\vec{M})$ to become a δ-distribution over $\vec{M} = \vec{0}$. By applying convolution properties, we can move the $\vec{A}_i$ term around, simplify the outer integral (effectively setting $\vec{M} = \vec{0}$), and define the following function:

$$K_n^0(\vec{X}, \boldsymbol{A}) = 1 - \prod_{i=1}^{n} \left[ 1 - \iint_{\mathbb{R}^2} P_k(\vec{X} + \vec{A}_i - \vec{\omega}) \cdot f(\vec{\omega})\, d\{\vec{\omega}\} \right] \tag{12}$$

Note that $K_n^0(\vec{X}, \boldsymbol{A})$ represents the same thing as $K_n(\vec{X}, \boldsymbol{A})$, but with zero bias error. Now consider this function with only one aimpoint, located at (0,0):

$$K^0(\vec{X}) = (P_k * f)(\vec{X}) = \iint_{\mathbb{R}^2} P_k(\vec{X} - \vec{\omega}) \cdot f(\vec{\omega})\, d\{\vec{\omega}\} \tag{13}$$

$K^0(\vec{X})$ represents the lethal area around one aimpoint—accounting for individual error, but not bias error. The equation above is a convolution, as represented by $(P_k * f)(\vec{X})$. In image processing, this represents a Gaussian filter with the parameters $\sigma_d$ and $\sigma_r$. Figure 1 demonstrates this blur to generate $K^0$. We can see that if $f(\vec{\omega})$ is a δ-distribution at 0 (representing no individual error) that $K^0$ and $P_k$ would become identical functions.

We can now combine Equations (12) and (13) rewrite $K_n^0$ in terms of $K^0$:

$$K_n^0(\vec{X}, \boldsymbol{A}) = 1 - \prod_{i=1}^{n} \left[ 1 - K^0(\vec{X} - \vec{A}_i) \right] \tag{14}$$

Now we have an image that represents a combination of images. This image represents a copy-paste of $K^0$ at each aimpoint, and sequentially combines them using the blending operator, $1 - (1 - source)(1 - destination)$. Figure 1 is generated using this operation for five aimpoints.

We can proceed one step further and combine Equation (11) and (14) to rewrite $K_n$ in terms of $K_n^0$ and simplify:

$$\begin{aligned} K_n(\vec{X}, \boldsymbol{A}) &= 1 - \iint_{\mathbb{R}^2} f_B(\vec{M}) \cdot \left(1 - K_n^0(\vec{X} - \vec{M}, \boldsymbol{A})\right) d\{\vec{M}\} \\ &= \iint_{\mathbb{R}^2} f_B(\vec{M}) \cdot K_n^0(\vec{X} - \vec{M}, \boldsymbol{A})\, d\{\vec{M}\} \\ &= (K_n^0 * f_B)(\vec{X}) \end{aligned} \tag{15}$$

This also represents a convolution with an image processing concept. $K_n$ is the resulting image of applying the filter $f_B$—a Gaussian blur defined by $\sigma_{Bd}$ and $\sigma_{Br}$—to $K_n^0$. Figure 1 demonstrates the blurring of $K_n^0$ to generate $K_n$. Similarly, if $f_B(\vec{M})$ defined a δ-distribution at 0, $K_n$ and $K_n^0$ would become identical functions.

**Weighted Target Area**

Our goal is to maximize the total weighted probability of kill. Weight is the degree to which the location $\vec{X}$ should experience damage relative to any other location. If we damage an area with a great weighted value, we are more successful in accomplishing the mission than if we were to spread damage away from it. Let $w(\vec{X})$ define the weight of a point in the target area. This function could describe a geometric shape where $w(\vec{X}) = 1$ inside the shape and $w(\vec{X}) = 0$ outside. It could also describe a probability density cloud around one or more point targets whose locations are less accurate. The weight could also be negative if an area should avoid damage, such as a church or school. This negative weight would have the effect of pushing aimpoints away. The function $w(\vec{X})$ should also be defined such that the weight approaches zero (or equals zero) in all directions approaching infinity to guarantee finite solutions.

