SANDIA REPORT
SAND2019-15200
Printed December 2019

Sandia National Laboratories

Mobile Device Application for Gunshot Trilateration

Stephanie Booth, Jeff Carlson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

The New Mexico Small Business Assistance program at Sandia National Labs and Parental Values, LLC have agreed to explore commonly known principles to describe techniques in trilateration. The objective from Parental Values' standpoint is to use these commonly known principles for the purpose of their own software development by their employees. The software would be meant for mobile devices held by minors and the software would be monitored by parent(s) and/or guardians. The software would be able to notify parent(s)/guardian(s) in the event of an active shooter in a proximity close enough for the mobile devices' onboard microphones to detect a gunshot's noise. This document is meant for the employees of Parental Values to understand the commonly known principles as it applies to their intended implementation.

CONTENTS

LIST OF FIGURES

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
App	Application

1. INTRODUCTION

This document outlines fundamentals of trilateration with statistics, recommending features and capabilities, and proposes commonly known general approaches a small-business could use to develop a mobile device application, commonly referred to as an app. Such an app would be running in the background on mobile devices which downloaded the app to detect certain noise characteristics, such as decibel level. This information could be collected by the app's central processing server which would perform particularly desired tasks. Such tasks could include sending notifications to specified phone numbers, comparing the noise characteristics to a database of noises of interest, localizing the source of gunshots, and/or warning nearby individuals about the presence, location, and threat of active shooters. Gunshot data sent to the central processing server by the mobile devices could be recorded by the central processing server and could be made available for additional parties or purposes. Database information can be supplied for gunshot profiles against which user data can be compared and a shot location as well as an idea of weapon type is possible.

2. TRILATERATION

2.1. Fundamentals

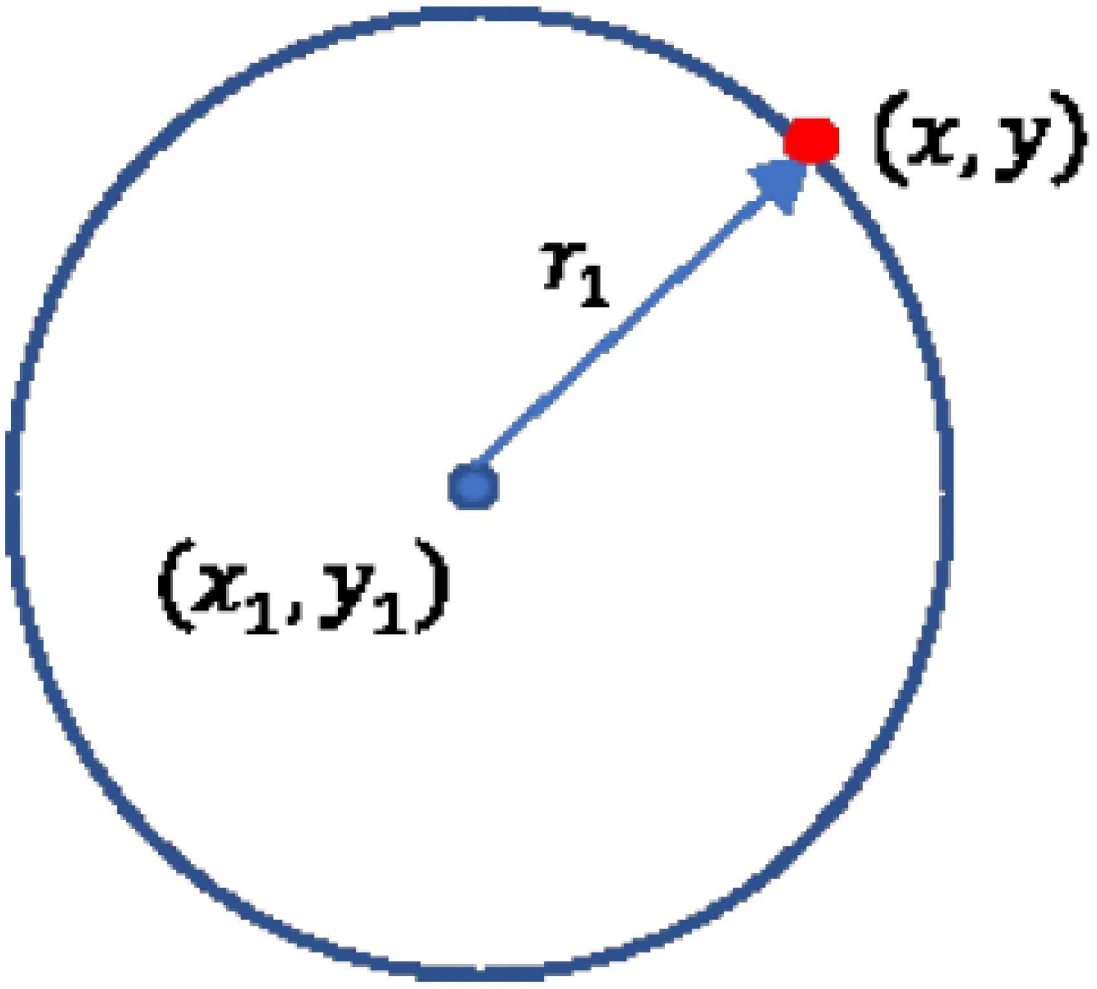

Figure 2-1: Illustration of Finding One Point Relative to Another

Trilateration can be accomplished in any dimension. For now, consider two dimensions, x and y. In the simplest case, trilateration determines the unknown position, (x, y), of a point by measuring the straight-line distances from that point to other points with known positions, (x_i, y_i). Here, the subscript, i, indicates the i^{th} point with known position. The technique is conceptually simple. Consider first, a single point with known position, (x_1, y_1). Assume the measured distance from this point to the point with unknown position is r_1. With this much information, we can only deduce that the point with unknown position is somewhere on the circumference of the circle with center (x_1, y_1) and radius r_1.

If the distance, r_2, from the point with unknown position to a second point with known position, (x_2, y_2)., is also measured, then we know that (x, y) is at one of two points where the circles defined by the two points intersect.

This is illustrated in the following diagram:

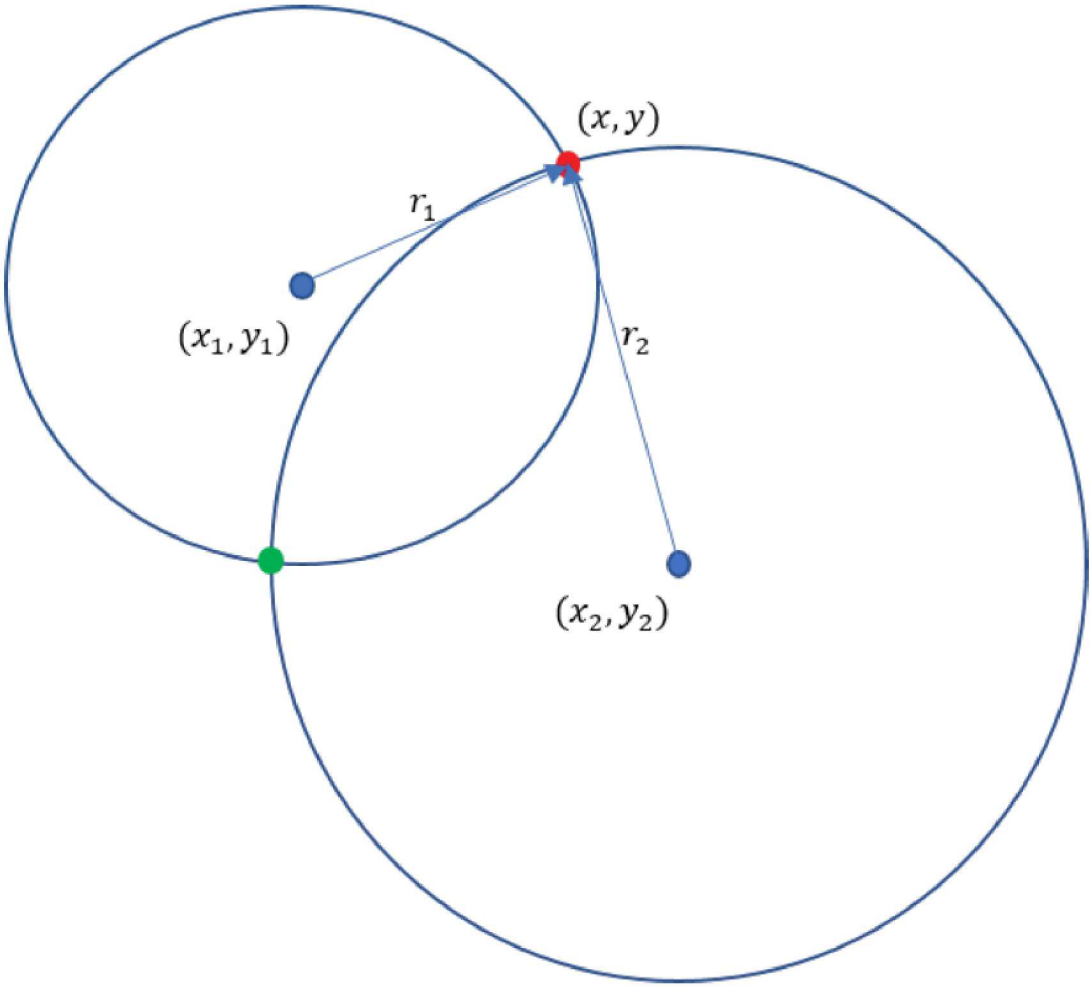

Figure 2-2: Illustration of Finding a Point Relative to Two Other Points

The location of the point indicated by the red dot is what we are trying to determine. The point indicated by the green dot is a "ghost", or ambiguous, point. To uniquely determine (x, y), we need a third point with known location, (x_3, y_3). The known positions of at least three points, along with the measured distances from the point with unknown position to the three points, allows (x, y) to be uniquely determined. The common point of intersection of the three circles, as illustrated in the following diagram, is at a single point (i.e., the point of interest).

