DEVCOM DAC-TR-2022-078
August 2022

Multipurpose Universal Simplified TLE Calculator (MUSTC)

by Matthew Banta

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so specified by other official documentation.

WARNING

Information and data contained in this document are based on the input available at the time of preparation.

TRADE NAMES

The use of trade names in this report does not constitute an official endorsement or approval of the use of such commercial hardware or software. The report may not be cited for purposes of advertisement.

DEVCOM DAC-TR-2022-078
August 2022

Multipurpose Universal Simplified TLE Calculator (MUSTC)

by Matthew Banta

DEVCOM Analysis Center

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2022	Technical Report	April 2018–June 2021

4. TITLE AND SUBTITLE
Multipurpose Universal Simplified TLE Calculator (MUSTC)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Matthew Banta

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Director
DEVCOM Analysis Center
6896 Mauchly Street
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM DAC-TR-2022-078

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The U.S. Army Combat Capabilities Development Command Analysis Center created an algorithm to estimate the target location error for position, navigation, and timing (PNT) sensors and systems. The algorithm can be used even if the exact algorithm that the system uses to find an object of interest is proprietary or unknown. The program is highly modular and expandable; therefore, it is relatively easy to add a wide array of different PNT sensors, systems, and targets. However, the only sensors that have been added at this time are Signals Intelligence systems using time difference of arrival, frequency difference of arrival, and/or angle of arrival as well as electro-optical/infrared (EO/IR) systems that may have laser range finders and photon counting detectors measuring the radiation from calibrated sources.

15. SUBJECT TERMS
target location error; TLE; position, navigation, and timing; PNT; Signals Intelligence; SIGINT; electro-optical/infrared; EO/IR; laser range finder; LRF

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Matthew Banta
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	112	19b. TELEPHONE NUMBER (include area code) (410) 278-6995

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

(U) Table of Contents

List of Figures

List of Tables

Acknowledgments

The author wishes to acknowledge the contributions of the following individuals from the U.S. Army Combat Capabilities Development Command (DEVCOM) Analysis Center (DAC) for their assistance with the creation of this report:

Brian Hairfield, DAC

William Harclerode, DAC

Elizabeth Jones, DAC

Brandi McGough, DAC

Scott Pridgeon, DAC

Gina Schafer, DAC

Jenny A Weathers, DEVCOM Army Research Laboratory

Executive Summary

The DOD has various sensors that the warfighter can use to find a location. Some sensors allow the warfighter to locate a potential threat. If the warfighter does not currently have access to a GPS, they may need to use sensors to determine their own location. Sensors can be used by themselves or linked together in a more complicated scenario to estimate the location of an object of interest. To ensure that the U.S. Army is equipping warfighters with sensors capable of performing their mission, it is critical that a model is created that can estimate the performance of these location sensors in any scenario.

The U.S. Army Combat Capabilities Development Command (DEVCOM) Analysis Center needed an algorithm to estimate the performance of various sensors and systems that are performing position, navigation, and timing (PNT) calculations. DEVCOM Analysis Center (DAC) developed the Multipurpose Universal Simplified TLE Calculator (MUSTC) model, which can be used to find the target location error (TLE) of a wide variety of sensors that can in turn be used to locate a wide variety of objects.

The MUSTC algorithm does not require the user to understand how the system uses the measurements that the sensors take in order to determine a location. All that is required to add a new sensor type to the MUSTC software is a model that estimates the raw values that the sensor(s) would measure as a function of the sensor and target parameters along with their locations.

For the algorithm to determine a TLE, the algorithm needs to know where all reference sensors and targets will be in the scenario, the variables that may affect the location measurement, and the uncertainties in these variables, as well as the location in space where the user would like to calculate a TLE value for the item of interest. The algorithm will then assume that the item of interest will nominally be located at the point where the user would like to estimate a TLE. Once the location is known, the software can use the measurement models to determine what the sensors will measure for the scenario. The software can then use these measurements, along with an optimization algorithm, to determine the TLE for the item of interest at the specified point in space.

The main advantage of the algorithm is that it can be expanded to determine how a variety of uncertainties in measurements from different sensor types can affect the total TLE, or the uncertainty of finding the location of an item of interest.

The main disadvantage of the algorithm is that the calculations can sometimes be time consuming due to repeated calls to a function that implements an optimization algorithm to calculate the TLE. There are many optimization algorithms that the program could

use, and some are faster than others. Even if the program uses a relatively fast optimization algorithm, the computation time can still add up if the optimization algorithm is called enough times. DAC has endeavored to mitigate this disadvantage by finding the fastest optimization algorithm available that still yields the correct answer, writing the program as a multi-threaded application so it can utilize the multiple cores that most modern computer processors have, and trying to find the best balance between accuracy of the final results and the number of times that the optimization algorithm must be called.