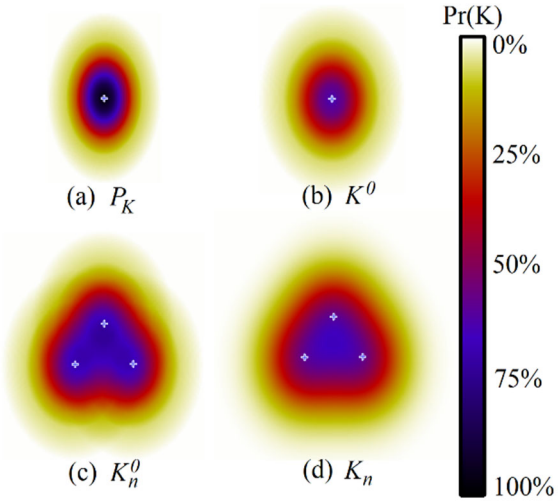


**Figure 1. Damage Pattern and Coverage Functions**

Let $F(\boldsymbol{A})$ be the expected total weighted probability of kill over the target area when given a collection of aimpoints, or:

$$F(\boldsymbol{A}) = \iint_{\mathbb{R}^2} \left[K_n(\vec{X}, \boldsymbol{A}) \cdot w(\vec{X})\right] d\{\vec{X}\} \tag{16}$$

In image processing, $K_n(\vec{X}, \boldsymbol{A}) \cdot w(\vec{X})$ represents an image created by blending $K_n$ and $w$ with the multiply operator. The function $F(\boldsymbol{A})$ could be computed by summing the pixels of that image into a single number. Our goal is to maximize this number, and we can maximize it by adjusting the location of each aimpoint (iteratively) to a more optimized location. We achieve a local maximum (or other extrema) when $\nabla F = \vec{0}$ and thus define our ∇ operator to be the following $2n \times 1$ vector:

$$\nabla^{\mathrm{T}} = \left[\frac{\partial}{\partial A_{1x}} \quad \frac{\partial}{\partial A_{1y}} \quad | \quad \frac{\partial}{\partial A_{2x}} \quad \frac{\partial}{\partial A_{2y}} \quad | \quad \cdots \quad | \quad \frac{\partial}{\partial A_{nx}} \quad \frac{\partial}{\partial A_{ny}}\right] \tag{17}$$

Differentiating Equation (15) with the ∇ operator from Equation (17) and setting the equation to zero gives us:

$$\nabla F(\boldsymbol{A}) = \iint_{\mathbb{R}^2} \left[\nabla K_n(\vec{X}, \boldsymbol{A}) \cdot w(\vec{X})\right] d\{\vec{X}\} = \vec{0} \tag{18}$$

The ∇ operator has no effect on the weight function, $w(\vec{X})$, because it does not depend on aimpoint locations. Differentiating on Equation (15), the value of $\nabla K_n(\vec{X}, \boldsymbol{A})$ will be:

$$\nabla K_n(\vec{X}, \boldsymbol{A}) = \iint_{\mathbb{R}^2} f_B(\vec{M}) \cdot \nabla K_n^0(\vec{X} - \vec{M}, \boldsymbol{A})\, d\{\vec{M}\} \tag{19}$$

The ∇ operator also has no effect on bias component of error which depends solely on the vector $\vec{M}$. Differentiating on Equation (14) cannot be represented without differentiating on each *xy* pair (which are separated by bars) of ∇ from Equation (17). Therefore, consider only the *i*-th element:

$$\begin{aligned}\nabla_i K_n^0(\vec{X}, \boldsymbol{A}) &= -\nabla_i \prod_{j=1}^{n}\left[1 - K^0(\vec{X} - \vec{A}_j)\right] \\ &= \prod_{\substack{j=1 \\ j\neq i}}^{n}\left[1 - K^0(\vec{X} - \vec{A}_j)\right] \cdot \begin{bmatrix} K_x^0(\vec{X} - \vec{A}_i) \\ K_y^0(\vec{X} - \vec{A}_i) \end{bmatrix}\end{aligned} \tag{20}$$

Note that $\nabla_i$, the *i*-th element of $\nabla$, is a $2 \times 1$ vector because it represents a 'sub-'gradient on the *xy* components of the *i*-th aimpoint. The resulting $\nabla_i K_n^0$ represents two images (one for the *x* differential and one for the *y* differential on the *i*-th aimpoint). Let $K_x^0$ be defined as the partial derivative of $K^0$ along the $\partial x$ direction and likewise $K_y^0$ along the $\partial y$ direction. Equation (20) describes how the lethal area increases or decreases as the *i*-th aimpoint is moved around the other aimpoints (excluding bias error). Up until now, the exact pattern of $P_k$ has been in support of a general solution, but we will use the Carleton damage pattern so that we can further evaluate $K_x^0$ and $K_y^0$.