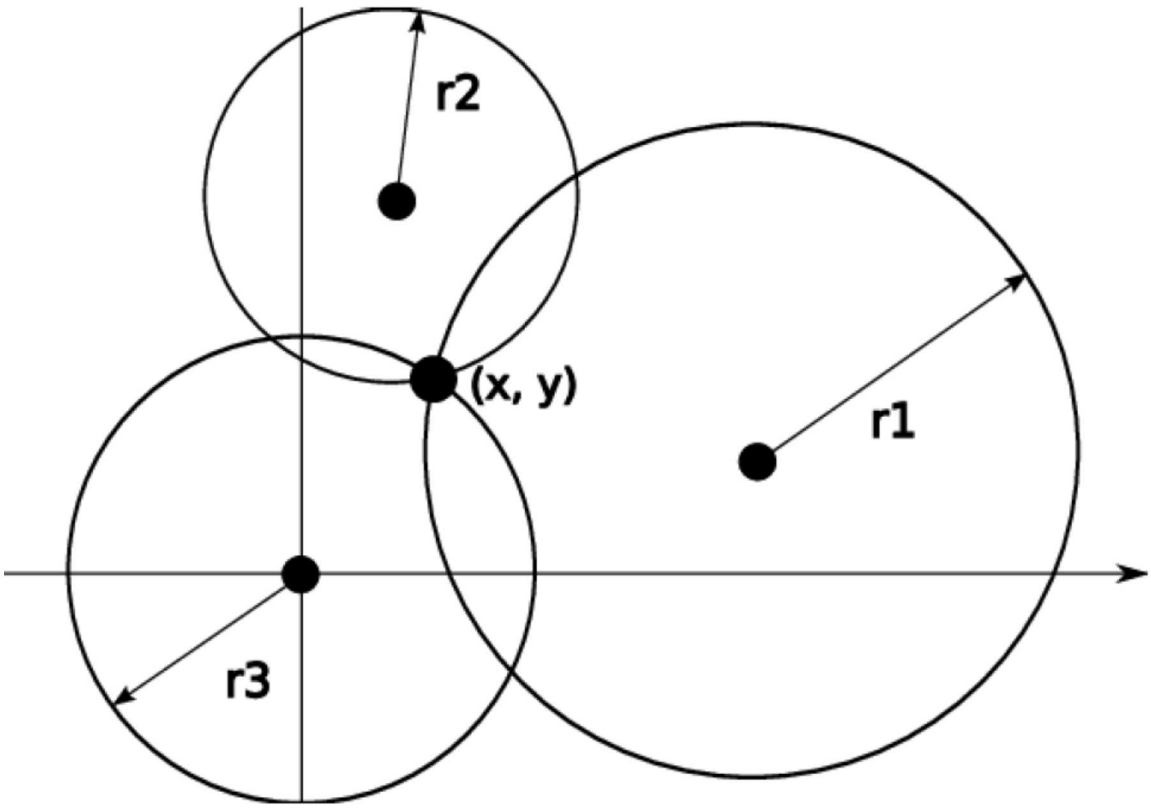

Figure 2-3: Illustration of Finding a Point Relative to Three Other Points

2.1.1. Measuring Radii

Assume a gunshot occurs at time t_0 and that the arrival time at each mobile device is t_i, where the subscript, i, indicates the i^{th} mobile device. If the shooter was kind enough to transmit t_0, then the distance from each mobile device to the shooter could be computed using:

$$r_i = c(t_i - t_0)$$

Figure 2-4: Equation for Finding Radii

.

Here, c is the speed of sound in air. Assume for now that c is known. In reality, c varies somewhat depending on temperature, pressure, and humidity. In the developments that follow, keep in mind that the inequality $t_i > t_0$ is true for all i. Unfortunately, it is unlikely the shooter will transmit the actual time of gunshots fired. Instead, you have to treat t_0 as an additional unknown. It turns out, to solve for t_0, all that is needed is another mobile device (four total). With at least four mobile devices it is possible to uniquely solve for x, y, and t_0. More users can provide improved accuracy in localization.

3. TRILATERATION ALGORITHMS

There are several possible algorithms that can be used to solve for (x, y, t_0). Following is a brief discussion on three different approaches.

3.1. Direct Approach

The direct approach solves the equations defined by the intersection of the circles with parameters x_i, y_i, and r_i. The equation for each circle is:

$$(x - x_i)^2 + (y - y_i)^2 = (c(t_i - t_0))^2$$

Figure 3-1: Equation of a Circle

This equation can be rearranged in terms of the constant, c, as

$$\frac{\sqrt{(x - x_i)^2 + (y - y_i)^2}}{(t_i - t_0)} = c$$

Figure 3-2: Equation of a Circle Algebraically Rearranged to Solve for C

If we assume c is the same along the paths from the gun to each mobile device, then the equations for each phone can be equated and solved for. Since there are three unknowns (x, y, t_0), we need at least three independent equations (four mobile devices) for a unique solution.

$$\frac{\sqrt{(x - x_1)^2 + (y - y_1)^2}}{(t_1 - t_0)} = \frac{\sqrt{(x - x_2)^2 + (y - y_2)^2}}{(t_2 - t_0)}$$

$$\frac{\sqrt{(x - x_1)^2 + (y - y_1)^2}}{(t_1 - t_0)} = \frac{\sqrt{(x - x_3)^2 + (y - y_3)^2}}{(t_3 - t_0)}$$

$$\frac{\sqrt{(x - x_1)^2 + (y - y_1)^2}}{(t_1 - t_0)} = \frac{\sqrt{(x - x_4)^2 + (y - y_4)^2}}{(t_4 - t_0)}$$

Figure 3-3: Three Equations to Solve For Three Unknowns

The problem with the direct approach is that both timing measurements and the GPS derived positions of mobile devices are often in error and, as a result, the circles seldom all intersect at a single, common point. Often, the direct approach fails to yield a solution. The following diagram illustrates this problem graphically. With three mobile devices and erroneous timing measurements there is not a common point of intersection. Instead, there are six ambiguous solutions, none of which are correct. The actual location is somewhere within the shaded region bounded by the arcs between points a, b, and c. However, this may be all that is needed for the first several iterations to achieve the desired goal of finding the vicinity of the shot.