1. INTRODUCTION

One way the U.S. Army Combat Capabilities Development Command (DEVCOM) Analysis Center supports the warfighter is by determining the maximum impact of taxpayer dollars while shaping the U.S. Army of the future. Understanding where various targets and assets are on the Earth is a critical part of many missions. It is therefore important for the DEVCOM Analysis Center (DAC) to have the tools to evaluate both current and potential future position, navigation, and timing (PNT) sensors.

DAC needs a tool that can find the target location error (TLE) for a wide range of sensors in a wide range of scenarios. Typically, the warfighter will use GPS to determine the location of an asset. However, there may be times when GPS is not available. In the absence of GPS, there are various techniques the warfighter can use to find their location, the location of a target, or the location of a remote asset. Industry continues to propose new sensors and techniques to help with location. It is necessary to have a model that can find the TLE for almost any current or future PNT sensor, and the model should work even if the exact algorithm that the sensor uses to find the location of the target is proprietary or unknown. DAC developed the Multipurpose Universal Simplified TLE Calculator (MUSTC) model to fill this need.

The program that implements the MUSTC algorithm was designed to be as modular as possible, making it easy to add new sensor types. All that is required to add a new sensor type is enough knowledge of the sensor to be able to accurately model the raw measurements that sensor will be making (Appendix A describes how to add new sensor types to the MUSTC software). Even though the model can handle a wide array of sensor types, this report will focus on the first three types of sensors that were added to the algorithm: Signals Intelligence (SIGINT), electro-optical/infrared (EO/IR) with laser range finders (LRFs), and photon-counting detectors measuring the radiation from calibrated light sources.

SIGINT systems are designed to intercept RF radiation that is emitted by the adversary's radar, jammer, and/or radio communications system. SIGINT systems are often tasked with determining the location of the object that is emitting the RF radiation. There are multiple techniques that SIGINT systems use to find the location of an emitter.

1. **Angle of Arrival (AOA)**. There are types of antennas that can be used to measure the angle from which incoming radiation originated. If more than one sensor located at more than one location relative to the emitter can measure this

angle, then they can use these angles and the sensor geometry to estimate the transmitter location.

2. **Time Difference of Arrival (TDOA)**. This technique involves measuring the time at which a given signal reaches each SIGINT sensor in an area. Since the sensor does not know the location of the emitter, it cannot directly measure the time it takes for the signal to travel from the emitter to a SIGINT sensor. However, they can measure how much longer it takes for the signal to reach one sensor as compared to another sensor. By measuring this difference between multiple sensors and knowing the exact geometry of the sensors receiving this signal, the transmitter location can be determined.

3. **Frequency Difference of Arrival (FDOA)**. For this technique to work, the SIGINT systems must be installed on platforms that are moving (e.g., an aircraft). When aircraft are flying at different angles relative to the transmitter, there will be a difference in the frequency of the received radiation due to the Doppler effect. If the operator knows the location, speed, and direction of travel for all sensors, then they can compare the frequency of the incoming radiation for all sensors and use this information to determine the emitter location.

Any book on SIGINT systems will supply equations that can be used to determine the location of an emitter when using just TDOA, FDOA, or AOA. However, sometimes there are system geometries where these equations may no longer be valid. For example, if there are only two sensors there may not be enough information to use just TDOA or FDOA to find the emitter location. In this case, TDOA, FDOA, and AOA may need to be used at the same time to find the emitter location. If the emitter is moving, then FDOA calculations become much more complicated. There are also scenarios where the emitter location is known, but the location of one or more of the SIGINT sensors are not. In this way the warfighter can determine the location of an asset with a SIGINT sensor even if GPS is not available.

If we use the algorithm that a SIGINT system uses to find the target location when we create an algorithm that determines the system TLE then we would have to create a new TLE model for each algorithm, or set of algorithms, that a SIGINT system might use. In some cases, the exact algorithm that a system uses might be proprietary and therefore might not be available for use as part of a TLE estimation algorithm. However, because the MUSTC algorithm only requires the user to model the raw measurements that a SIGINT system is charged with making to determine the TLE of the system, the MUSTC algorithm can be used to estimate the performance of almost any SIGINT system in any scenario using any algorithm.

EO/IR systems with LRF are another system used to find object location. EO/IR sensors are typically imaging sensors mounted on some type of gimbal mount. By determining where an object is in an image and noting the angle at which the gimbal mount had to traverse to be able to image the target, EO/IR sensors can determine the azimuth and elevation angles between the target and the sensor. LRF systems have lasers that are used to illuminate the target. By measuring the time it takes for the laser radiation to leave the sensor, hit the target, and then return to the sensor again, LRF sensors can determine the range to a target.

EO/IR sensors with LRF are similar to SIGINT systems that use AOA to find the location of an RF emitter. The only difference is that when EO/IR sensors are equipped with LRF, they can also measure the distance to a target. Sensors that measure location by measuring the angle and/or distance to a target are often relatively simple to model. When the scenario is simple enough, complicated algorithms like the MUSTC are not needed to determine the performance of a system that measures the angle to a sensor.