**Carleton Damage Function**

We can combine Equations (6) with (13) and fully evaluate the integrals to determine $K^0$:

$$K_0(\vec{X}) = \frac{D_0 \cdot R_1 \cdot R_2 \cdot e^{-\left[\frac{x^2}{2 \cdot D_0 \cdot \sigma_d^2 + R_1^2} + \frac{y^2}{2 \cdot D_0 \cdot \sigma_r^2 + R_2^2}\right]}}{\sqrt{(2 \cdot D_0 \cdot \sigma_d^2 + R_1^2)(2 \cdot D_0 \cdot \sigma_r^2 + R_2^2)}}, \text{ where } \vec{X} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{21}$$

The Carleton can be a useful approximation because it simplifies and removes the integrals from Equation (13). In fact, the new pattern, $K_0(\vec{X})$, can be written as a Carleton:

$$K_0(\vec{X}) = D_0' \cdot e^{-\left[\frac{x^2}{(R_1')^2} + \frac{y^2}{(R_2')^2}\right]}, \text{ where } \vec{X} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{22}$$

with the following parameters:

$$D_0' = \frac{D_0 \cdot R_1 \cdot R_2}{\sqrt{(2 \cdot D_0 \cdot \sigma_d^2 + R_1^2)(2 \cdot D_0 \cdot \sigma_r^2 + R_2^2)}}, \; (R_1')^2 = \frac{D_0'}{D_0}(2 \cdot D_0 \cdot \sigma_d^2 + R_1^2), \text{ and } (R_2')^2 = \frac{D_0'}{D_0}(2 \cdot D_0 \cdot \sigma_r^2 + R_2^2) \tag{23}$$

To satisfy all elements from Equation (20), we need to find the differentials. Due to the exponential nature of the Carleton damage pattern, we can write the differentials, $K_x^0$ and $K_y^0$, in terms of $K_0$:

$$K_x^0(\vec{X}) = \frac{-2 \cdot D_0' \cdot x}{(R_1')^2} \cdot K_0(\vec{X}) \text{ and } K_y^0(\vec{X}) = \frac{-2 \cdot D_0' \cdot y}{(R_2')^2} \cdot K_0(\vec{X}), \text{ where } \vec{X} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{24}$$

**Converging on a Solution**

Equation (18) is used to test a proposed aimpoint configuration as an extrema. If the value is non-zero, we should move our aimpoints (the elements of $\boldsymbol{A}$) toward the direction of a local maximum. Solving $\nabla F(\boldsymbol{A}) = \vec{0}$ completely will produce a set of extremas. The value of $F(\boldsymbol{A})$ should be calculated for each extrema and the highest value from that set will yield an optimal aimpoint configuration. Requiring that the damage and target weight functions reduce to 0 at the infinities ensures the optimal solution has aimpoints with finite coordinates.

We may not be able to identify all solutions for $\nabla F(\boldsymbol{A}) = \vec{0}$ because there may be infinite solutions. However, we can repetitively test a finite collection of aimpoint configurations, identify local extrema, and test for the one(s) that maximize $F(\boldsymbol{A})$, but our best solution is only limited to the configurations we have sampled. In general, the absolute maximum cannot be guaranteed computationally.

A technique of generating these sets are to randomly place aimpoints about the target area (sampling the weight density function as as a probability density), solve for the local extrema, and repeat the process. The number of repetitions will depend on processor speeds and the acceptance threshold of near-optimal solutions.

**Weight Function Implementations**

The weight function has been an abstract concept until now. We have described it as a surface function that aligns a weighted distribution over a target. The following subsections describe implementations of the weight function that are useful in filling mission capability gaps. The first implementation aims to fill a target area uniformly with rounds when specific enemy locations are not known or their locations are spread evenly throughout the target area. The second implementation optimizes lethality on enemy locations that are either known or have a probability density. This implementation can also optimize considering high and low payoff targets.