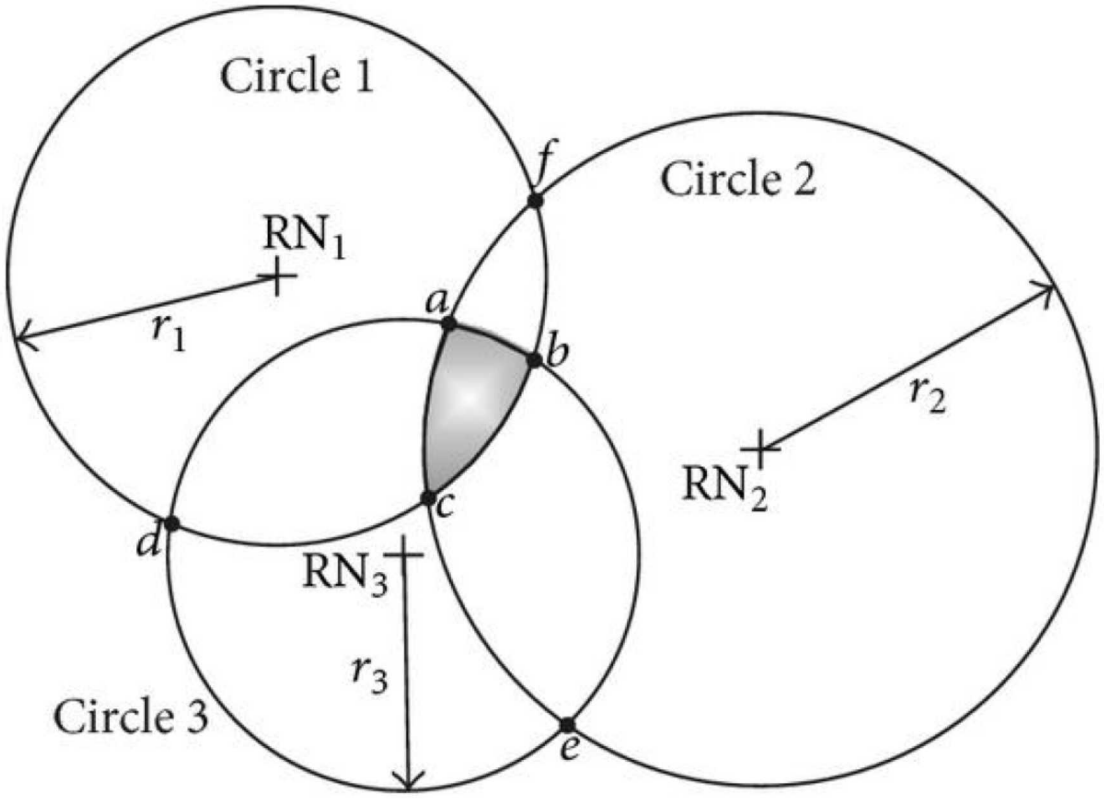

Figure 3-4: Illustration of Three Circles Intersecting Over a Region Rather Than a Single Point

3.2. Graphical Approach

A graphical approach, similar to the Hough Transform used in image processing, can yield a robust solution. The approach uses a discrete, two-dimensional grid. Given x_1, y_1, and r_1, a corresponding circle is drawn on the grid. A grid cell value is incremented by one if the circumference of the circle intersects the grid cell, or if a point on the circumference is within a specified distance to the grid cell. This is repeated for all circles. Then, the location of the gunshot is determined based on the region in the grid that contains the largest sum of grid cell values. The size of the region is determined according to the standard deviation of time-measurement errors.

3.3. Optimization Approach

Using this approach, a performance function is chosen and optimized. Begin with the equation of a circle given by:

$$(x - x_i)^2 + (y - y_i)^2 = (c(t_i - t_0))^2$$

Figure 3-5: Recall the Equation of a Circle

.

A distance error can easily be defined in one of two ways.

$$e_i = (x - x_i)^2 + (y - y_i)^2 - c^2(t_i - t_0)^2$$

Figure 3-6: Distance Error Definition 1

$$e_i = \sqrt{(x - x_i)^2 + (y - y_i)^2} - c(t_i - t_0)$$

Figure 3-7: Distance Error Definition 2

The first error equation is actually in terms of squared distances while the second error equation is in terms of distances. A least-squares performance function is defined as:

$$E = \sum_{i=1}^{N} e_i^2$$

Figure 3-8: Least- Squares Performance Function

An optimization procedure, such as steepest descent or fixed-point iteration, can then be used to determine the values of the unknown parameters, (x, y, t_0), that minimize E. The problem with least squares, however, is that it only takes a single outlier in a timing measurement (e.g., due to an echo or reflection) to throw off the solution. A more robust performance function is given by:

$$P = \frac{1}{N} \sum_{i=1}^{N} \exp\left(-\frac{e_i^2}{2\sigma^2}\right)$$

Figure 3-9: Robust Performance Function

With this function, smaller errors are weighted much more heavily than larger errors. Outliers minimally
contribute to the value of P. It is relatively easy to see that if all the e_i are zero, then P=1. However, if all the errors are large, then P tends towards zero. The range of P is from zero to one. The goal is to find the values of the unknown parameters that maximize P. The parameter, σ, in the

performance function is a weighting factor that is chosen based on the standard deviation of timing-measurement errors. As before, an optimization procedure, such as steepest descent or fixed point iteration, can be used to determine the values of the unknown parameters that maximize P. Although somewhat more complex than least squares, optimization of P generally yields much more robust results.

3.4. Trilateration Example

The following example illustrates trilateration using both E and P. For this example, assume the true location of the shooter is at (x, y) = (0,0) and that a shot is fired at t_0= 0 seconds. Then, the actual distance between a shooter and another point is:

$$d_i = \sqrt{x_i^2 + y_i^2}$$

Figure 3-10: Distance Between Two Points

The shortest distance between two points is the square root of the sum of the squares. This we know from Pythagoras' Theorem to find the hypotenuse. The greater the number, the farther the points are from each other and the smaller number, the closer they are.

Assume there are N = 15 app users within the vicinity of the shooter. The positions of the app users are assumed to be uniformly distributed as x_i~ U (-1000ft, 1000 ft) and y_i ~ U (-1000 ft, 1000 ft). Also assume that c = 1000 ft/sec. The measured time of arrival of the gunshot at each mobile device is given by:

$$t_i = t_0 + \frac{d_i}{c} + n_i$$

Figure 3-11: Time of Arrival

where n_i is normally distributed measurement noise, n_i ~ N (0, σ^2_e). The following MATLAB code generates the positions of mobile devices and the corresponding measurement times for a gunshot fired at t_0 = 0 seconds.

3.4.1. Simple MATLAB Example Code

```
% Experiment: Trilateration of Gunshots
clear;
close all;
clc;
% Generate positions of mobile devices
N = 15;
X = 2000*rand(1, N) - 1000;
Y = 2000*rand(1, N) - 1000;
% Specify speed of sound
c = 1000;
% Compute true distances from shooter
D = sqrt(X.^2 + Y.^2);
% Compute measured arrival times
se = 0.01; % se is the specified standard
deviation of timing errors
t = D./c + se*randn(1, N);
% Generate initial guess for location of the
shooter and t0
x = mean(X);
y = mean(Y);
t0 = min(t);
% Plot positions of mobile devices, shooter,
and initial guess of shooter
% location
plot(X, Y, '.', 'MarkerSize', 10);
grid on;
hold on;
xlim([-1000, 1000]);
ylim([-1000, 1000]);
plot(0, 0, '.', 'MarkerSize', 20);
plot(x, y, '.', 'MarkerSize', 20);
title("Locations of Assets and Shooter");
xlabel("X Position (ft)");
ylabel("Y Position (ft)");
```

```
legend("Assets", "Shooter", "Initial
Guess");
```

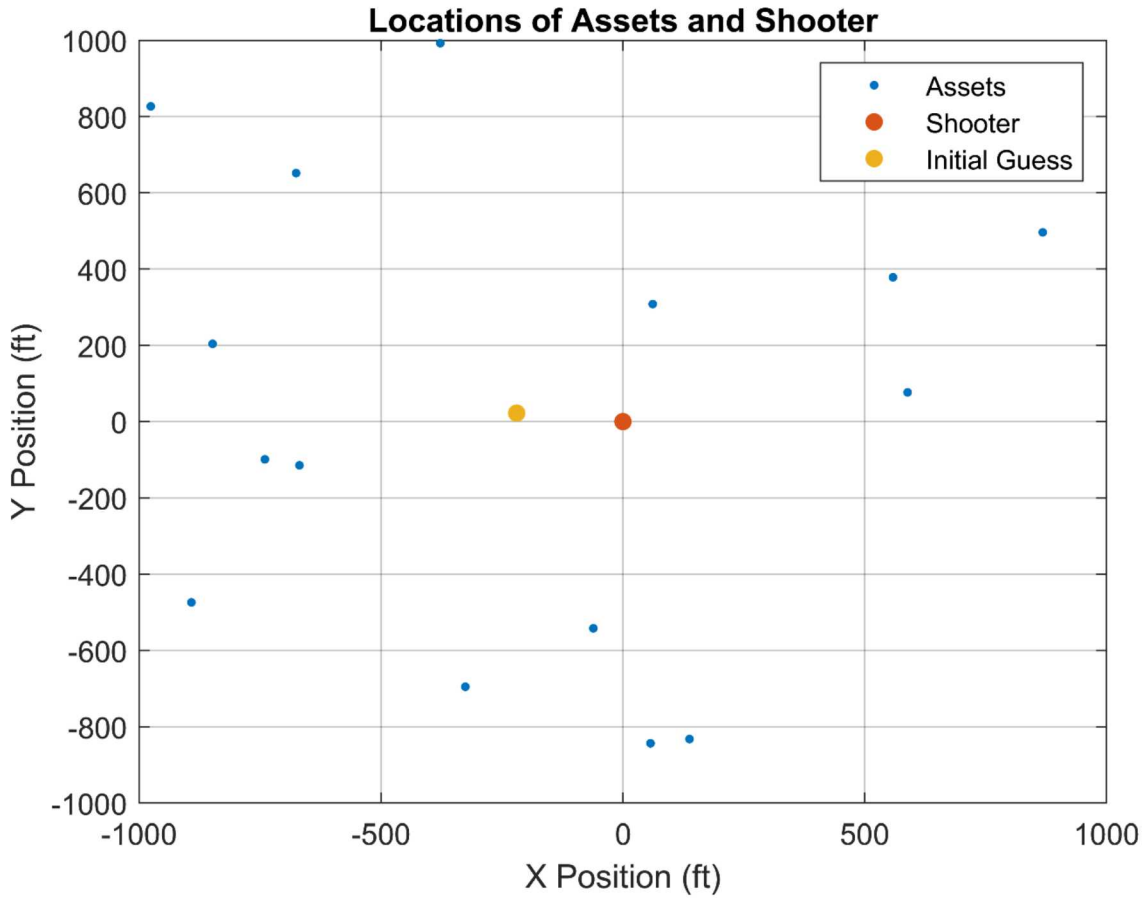

Figure 3-12: MATLAB Graph of Simple Example

It is important to remember that this plot is but one random case. The next step is to perform this case over many iterations to find the statistics for each case then plot them to compare the performance of the estimation method.