However, if angle sensor measurements are combined with other sensor types, then the algorithm can become complicated enough that MUSTC would be necessary to model their performance. The inclusion of these types of sensors also has the advantage of verifying the accuracy of the MUSTC model by seeing if it can correctly find the TLE for a scenario that is simple enough that the expected TLE can be calculated by hand.

The final sensor type included in the initial set is photon counting detectors measuring radiation from light sources giving off a known amount of radiation. The main advantage of adding this sensor type is that it allowed for testing and demonstration features that could be needed to add additional sensor types. For example, the TLE as measured by a photon counting detector can depend on variables other than the location and/or velocity of the reference sensors/assets, and uncertainties in the direct measurements that the sensors are making. The uncertainty in finding a location based on the measurement from a radiation source emitting a known amount of energy can depend on the following: uncertainties in the amount of radiation the source is actually emitting, the extinction coefficient between the source and the sensor, and the transmittance of the optics that the sensor uses. Also, modeling the uncertainty with which a photon counting detector measures incoming radiation may require the use of a Poisson distribution instead of a normal distribution as required for other sensor types. Successfully modeling a photon counting detector measuring radiation from a calibrated source proves that the algorithm can handle different statistical distributions as well as variables affecting various aspects of the scenario.

As with SIGINT systems using TDOA, photon counting sensors as initially modeled in the MUSTC algorithm measure the distance between the transmitter and the receiver.

However, instead of indirectly measuring the distance to the transmitter by measuring the additional time it takes for the radiation to travel from one sensor to the other, photon counting detectors are currently modeled in the MUSTC program to directly measure the distance based on the amount of radiation that hits the sensor. If the sensor is further from the target, then it will measure less radiation than if it were closer. The program assumes that the radiation source and the sensor have been properly calibrated so they can be used for this purpose. However, if the system has not been calibrated properly then the program can easily determine the TLE caused by that discrepancy.

The inclusion of these vastly different types of sensors demonstrates the versatility of the MUSTC algorithm. Since the MUSTC model can handle RF as well as multiple types of optical sensors, it can also be expanded to model an even wider array of other PNT sensors.

2. METHODS, ASSUMPTIONS, AND PROCEDURES

2.1. Algorithm Description

In this section, the algorithms that drive the MUSTC program will be described. The algorithm is based on the fact that when a function (*f*) depends on multiple largely independent variables (*A*, *B*, and *C*), then the uncertainty in the function *f* (σf) can be estimated using the following equation:

$$\sigma_f \approx \sqrt{{\sigma_A}^2 + {\sigma_B}^2 + {\sigma_C}^2}. \quad (1)$$

Here, σ_A, σ_B, and σ_C are the uncertainties in the three variables (*A*, *B*, and *C*). When each variable uncertainty is known, they can be combined to estimate the total uncertainty in the function.

The model is also based on the fact that when trying to model the performance of a sensor system, the exact algorithm the system is using to find the location of a target may not be known. Even if the algorithm the system is using is known, creating a model based on this algorithm would require a different model for each sensor location algorithm that may be encountered. The algorithm that is used to locate an object can be difficult or unknown; however, if the location of the target is known ahead of time, then the raw measurements that the system feeds into its algorithm are often simple to model. Figure 1 highlights this point.