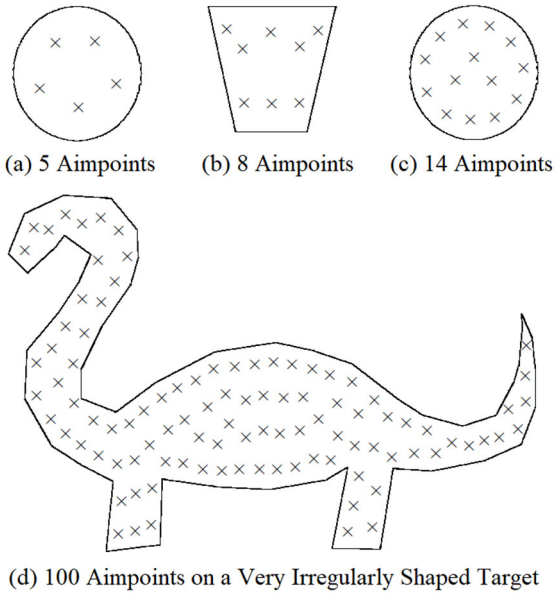


**Figure 2. Simple and Complex Polygon Solutions**

**Polygon Target**

The weight function for this implementation is very simple. Since $F(\boldsymbol{A})$ is the objective function on which we will optimize, any scalar can be divided out from Equation (16) because optimization will yield the same aimpoint solution. Therefore, if the point, $\vec{X}$, is located within a described polygon, the value of $w(\vec{X})$ will be one, otherwise $w(\vec{X}) = 0$. Finally, we engage this target area and let the aimpoint algorithm decide the best configuration of aimpoints. The algorithm will balance two forces when converging to a solution: the interaction an aimpoint has with the edge of the polygon—which pulls aimpoints towards the center of the target area—and the aimpoint interactions—which forces aimpoints away from one another to reduce overlap of effects.

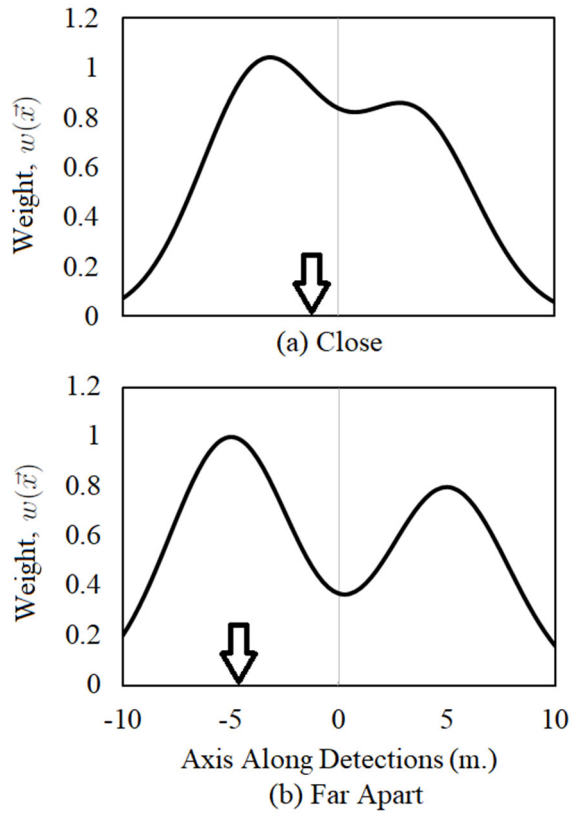


**Figure 3. Single Aimpoint Comparison on Enemy Detection Spacing**

**Enemy Detections**

The weight function for this implementation is more complex; it most resembles a linear combination of probability density functions for every detection (optionally dropping the scalar $2\pi$). The target location error (TLE) describes the variance from a detection that we believe an enemy to be. The priority scales this surface function so that high payoff targets have a higher peak than lower payoff targets. Let $\boldsymbol{P}$ be a set of four dimensional tuples describing each enemy location ($u$ and $v$), target location error ($\sigma^2$), and priority ($p$). The weight function will be generated by:

$$w(\vec{X}) = \sum_{(u,v,\sigma^2,p)\in \boldsymbol{P}} \left[ \frac{p}{\sigma^2} \cdot e^{-\frac{(x-u)^2+(y-v)^2}{2\cdot\sigma^2}} \right] \qquad (25)$$