3.5. Metrics

As mentioned in the "Measuring Radii" section, there is a minimum of four devices with the app downloaded and running (users) needed to solve for a unique point location. What if we only have one or two or three users? What happens if we have 10 users? The example code in this section can answer those question by either varying the parameter and running the simulation or by redefining the variables as a range and performing the simulation for each number in the range. The number of sensors is herein defined as a range from 5 sensors to 100 sensors and each number of sensors has 1000 random locations in a 1000ft radius of the defined shooter position. The more times a scenario is evaluated, the more one can be confident in understanding the likelihood of the way that scenario

plays out because truly random fluctuations have an average of zero; therefore, to remove them one should average a large number of measurements.

The following explores detection and localization performance assessments using modeled data. Detection performance metrics contain probability of detection, P_D, and probability of false alarm, P_{FA}. Localization performance metrics include bias, B, and uncertainty, U. The assessments provide useful performance information in terms of key variables including the number of app users and their spatial distribution, the number of shots fired, how gunshot timing errors translate to localization errors, how outliers in timing measurements translate to localization errors, and how errors in positions of mobile devices affect localization accuracy. Localization involves detecting and estimating the time of gunshots heard at each mobile device and broadcasting data collected at each phone (e.g., GPS location, phone orientation, pressure, temperature, humidity depending on the onboard components, and which components the app has permissions to use) along with the estimated time of detected gunshots to other app users located within the proximity of the shooter. With this data, trilateration is used to accurately estimate the source of shots fired. Multiple gunshots fired from the same location and/or multiple app users can provide improved accuracy and confidence in gunshot location estimates.

In order to begin the modeling of data, assumptions and boundary conditions must be made. Assumptions include all data packets sent from the mobile devices to the server do not require additional retransmission protocol, mobile devices have minimum operational and hardware requirements to satisfy the detection of noise characteristics and wireless transmission of all pertinent information, there are no barriers to acoustic propagation through the air, and the space is predefined. We are also assuming probabilities of detection and probabilities of false alarm. These metrics will vary according to each situation because of the randomness of each unique situation and the incredible variety in mobile devices in service, the microphone quality, the connection of each phone to the GPS satellites and mobile networks, and the gun system itself (which may include silencers) among many other unknowns. This doesn't mean we can't model situations and probabilities, it simply means we must do many of them such that we minimize uncertainty and develop robust code(s). Once an established code is developed with these assumptions, it can be improved upon to include greater robustness to handle these realistic and complex environmental conditions.

The assumption that all mobile devices have the necessary requirements is not entirely valid in real world scenarios that Parental Values, LLC is attempting to address; many parents specifically provide their children with extremely simple devices that can only transmit to certain places and only certain information. The app's download page should provide the software and hardware requirements a device would need to run the app successfully.
Additional considerations must be neglected to perform analysis. The scenarios of a sniper or longer-range shooter are not considered due to the fact that the weapon is fired from a considerable distance. Although there is optimization described, this doesn't eliminate barriers like acoustic attenuators, such as thick walls or walls made of absorbing materials, ricochets of bullets, or attached silencers. The variability of component quality in mobile devices and the associated individual errors are not discussed.

One may want to use a sum of squares technique to find the dispersion from a mean in a simulated data set called regression analysis. It is used to determine what mathematical function best fits a data set. The greater the number, the more data points lie farther from the mean and thus there is large

variability in the data set. The smaller the number, the more points lie close to the mean and there is lower variability in the data set (usually the preferred state).

3.6. Fixed Point Iteration

There are many ways to solve for where radii may intersect. However, we will explore how to use fixed-point iteration when getting each sensor to activate based on the time of arrival of the sound.

3.6.1. Simple Gunshot Position-Time Estimator Code

```
%% Gunshot Position-Time estimator

% Clean up
clear all;
close all;
clc;

% Initialize variables
c = 1000; % speed of sound in air in feet/second
Nmin = 5; % minimum number of sensors
Nmax = 100; % maximum number of sensors
Rmax = 1000; % maximum distance (feet) of sensor
from gunshot position

% Specify experiment parameters for bias &
uncertainty analysis
M = 100000; % number of experiments

% Conduct experiment
sigmaT = 10/c; % timing measurement noise standard
deviation (sec)
sigmaXY = 10; % sensor measurement noise standard
deviation (feet)
maxCnt = 10000; % maximum number of iterations
allowed for FPI to converge
eps = 0.000001; % accuracy requirement for FPI
solution
divergence = zeros(1, Nmax - Nmin + 1);
Bx = zeros(1, Nmax - Nmin + 1);
Ux = zeros(1, Nmax - Nmin + 1);
By = zeros(1, Nmax - Nmin + 1);
```

```
Uy = zeros(1, Nmax - Nmin + 1);
Bt = zeros(1, Nmax - Nmin + 1);
Ut = zeros(1, Nmax - Nmin + 1);
sVal = zeros(1, Nmax - Nmin + 1);  % for plotting
x-y (number of sensors)
n = 0;
for N = Nmin:Nmax
    N
    n = n + 1; % this is an index into a sensor
array
    sVal(n) = N;
    xErr = zeros(1, M);
    yErr = zeros(1, M);
    tErr = zeros(1, M);
    divCnt = 0;
    k = 0;
    for j = 1:M

        % Specify random positions of each sensor
(in polar coordinates)
        R = Rmax*rand(1, N);
        Theta = 2*pi*rand(1, N);

        % Convert from polar to rectangular
coordinates
        X = R.*cos(Theta) + sigmaXY*randn(1, N);
        Y = R.*sin(Theta) + sigmaXY*randn(1, N);

        % Compute time of arrival measurements
        T = R/c + sigmaT*randn(1, N);

        % Use FPI to solve for x, y, and t.
        % Use initial guesses of 0, 0, and 0.
        oldx = 0;
        oldy = 0;
        oldt = 0;
        ex = 1000;
        ey = 1000;
```

```
        et = 1000;
        cnt = 0; % divergence counter
        while (ex > eps || ey > eps || et > eps) &&
cnt < maxCnt
            cnt = cnt + 1;
            sxnum = 0;
            sxden = 0;
            synum = 0;
            syden = 0;
            stnum = 0;
            stden = 0;
            for i = 1:N
                dx = oldx - X(i);
                dy = oldy - Y(i);
                dt = oldt - T(i);
                err = dx^2 + dy^2 - (c*dt)^2;
                sxnum = sxnum + err*dx;
                sxden = sxden + err + 2*dx^2;
                synum = synum + err*dy;
                syden = syden + err + 2*dy^2;
                stnum = stnum + err*dt;
                stden = stden + err - 2*(c*dt)^2;
            end % close for loop on i

            % compute new parameter values
            if sxden > 0
                newx = oldx - sxnum/sxden;
            else
                newx = oldx;
            end
            if syden > 0
                newy = oldy - synum/syden;
            else
                newy = oldy;
            end
            if stden > 0
                newt = oldt - stnum/stden;
            else
```

```
                newt = oldt;
            end

            % compute absolute errors
            ex = abs(newx - oldx);
            ey = abs(newy - oldy);
            et = c*abs(newt - oldt);

            % Update old values with new values
            oldx = newx;
            oldy = newy;
            oldt = newt;
        end % close while loop

        if cnt >= maxCnt
            divCnt = divCnt + 1;
        else
            k = k + 1;
            % err = estimate - true value, (but
true value was 0)
            xErr(k) = newx;
            yErr(k) = newy;
            tErr(k) = newt;
        end % close if-else loop

    end % close for loop on j

    xErr = xErr(1:k);
    yErr = yErr(1:k);
    tErr = tErr(1:k);

    % Compute Bias and Uncertainty
    divergence(n) = divCnt;
    Bx(n) = mean(xErr);
    Ux(n) = std(xErr);
    By(n) = mean(yErr);
    Uy(n) = std(yErr);
    Bt(n) = mean(tErr);
    Ut(n) = std(tErr);
```

```
end % close for loop on N

%% plot Estimation Bias and Uncertainty
% Plot Bx and Ux first
figure;
plot(sVal, Bx);
hold on;
grid on;
plot(sVal, Ux);
title("x: Bias and Uncertainty");
xlabel("Number of Sensors");
ylabel("B and U");
legend("Bx", "Ux");

% Plot By and Uy second
figure;
plot(sVal, By);
hold on;
grid on;
plot(sVal, Uy);
title("y: Bias and Uncertainty");
xlabel("Number of Sensors");
ylabel("B and U");
legend("By", "Uy");

% Plot Bt and Ut third
figure;
plot(sVal, Bt);
hold on;
grid on;
plot(sVal, Ut);
title("t: Bias and Uncertainty");
xlabel("Number of Sensors");
ylabel("B and U");
legend("Bt", "Ut");

% Plot divergence percentage last
figure;
```

```
plot(sVal, divergence/M);
hold on;
grid on;
title("Divergence Percentage");
xlabel("Number of Sensors");
ylabel("Divergence");
```