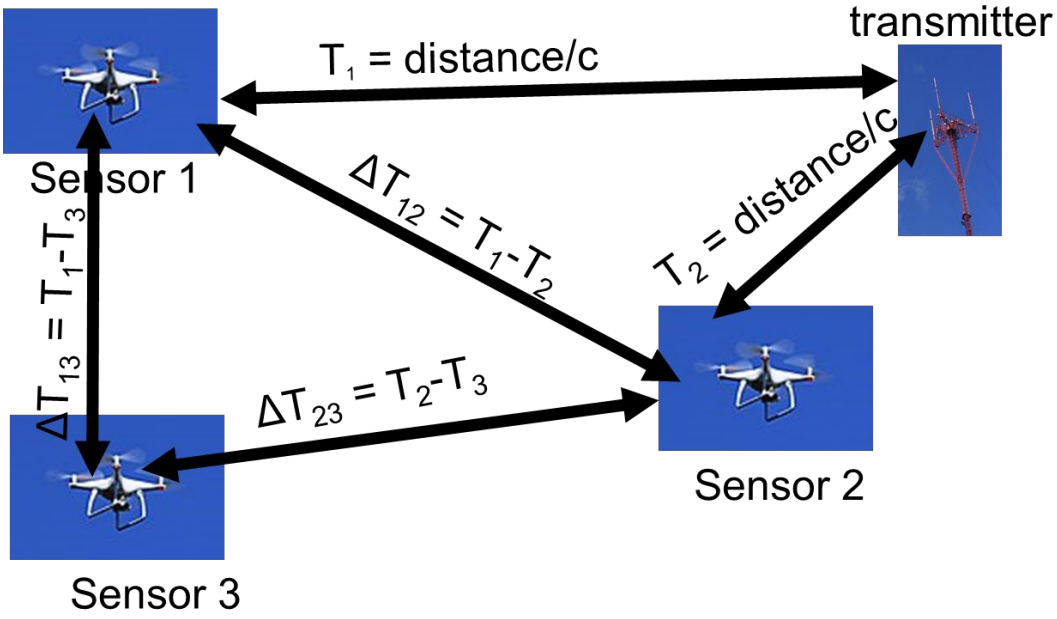

Figure 1. Geometry showing how to find the time difference for each sensor if the location of the emitter is known and the sensor is a SIGINT system using TDOA

As seen from Figure 1, when the emitter location is known, then the time it takes for radiation to hit one sensor is the distance between the sensor and emitter divided by the speed of light (*c*). The TDOA between two SIGINT sensors is therefore equal to the time it takes the radiation to hit one sensor minus the time it takes the radiation to hit the other sensor.

Because LRF systems can measure the range to the target directly, modeling them is even easier. LRF models do not need to divide by the speed of light or subtract the measurement that one sensor makes from the measurement of the other.

Photon counting detectors measuring radiation from a known light source also measure the distance between the emitter and the detector. However, the measurements they make are not as straightforward as the measurements LRF systems make. To model raw measurements from a photon counting detector, a link budget analysis must be performed to determine the amount of radiation from the emitter that will make it onto the detector.

If the emitter that the photon counting detector will measure is assumed to be emitting radiation in equal amounts in all directions, then the radiant intensity (power per solid angle) will be equal to the power transmitted divided by 4π. Because of the geometry of the experiment, the maximum amount of radiation that the sensor will have access to depends on the solid angle over which the sensor will have access to the emitter's radiation. Assuming that the sensor is a square, this solid angle will be equal to

$$\Omega = \frac{a^2 \cos(\theta)}{d^2}. \tag{2}$$

Here, Ω is the solid angle, a is the area of the detector, θ is the angle between the incoming radiation and the direction the detector is pointed, and d is the distance between the emitter and the transmitter. Thus, the radiation that hits the detector will be equal to

$$P_d = T\left(\frac{P_t}{4\pi}\frac{a^2 cos(\theta)}{d^2}e^{-\alpha d} + P_b\right). \tag{3}$$

Here, T is the transmittance of the optics attached to the sensor, α is the extinction coefficient of the atmosphere between the emitter and detector, P_t is the power of the transmitter, and P_b is the background radiation. Because most users will not want to flood their sensor with background radiation, they will either put a notch filter that blocks all radiation except for a very narrow band of wavelengths that covers the radiation the transmitter transmits, or they will only run the experiment in a dark location or at night. However, there will almost always be some additional background radiation.

Because the energy of a photon is equal to $h\nu$, where h is Planck's constant and ν is the frequency of the photon, it is easy to determine how many photons will hit the detector per second. The MUSTC algorithm uses the following formula to estimate the number of photons the photon counting detector will register:

$$p = \left(E\ P_d \frac{\lambda}{h\ c} + n\right)\tau. \tag{4}$$

Here, p is the number of photons counted, E is the photon detection efficiency, λ is the wavelength, n is the dark current or shot noise, and τ is the integration time.

Expanding on the analysis shown in Figure 1, it is possible to model the raw measurements that an even wider array of sensors will make if the location of the target and sensors are known. For EO/IR sensors, SIGINT systems using AOA, and/or SIGINT systems using FDOA, the algorithm only needs to consider the geometry of the scenario and the angles between different vectors. Figure 2 shows these angles and vectors.