$$\text{where } \vec{X} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Similar to the normal distribution curve, as $\sigma^2$ increases, the peak of its detection will become smaller, conversely as it decreases, the peak will become taller. The priority of an enemy detection allows an inaccurate detection of a high payoff target to compete with an accurate detection of a low payoff target. We can expect a small number of aimpoints to converge over enemy locations with the smallest TLE and higher payoff targets. See Figure 3 to see how this algorithm may fare against a cluster of enemy

detections. This figure represents the profile of a weight function along the line between two enemy detections and the location along that profile where a single aimpoint is optimal. The graphic on the left demonstrates how one aimpoint (shown by the arrow) may strategically destroy two enemy targets given they are close enough to the anticipated detonation. The graphic on the right shows how choosing to engage one detection is more optimal than attempting to engage both.

## IMPLEMENTATION

Image processing has been discussed throughout this research because the techniques used in image processing can be applied to the computation of damage coverage and its differentials. Since we are only concerned with the assessment of effects on a two-dimensional plane, we can represent the damage pattern, weight function, and coverage as a two-dimensional images. If we view these surface functions as images, we can utilize existing SIMD (single instruction/multiple data) instruction sets. These instructions are used heavily in image processing because the same instructions can be performed on multiple pixels concurrently. Numerous image manipulation libraries exist to accelerate image processing in this way. Instead of using all of the color channels, we only need a single channel to represent these surface functions. It may be easier to consider this channel as a grayscale image where each pixel represents some value between 0% (black) and 100% (white). Also, the equations given in this section will be more useful for the implementation of this algorithm because they focus on operations and not just theory.

To move from $P_k$ to $K^0$ in Equation 13 or from $K_n^0$ to $K_n$ in Equation 15, a Gaussian blur is performed. This is a common operation used by image manipulation software to make part of image appear out of focus. We can relate this operation to our surface functions as:

$$K^0 = (P_k * f) = blur(P_k, \sigma_d, \sigma_r), K_n = (K_n^0 * f_B) = blur(K_n^0, \sigma_{Bd}, \sigma_{Br}) \tag{26}$$

where,

$$blur\left(\left(\boldsymbol{Z}, \sigma_x, \sigma_y\right) \circ (x, y)\right) = \iint_{\mathbb{R}^2} \frac{\boldsymbol{Z}(x-u, y-v)}{2\pi\sigma_x\sigma_y} e^{-\left[\frac{u^2}{2\sigma_x^2} + \frac{v^2}{2\sigma_y^2}\right]} dudv \tag{27}$$

To move from $K^0$ to $K_n^0$ in Equation 14, we composite $K^0$ into a single image after each $K^0$ has been translated to an aimpoint location, $A_i$. This blending is performed with the combine operation. Let $K_n^0 = \boldsymbol{Z}_n$ where:

$$combine\big((\boldsymbol{P}, \boldsymbol{Q}) \circ (x, y)\big) = 1 - (1 - \boldsymbol{P}(x, y))(1 - \boldsymbol{Q}(x, y)) \tag{28}$$

$$translate\big((\boldsymbol{P}, u, v) \circ (x, y)\big) = \boldsymbol{P}(x - u, y - v) \tag{29}$$

$$\boldsymbol{Z}_i(x, y) = \begin{cases} 0 & , i = 0 \\ combine\left(\left(\boldsymbol{Z}_{i-1}, translate(K^0, A_i)\right) \circ (x, y)\right) & , else \end{cases} \tag{30}$$

To incorporate the weight function, we blend $K_n$ with the weight function in Equation 16 using multiplication before summing up all pixel values. This sum represents the result of an objective function given a collection of aimpoints:

$$multiply\big((\boldsymbol{P}, \boldsymbol{Q}) \circ (x, y)\big) = \boldsymbol{P}(x, y) \cdot \boldsymbol{Q}(x, y) \tag{31}$$

$$sum(\boldsymbol{P}) = \sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} \boldsymbol{P}(x, y) \tag{32}$$

$$F = sum\big(multiply(K_n, w)\big) \tag{33}$$

The differentials under $\nabla F$ will be setup similarly, but these underlying images must handle the representation of a positive or a negative slope. The blurring operation defined by Equation 27 does not change. Thus,

$$\nabla K^0 = blur(\nabla P_k, \sigma_d, \sigma_r), \nabla_i K_n = blur(\nabla_i K_n^0, \sigma_{Bd}, \sigma_{Br}) \tag{34}$$