3.6.2. Explanation and Plots

The variable, ***sigmaXY***, is the variable by which one can set sensor measurement noise standard deviation in feet. The example was done with two numbers, 10 feet (as seen in §3.6.1), and 100 feet. Some of the better sensors available today in many mobile phone systems have low standard deviations but there are still many that aren't that good. In this way, we can 'bound' the system with sensors that perform reasonably well (10 ft standard deviation) and those which perform poorly (100 ft standard deviation). It can start as a good case scenario and a bad case scenario and one can observe a uniformly distributed array of sensors and for each number of sensors (5-100 number of sensors) 10,000 different random setups are generated.

This may not be the best set-up as in school systems, there are clusters of students in classrooms or auditoriums or bathrooms or football fields or bleachers, etc. The next step after establishing a good system with a random set-up is to try a clustered set-up and define classroom like areas within the overall area where sensors can be generated. Then one would have a certain number of sensors in clustered areas within the overall area and then repeat the simulation. Additional improvements of the same type can come later and the accuracy of how the simulation and code(s) replicate real world examples will improve.

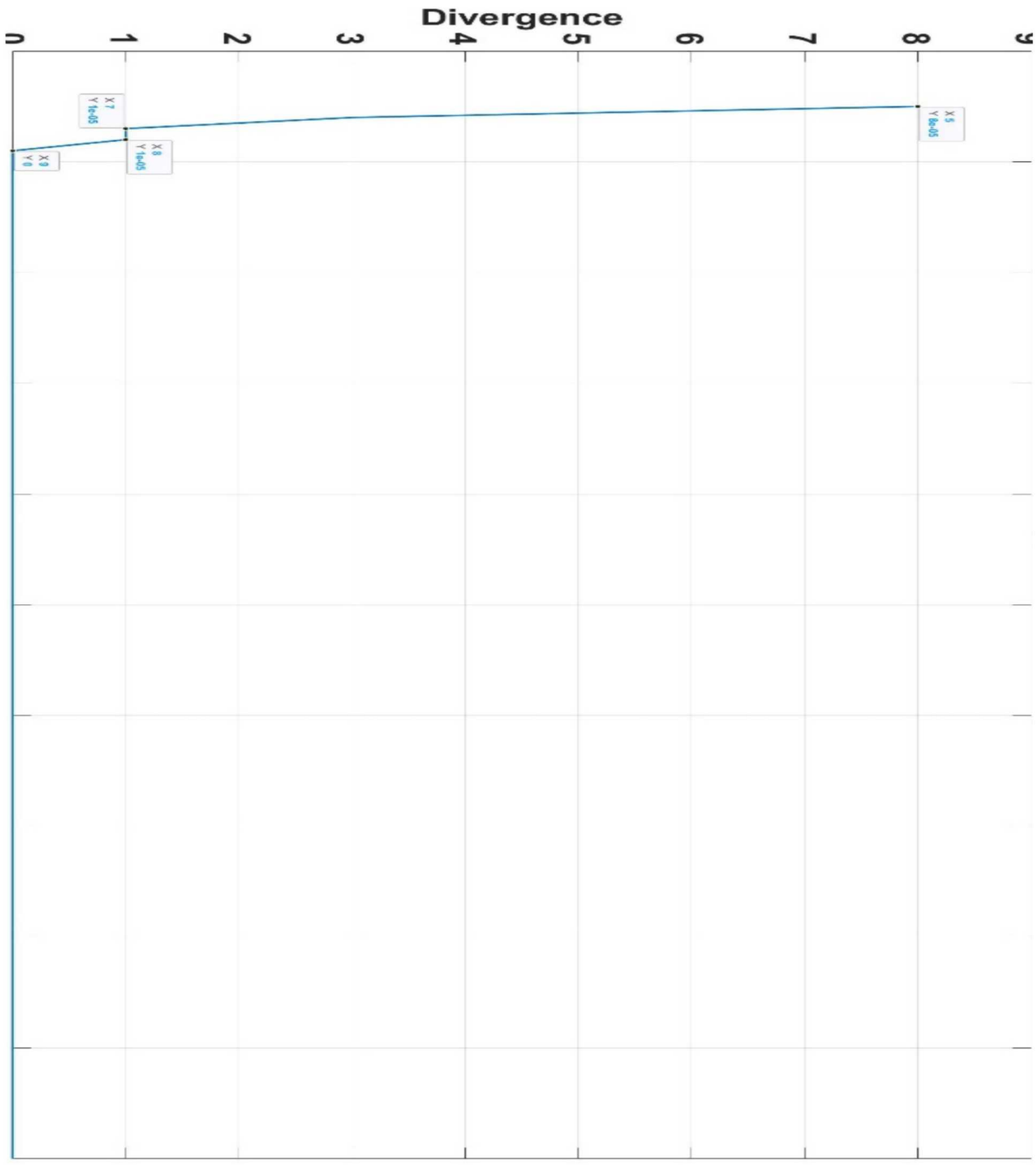

Figure 3-13: Divergence of the 'Worst-Case' Scenario of 100 ft Standard Deviation

When a numerical method converges, we get a solution to the problem. Conversely, when a numerical solution diverges, we do not get a solution. For the worst-case scenario of 100 ft standard deviation, the divergence is pretty small for our 10,000 iteration per number of sensors. After we have 10 sensors, the divergence never happens meaning all our iterations yielded solutions. Even in our worst-case scenario, we only need 10 users to provide a solution consistently.

Once we get 16 users with poor sensors, our uncertainty for the estimated time of arrival correlation to the actual time of the shot is zero.

By the time we get 40 users with poor sensors, we can pin-point the shot within 30 feet.