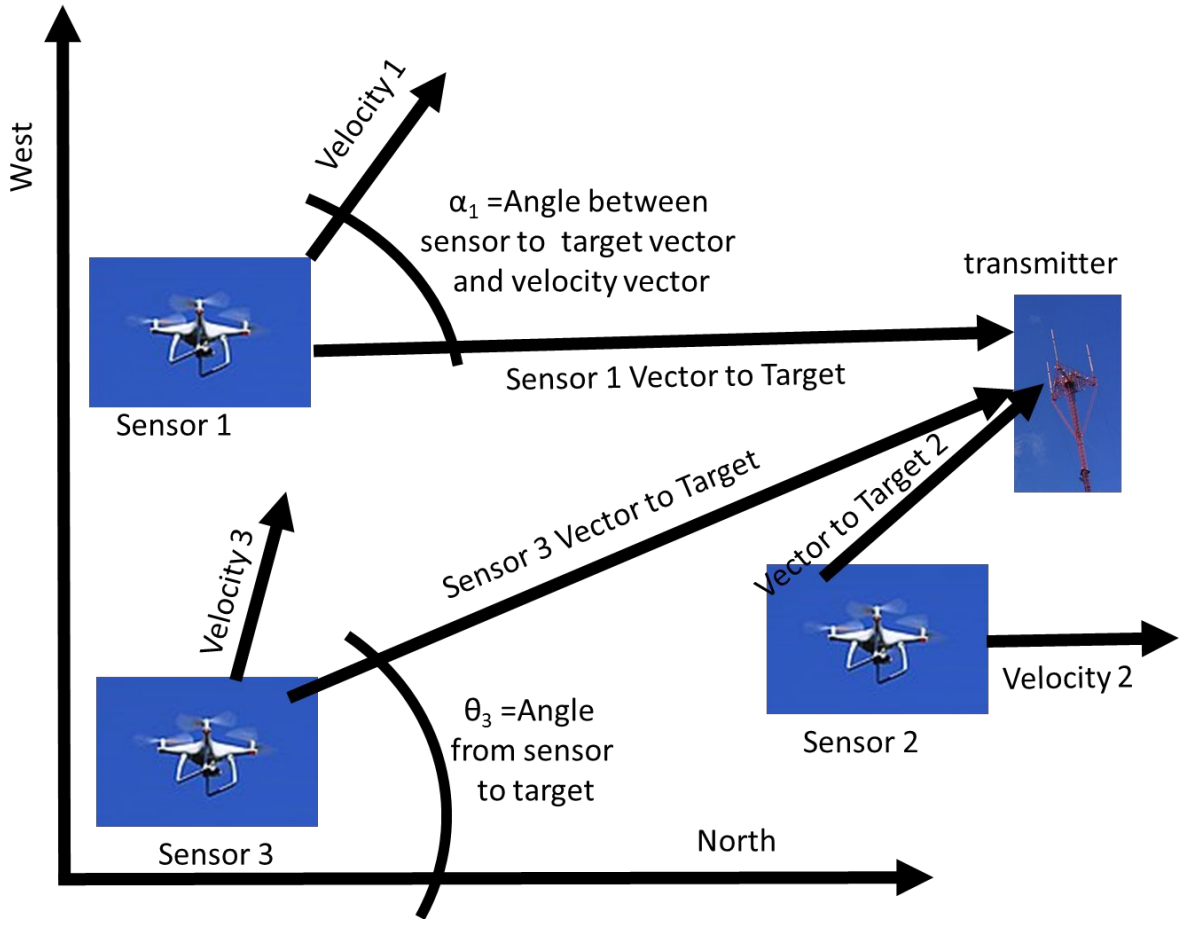

Figure 2. Geometry showing how to find the angles needed to model EO/IR, FDOA SIGINT, and AOA SIGINT when the location of the emitter is known

Figure 2 highlights the angles the algorithm requires to analyze EO/IR, AOA SIGINT, and FDOA SIGINT systems. In particular, the algorithm needs to find the angles that the radiation travels to go from the target to the sensors (called θ_1, θ_2, and θ_3 following the convention from Figure 2). EO/IR sensors or SIGINT systems using AOA can measure these angles directly and are therefore the only values that need to be considered when analyzing these types of sensors.

When analyzing SIGINT systems using FDOA, the algorithm must consider the angle between the vector from the sensor to the target and the velocity vector (α_1, α_2, and α_3 in Figure 2). Once these angles are found, the algorithm can use the following equation to determine the difference in Doppler shift induced frequency as measured by two sensors:

$$\Delta F_{12} = f_o \frac{|V_1| cos(\alpha_1) - |V_2| cos(\alpha_2)}{c}. \quad (5)$$

Here, ΔF_{12} is the difference in frequency measurement between Sensor 1 and Sensor 2, f_0 is the base frequency that the target is emitting, v_1 is the velocity of Sensor 1, α_1 is the angle between the vector from the sensor to the target and the velocity vector, v_2 and α_2 are the velocity and angle for Sensor 2, and c is the speed of light.

We now have a model that can estimate the raw measurements a sensor would make when the target of the location is known. DAC can therefore use this model to estimate how variables that could affect the final location measurement will affect the raw measurements that the sensors will make. Some of the variables that could affect the final TLE measurements include uncertainties in knowing the location of the reference assets and uncertainties in the actual raw measurement values that the sensors make.

It is trivial to use these simple raw measurement models to see how a measurement error will affect the raw measurements that the system will make—simply add this error to the results of the measurement model. Likewise, it is not difficult to use these measurement models to see how an uncertainty in location of a reference asset can affect the raw measurements the system will make—simply move the asset to a different location and see what the sensors would measure in this new scenario. In fact, it is possible to model how the raw measurements change as a function of almost any variable that could affect the final TLE.

However, it is not changes in the raw measurements the system will make as a function of these variables that is important. Instead, we would like to know how these changes in the raw measurements of the system can affect the final measurement of the location of the object of interest. To convert the uncertainties in the raw measurement errors to errors in the target's location, the MUSTC algorithm uses an optimization algorithm.