$$\nabla_i F = sum\big(multiply(\nabla_i K_n, w)\big) \tag{35}$$

The effort to transform multiple $K^0$ into a single $\nabla_i K_n^0$ image will be handled differently as shown in Equation 20. If we differentiate on aimpoint *i*, then all other $K^0$ images are combined into a single image as demonstrated in Equations 28 through 30, but this result is excluded from the differential $\nabla K^0$ for aimpoint *i*. This behavior is given by the following equations:

$$\boldsymbol{Z}_{i,j}(x,y) = \begin{cases} 0 & ,j = 0 \\ \boldsymbol{Z}_{i,j-1}(x,y) & ,j = i \\ combine\left(\left(\boldsymbol{Z}_{i,j-1}, translate\left(K^0, A_j\right)\right) \circ (x,y)\right) & ,else \end{cases} \tag{36}$$

$$exclude\big((\boldsymbol{P},\boldsymbol{Q}) \circ (x,y)\big) = \boldsymbol{P}(x,y) \cdot (1 - \boldsymbol{Q}(x,y)) \tag{37}$$

$$\nabla_i K_n^0 = exclude\big(translate(\nabla K^0, A_i), \boldsymbol{Z}_{i,n}\big) \tag{38}$$

**Determining the Number of Aimpoints**

The number of aimpoints is generally not a desired metric to use because it depends on the target area, the munition size, and the desired effectiveness for the mission. As we use optimal locations for *n* aimpoints, we can expect the optimal solution to become more effective with each additional aimpoint. We can also expect the effectiveness to asymptotically approach 100% coverage. Figure 4 demonstrates the diminishing returns on additional aimpoints for select missions. These graphs show the diminishing returns from adding additional aimpoints. Both graphs have selected three munitions capable of destroying 10% and 30% of the target area. The figure on the left is computed from a circular target. The figure on the right was computed from a very complex polygon. The graphs in Figure 4 are similar and may suggest that the desired number of aimpoints is based on target area and effects area has very little correlation to the complexity of the shape. This figure can be used to create a simple heuristic to determine the number of aimpoints needed for a mission without processing.

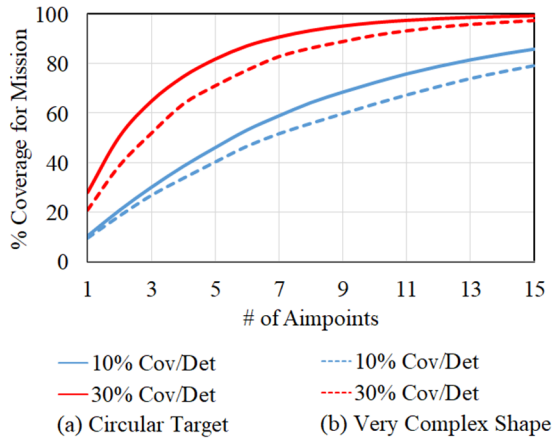


**Figure 4. Comparing Diminishing Returns Between Shape Complexity**

**Computational Performance**

Many factors will have an effect on how long it will take this algorithm to produce useful results. Since repetitions are sampled independently, we can expect the total time taken to be a multiple of the mean time for one repetition. Therefore, the number of repetitions will have a linear effect on the total time. Since repetitions also do not interact with one another, computations for each repetition can be done on its own processor or thread. However, most factors will not have a linear complexity. For instance, the number of aimpoints will increase the number of interactions with other aimpoints in a quadratic complexity (each new aimpoint creates $n - 1$ interactions with other aimpoints). Figure 5 demonstrates this behavior on a circular target.

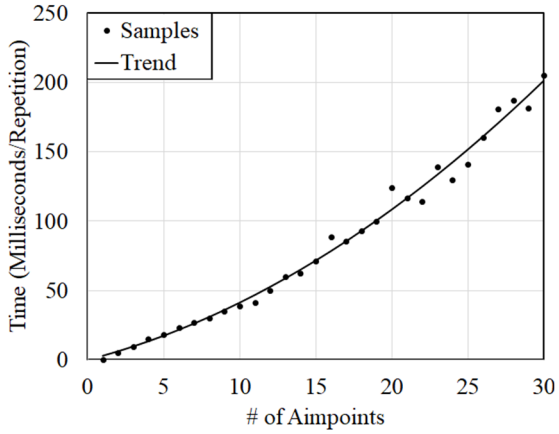


**Figure 5. Computational Performance**

**FUTURE WORK**

This optimization model is timeless. Unless all rounds can be delivered simultaneously, only the first few rounds will catch the enemy off-guard. As the enemy becomes