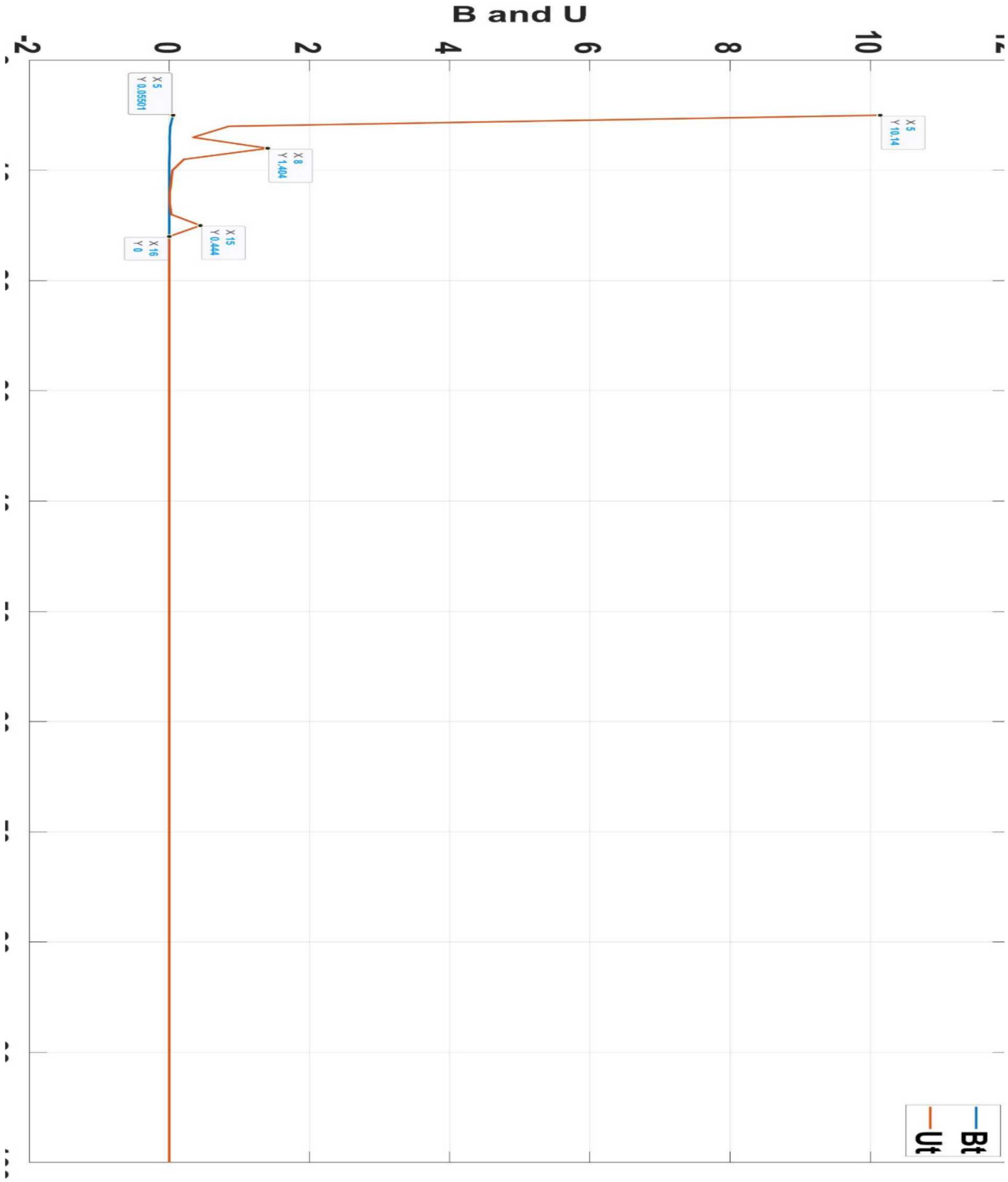

Figure 3-14: Time Estimation Bias and Uncertainty of the 'Worst-Case' Scenario of 100 ft Standard Deviation

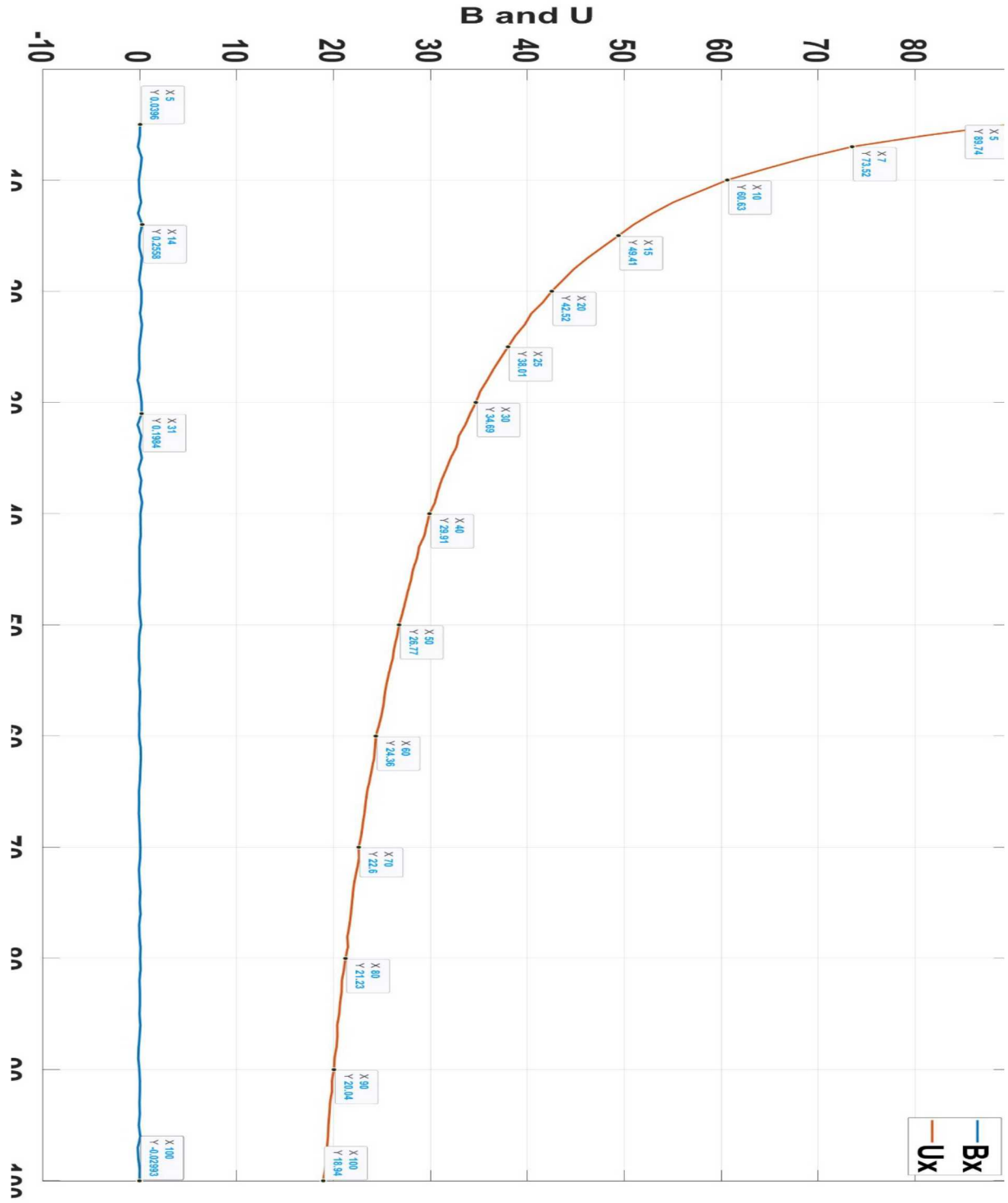

Figure 3-15: The X Position of the Shot Bias and Uncertainty in the 'Worst-Case' Scenario of 100 ft Standard Deviation