Optimization algorithms require a cost function and an initial guess. The cost function is a function that reaches its minimum value (or maximum value if configured to do so) when it is fed the final values of the parameters that are of interest. The optimization algorithm will feed the initial guess to the cost function. The optimization algorithm will then begin searching around the initial guess to find a parameter set that returns a cost function that has a value that is less than the value at the initial guess. The algorithm will keep searching until it finds the cost function's minimum value. Ideally, this minimum value will be the global minimum.

The cost function that the MUSTC algorithm feeds to the optimization algorithm is the average difference between all the raw measurement values after an error has been added to the variable of interest and the measurement values when the error has been removed but the target of interest has been moved to its new location. The optimization algorithm will move the object of interest around until it finds a place where the average difference between the raw measurements at this new location is as close as possible to the raw measurements when the target of interest was at its original location; however, an error has been added to one of the variables that could affect the TLE. The difference between the new location of the target of interest and the starting location is the error in location measurement that results from the uncertainty in the variable of interest. In this way, the MUSTC algorithm can find the error in location for an uncertainty in almost any variable of interest. The algorithm can do this for a wide array of configurations and sensor types even if the system's exact target location algorithm is not known.

One pitfall of using an optimization algorithm is the possibility of finding a local instead of a global minimum. This pitfall can be mitigated by having a good initial guess and by using an optimization algorithm that is well suited for this particular problem. Because the program that implements the MUSTC algorithm is so modular, it is easy to try a multitude of optimization algorithms to see the one that yields the best results. Appendix B discusses how to select the best optimization algorithm for the application.

One way to ensure a good initial guess is by starting with a small error on the variable of interest. Because the error is small, the location measurement error will also be small and therefore the starting location of the object of interest will be a good guess. Once the location error for this small measurement error or perturbation is known, we can add a larger perturbation to the variable of interest. The initial guess for this slightly larger perturbation on the variable will be the location error from the slightly smaller perturbation tested earlier. In this way, it is possible to map out how greater errors or perturbations on the variable of interest can lead to greater errors in the measurement

of the location of the target of interest without having to worry about hitting a local instead of a global minimum.

Figure 3 highlights an example of how a perturbation or error on the time measurement of a single SIGINT sensor in a group of SIGINT sensors using TDOA to find an emitter can lead to an error in the measurement of the RF emitter location.

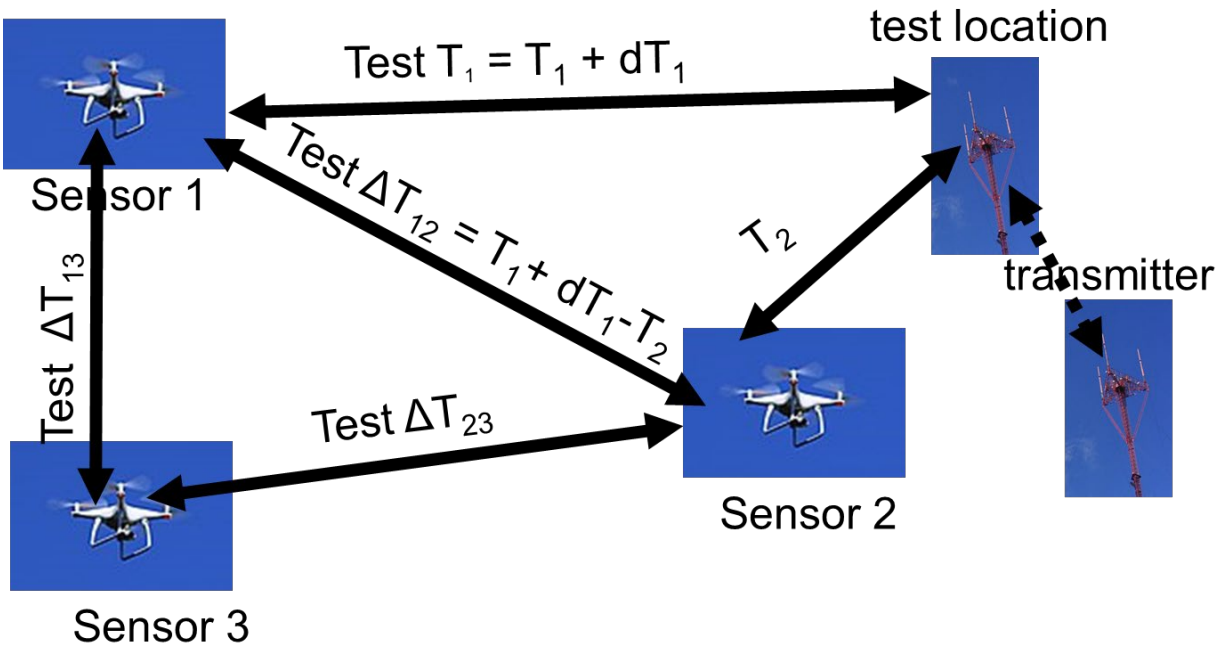

Figure 3. Example showing how to determine the error in finding the emitter location if a perturbation is added to the time of arrival SIGINT system as measured by Sensor 1