defilade, the effective burst radius of rounds being fired will diminish. A future aimpoint optimization algorithm may take first-round fire-for-effect into consideration. One may consider optimizing on a few aimpoints for maximum coverage and simultaneous impacts. The effects from these rounds will then be used to distort the weight function for the expected kills. Additional aimpoints are then optimized over the protected enemy in the modified weight function. Adding time into the model also allows the algorithm to become more dynamic. Consider the round-impact errors mentioned earlier. If we fire a round, our sensors can tell us where the round will hit. This feedback could change the optimal locations for rounds not yet fired.

Volume of fire has not been addressed because mathematically it is not optimal to fire multiple rounds at a single aimpoint, but slewing a cannon to a new aimpoint takes time. To avoid slewing time, round-impact deviations may be considered adequate for spreading effects. If a large quantity of cannon rounds will be used, but only a few cannon are assigned, consider optimizing a single aimpoint for each cannon to shoot multiple rounds. Consider how the effects of $K^0$ will change. Using Equation 13, consider only the aimpoint located at (0, 0) with $n$ rounds. After all rounds have hit, the expected probability of kill from this aimpoint at location $\vec{X}$ will be $1 - (1 - K^0(\vec{X}))^n$.

## CONCLUSION

In this paper, we have introduced a mathematical modeling algorithm to advance aimpoint configurations on complex target areas. Current aimpoint configurations are limited to cookie-cutter distributions best fitted to regular shapes. The new algorithm frees users from many of the current limitations. It considers number of aimpoints, the range/deflection errors (both shared and individual), the weight function, and the damage function then produces an optimal configuration of aimpoints. This new methodology gives users six new capabilities. First, the new aimpoint algorithm frees one from the constraints of regular-shaped target areas. It allows planners to define irregular-shaped targets. Second, the new algorithm has the analysis tools to tell us how many aimpoints are required to cover a target area with effects. Third, it balances two forces -- pushing aimpoints away from target area boundaries to limit spillage of effects outside the target area, while pushing aimpoints away from one another to avoid excessive overlap. Fourth, it applies a uniform distribution to irregular-shaped target areas and generates an optimal spread of effects throughout the area. Fifth, it considers actual enemy detections in an area, and generates a weighted surface area distribution to concentrate aimpoints on specific areas where enemy equipment is positioned. Last, it allows users to assign high priority to specific types of enemy equipment. These priorities factor into the weighted surface area distribution to ensure a higher concentration of aimpoints over such equipment. Together, these added features give users a more robust set of capabilities to better plan and deliver effects to target areas, while lowering resource requirements such as cost, transportation, loading, and manufacture of munitions.

## REFERENCES

Dastrup, B.L. (1992). *King of Battle: A Branch History of the U.S. Army's Field Artillery*. Office of the Command Historian, U.S. Army

Wang, H.Y., Moten, C., Driels, M., Grundel, D., & Zhou, H. (2016). Explicit Exact Solution of Damage Probability for Multiple Weapons Against a Unitary Target. *American Journal of Operations Research*, *6*, 450-467.

Black, A., Mahoney, I.F., Sivazlian, & B.O. (1985). *Variability of Measures of Weapons Effectiveness*. Air Force Armament Laboratory.

Chusilp, P., Charubhun, W., & Kanantchai, P. (2014). Monte Carlo Simulations of Weapon Effectiveness Using $P_k$ Matrix and Carleton Damage Function. *International Journal of Applied Physics and Mathematics*, *4*, 280-285.

Klopcic, J.T. (1990). *A Comparison of Damage Functions for Use in Artillery Effectiveness Codes*. Ballistic Research Laboratory.

Anderson, C.M. (2004). *Generalized Weapon Effectiveness Modeling*. Masters Thesis, Naval Postgraduate School, Monterey, CA.

Department of the Army, Washington, DC. (2014) *Field Artillery Operations and Fire Support (FM 3-09)*.

Way, J. (2015). *Fragmenting Munition Lethality Data Compression by Neural Network Regression*. Technical Report, US Army Materiel Systems Analysis Activity.