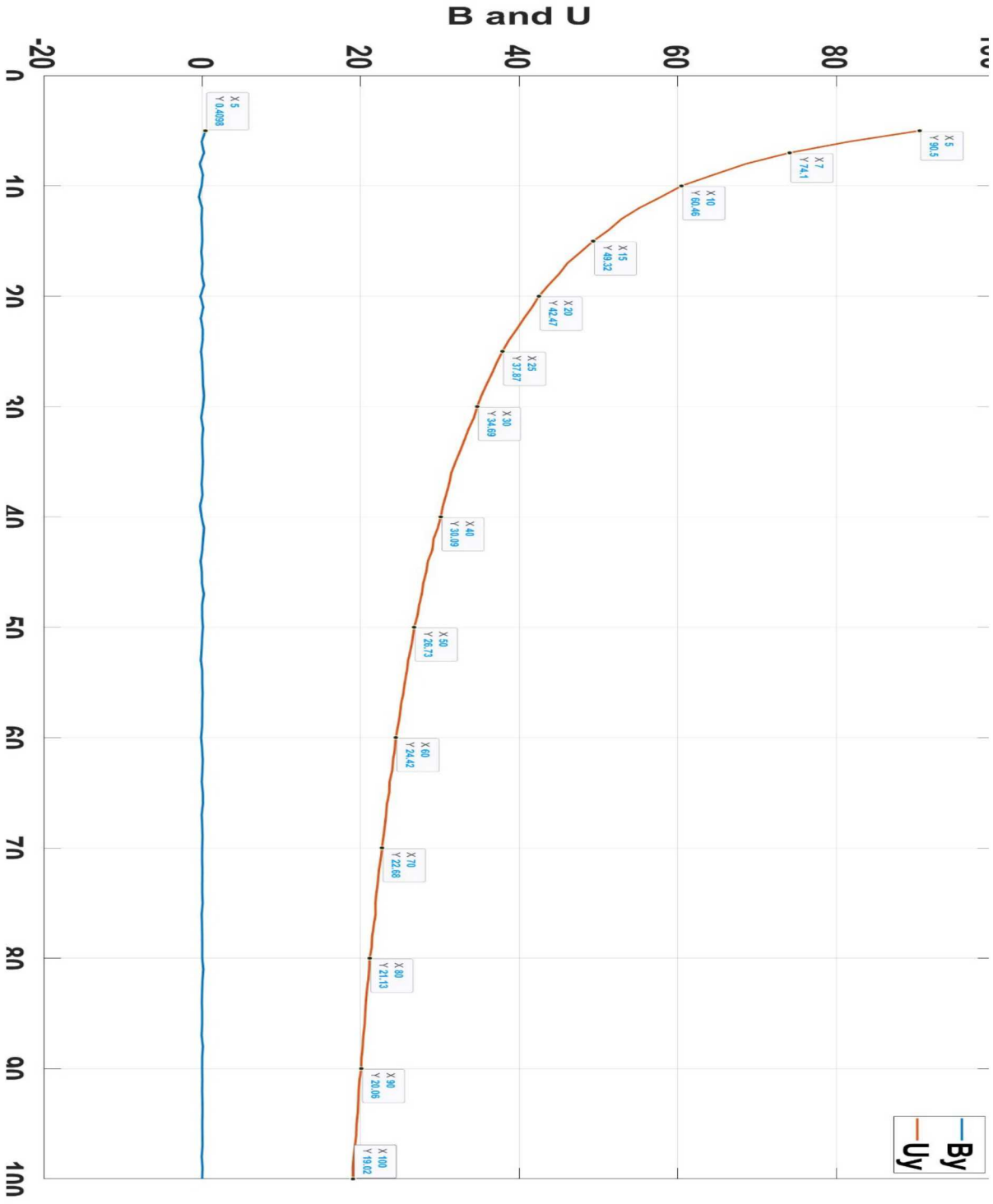

Figure 3-16: The Y Position of the Shot Bias and Uncertainty in the 'Worst-Case' Scenario of 100 ft Standard Deviation

Now for good sensors, we always converge, only need seven users to get the time of arrival and shot time correct, and with seven users, we can estimate the location of the shot within 12 feet.

This is but one random solution set. Each solution set would come up with slightly different numbers and configurations as they are innumerable. However, the trends should follow a similar pattern. Once an approach to the problem has been established, this is how one would test the code's performance. These performance metrics would then inform the next improvement iteration until the performance was acceptable to release.

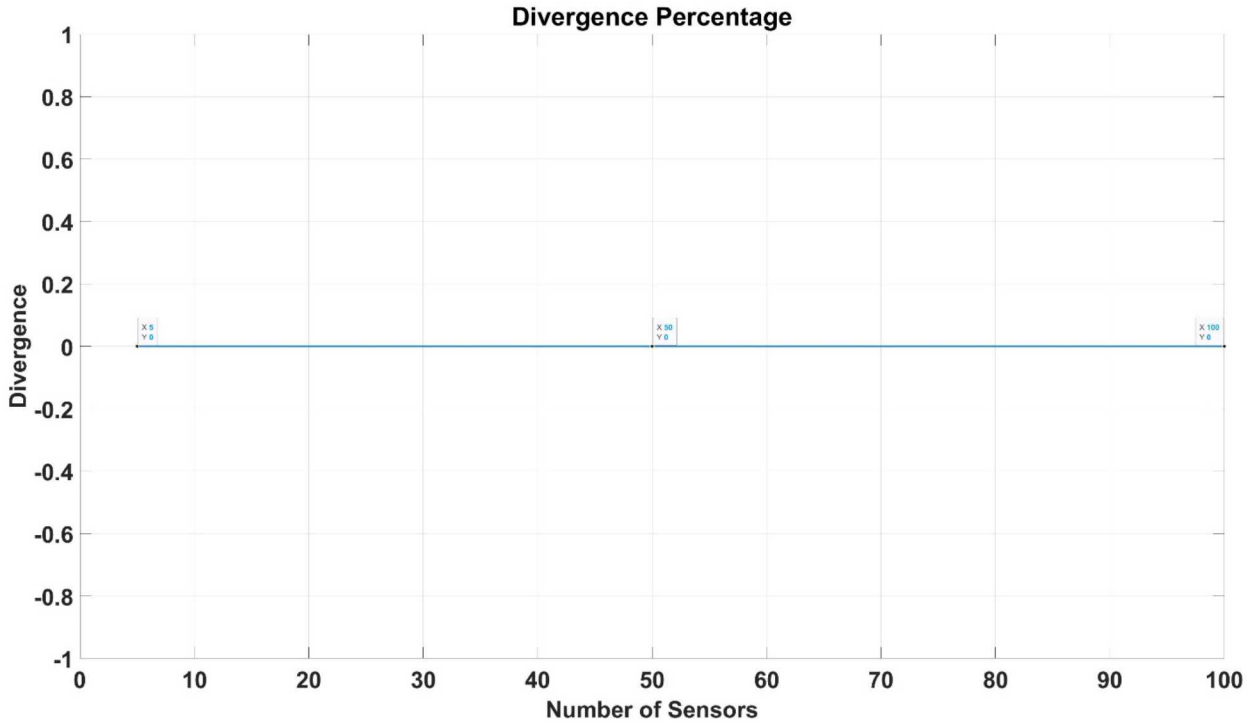

Figure 3-17: Divergence of the 'Best-Case' Scenario of 10 ft Standard Deviation

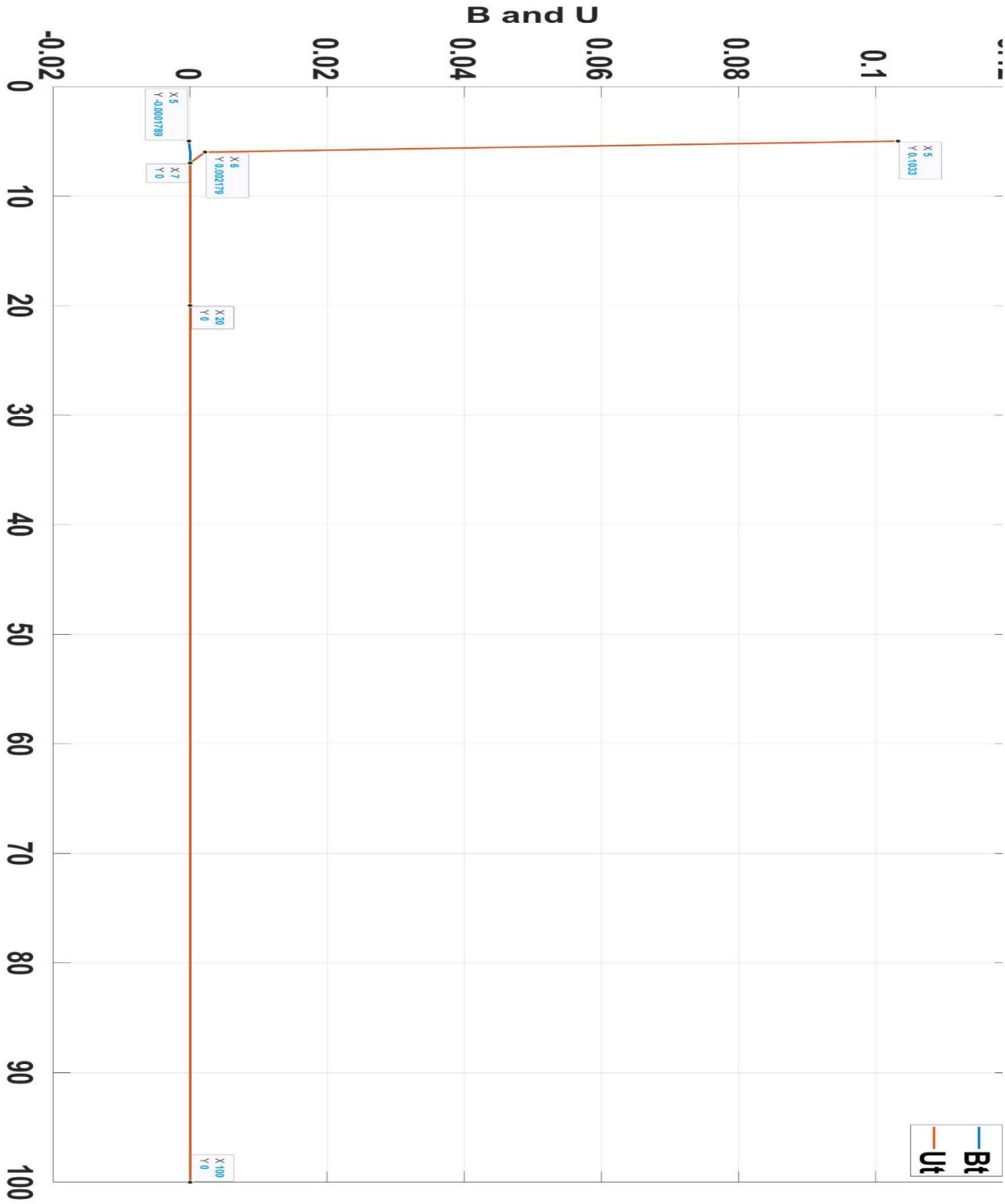

Figure 3-18: Time Estimation Bias and Uncertainty of the 'Best-Case' Scenario of 10 ft Standard Deviation