As seen in Figure 3, the MUSTC algorithm added a perturbation (dT_1) to the time it takes for the radiation to hit Sensor 1. The new difference between the time it takes for the radiation to hit Sensor 2 after it hits Sensor 1 is given by

$$\Delta T_{12} = T_1 + dT_1 - T_2. \tag{6}$$

Here, ΔT_{12} is the time difference of arrival between Sensors 1 and 2, T_2 is the time it takes for the radiation to hit Sensor 2, and T_1 is the time it takes for the radiation to hit Sensor 1. They are both raw measurements a TDOA SIGINT system will make when finding the location of an emitter. Once the algorithm adds this perturbation, it tasks the optimization algorithm with moving the location of the emitter around until it finds a test location that will produce the same time difference of arrivals as it obtained after it added the perturbation (or at least it can find a location that produces time differences that are as close as possible to the time differences it obtained after adding the perturbation).

To translate this perturbation into an uncertainty in location based on an uncertainty in the time measurement for Sensor 1, the algorithm adds a series of perturbations to Sensor 1. If the location error caused by the *i*th perturbation is ΔL_i, and the probability of the *i*th perturbation is P_i, then the total uncertainty in the emitter location caused by an uncertainty in the variable the algorithm is perturbing (σ_l) is given by

$$\sigma_l = \sqrt{\sum_{i=1}^{n} P_i \, (\Delta L_i)^2}. \tag{7}$$

If the number of perturbations is too low, and thus the distance between perturbations is too high, then not only will this cause the possibility of the optimization algorithm finding a local instead of a global minimum, but it might also cause a round-off error based on the fact that there might not be enough probability values in Equation (4). On the other hand, if there are too many perturbations the calculation could require excessive computational time.

When the distance between perturbations was initially chosen, it was decided to err on the side of accuracy at the expense of computational time. One way to be sure that the perturbation size is small enough to be acceptable is to ensure that distance between perturbations is in a range such that small changes to these distances does not produce significant changes to the final calculated TLE values. Updates made to the algorithm in order to decrease the computational time are discussed in Appendix C.

Figure 4 shows the results the algorithm gets when it tries to find how an uncertainty in the value of a single variable can cause an uncertainty in finding the object of interest's location in a single direction.