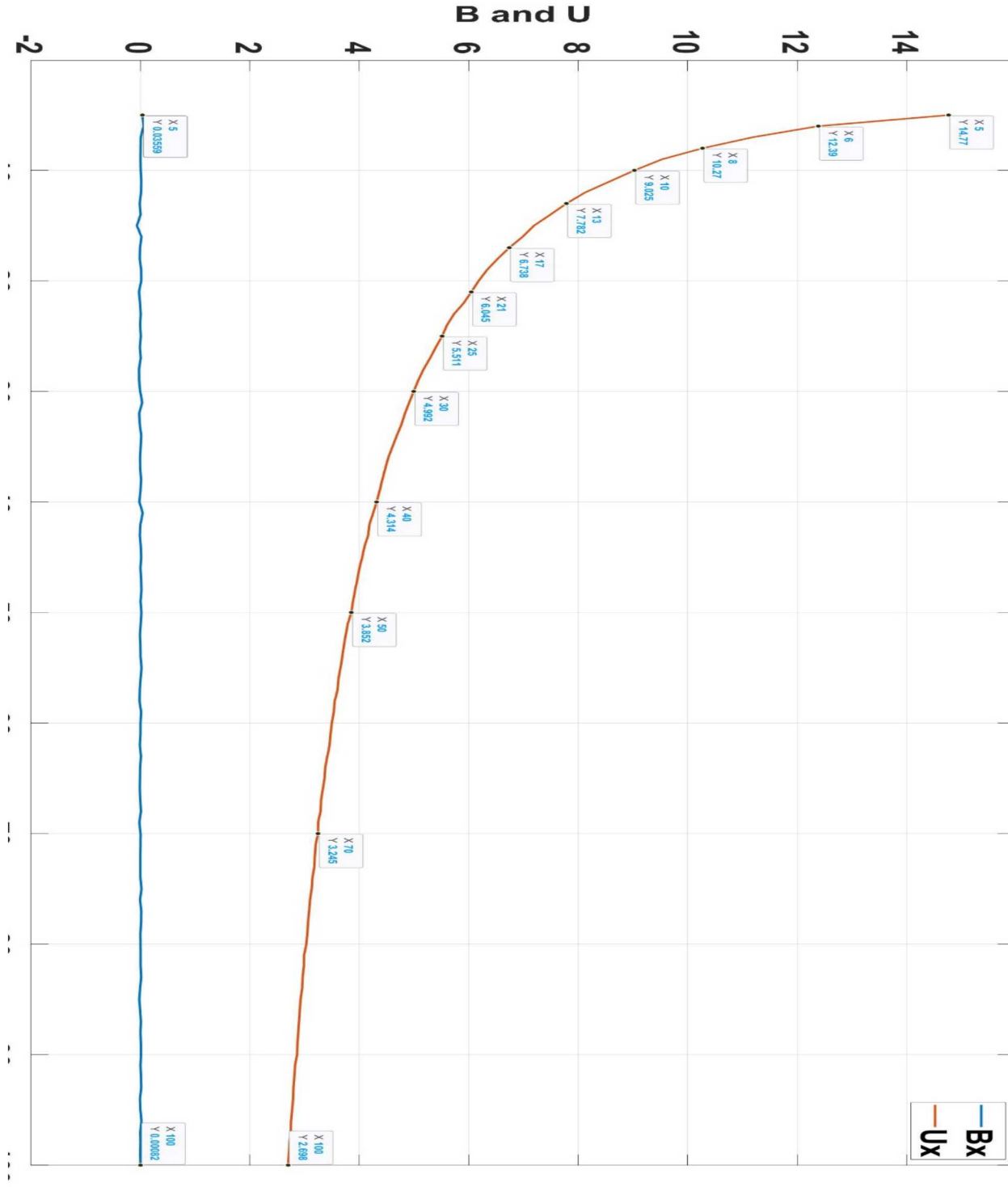

Figure 3-19: The X Position of the Shot Bias and Uncertainty in the 'Best-Case' Scenario of 10 ft Standard Deviation

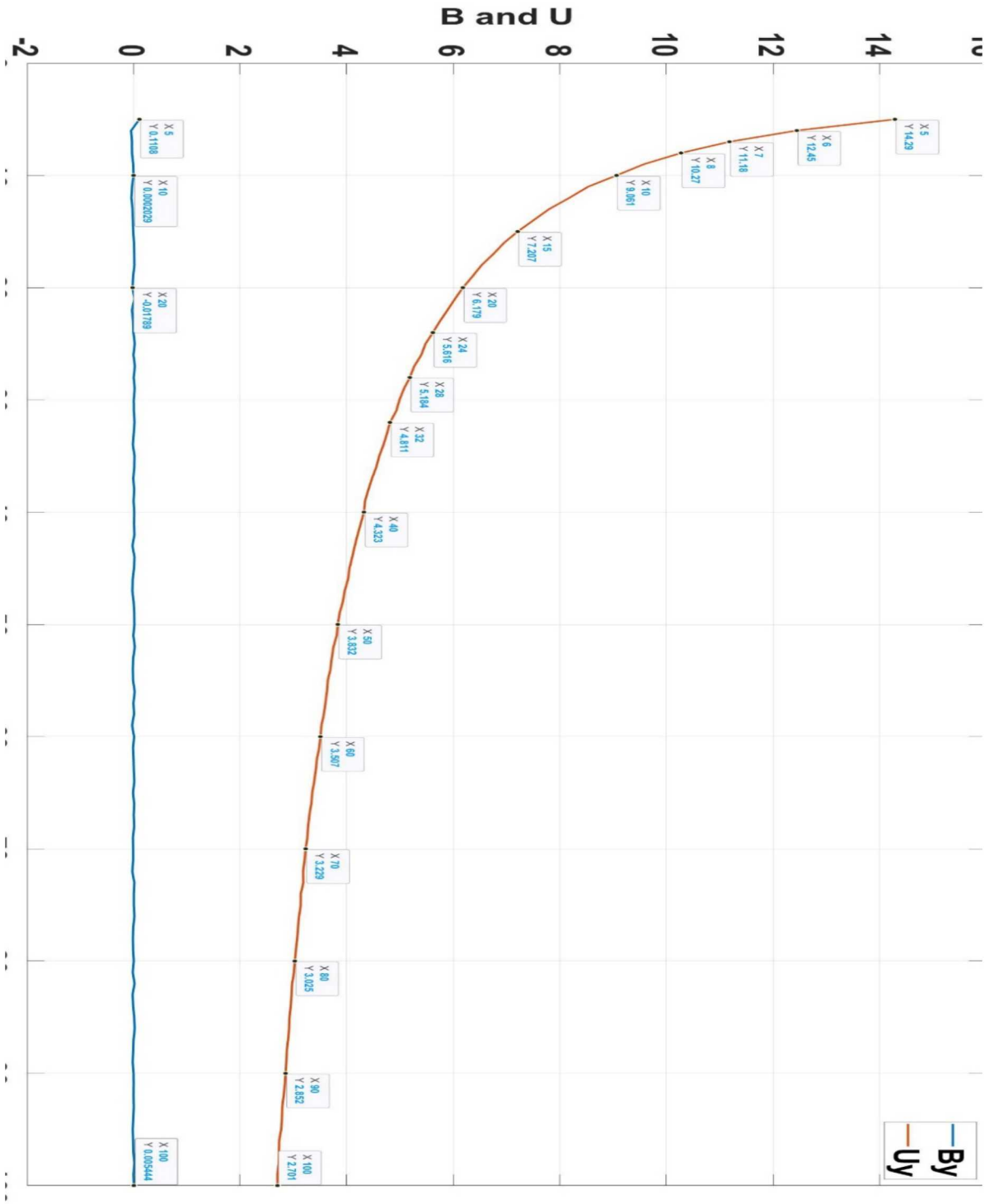

Figure 3-20: The Y Position of the Shot Bias and Uncertainty in the 'Best-Case' Scenario of 10 ft Standard Deviation

4. CONCLUSION

This is a problem that can be solved in many ways using several different approaches. There may even be the case that many ways are used as the iterations progress and one may help refine another. Once an approach has been selected and developed, the next step is to test it and use the results of the test to improve. Then repeat the improvement and testing process until it has been refined enough to take into a real-world simulation. Once the real-world simulations produce results worthy of release, the product is ready for user beta testing. Several more iterations will be likely before the product is ready for high-fidelity consideration.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Juan Martinez	01983	jmart30@sandia.gov
Technical Library	01977	sanddocs@sandia.gov

Email—External (encrypt for OUO)

Name	Company Email Address	Company Name
Jason Boxum	management@parentalvalues.com	Parental Values, LLC

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.