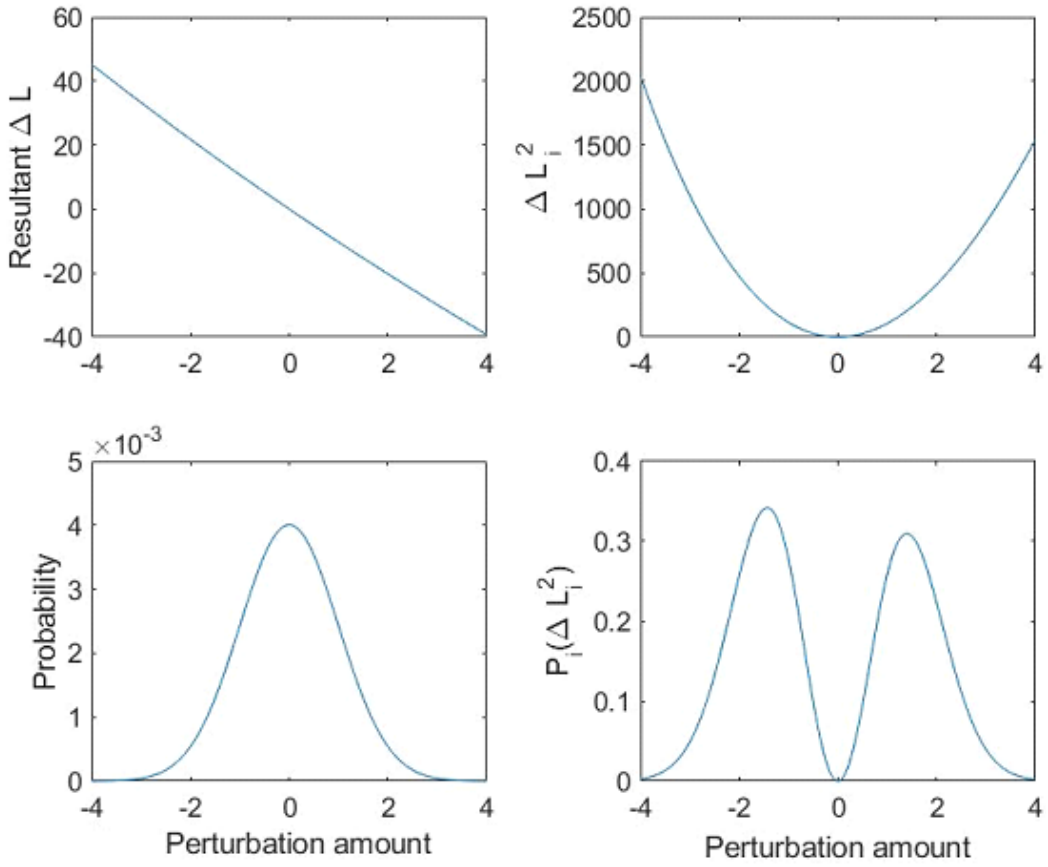

Figure 4. Example values used to find the uncertainty of the target of interest's location in a single direction caused by an uncertainty in knowing the value of a single variable

The upper-left plot in Figure 4 shows the difference in location between the object of interest and the test location of the object after the algorithm adds perturbations that go from –4 to +4 standard deviations from the nominal value of the variable.

The upper-right plot in Figure 4 shows the square of the difference in location. The bottom-left plot shows the probability of the *i*th perturbation. The program used a Gaussian distribution for the calculations in Figure 4. However, if the program were to explore uncertainties in the raw measurements of the photon counting detector, it would use a Poisson distribution instead. If additional distributions are required, it should be easy to add them. For all distributions, the probability function must be normalized so that the sum of all probabilities is equal to 1.

The bottom-right plot shows the probability of the perturbation multiplied by the square of the difference in the location after adding each perturbation. If the algorithm takes the square root of the sum of all points in this plot, then it will be left with the total uncertainty in finding the location of the object in a single direction caused by an uncertainty in knowing the value of a single variable.

The algorithm must repeat the calculations in Figure 4 three times to find a 3-D TLE. The algorithm must then repeat these calculations for all variables that could affect the final TLE. Once the algorithm has all the individual TLE values, it can use Equation (1) to combine these into the final 3-D TLE.

2.2. Determining the Uncertainty in Each TLE Measurement Variable

The MUSTC algorithm determines how uncertainties in different variables can affect the TLE of location measurements. Before anyone can use the algorithm, they must be able to produce estimates on the magnitude of the uncertainties on these variables. If a different sensor type is added to the MUSTC program, the program will need to be extended so that it can find the uncertainties on the variables that can affect these new sensor types. This section will outline how to determine the uncertainties on the variables for the sensors that were initially included in the MUSTC program (i.e., SIGINT sensors, EO/IR sensors with LRF, and photon counting detectors).

For some of the measurements that the sensors make, estimations for these uncertainties are available on the sensor's specification sheet. For example, the uncertainties in angle as measured by SIGINT systems using AOA or EO/IR sensors, or ranges as measured by LRF sensors, are usually predetermined by the manufacturer and included in the specification sheet. Regarding the uncertainty in counting photons, because uncertainties in this variable are represented by a Poisson distribution, the standard deviation will be equal to the square root of the mean signal.

There are other uncertainties that are not typically listed on a sensor's specification sheet but that can still be determined relatively easily. The uncertainties in the speed and location of the platforms that contain sensor assets are examples of such variables. Many platforms will use a GPS to find their location. The uncertainty in measuring location via GPS will depend on the scenario, but an uncertainty of around 5 m in each direction is a reasonable rule of thumb for this value. However, the location uncertainty may be different for different scenarios and may change over time. For example, a moving aircraft will often pair a GPS receiver with an inertial navigation system (INS) to augment the location measurement.

INS sensors are often used to augment the location measurement to ensure that the aircraft will have an estimation of the location even when GPS values are not available (i.e., if it is in between the times when the GPS will update the location value or if the GPS is currently being jammed or denied). In this case, the actual location uncertainty will depend on the INS sensor and the length of time that the sensor has gone without a fresh GPS location measurement.

Another factor that must be considered when measuring the location uncertainty is that this uncertainty can often be broken into two measurements: a location relative to other sensors that might be in use and a location relative to the center of the Earth. If each sensor only has a GPS and/or INS to find its location, then these two measurements are combined into one. However, if the scenario has systems that have a method of directly measuring the location of the sensors relative to each other (perhaps multiple sensors are attached to the same aircraft) then these measurements (the location of the formation relative to each other and relative to the center of the Earth) must be split in two and considered separately.

As with location, the measurement of velocity can depend on the sensor and the scenario. For example, some sensors may use GPS and/or INS to estimate the velocity as well as the location. Such a velocity measurement will depend on many of the same factors that are used to estimate the location uncertainty.

For SIGINT systems using TDOA and/or FDOA, it is necessary to know the uncertainties in the time and/or frequency measurements that the sensors make. However, these measurement uncertainties are not typically included in the specification sheets for these sensors. The algorithm can use the standard formula to estimate the uncertainty of these values based on standard emitter and sensor parameters that can be found in the specification sheets for these products.

The standard formula for the uncertainty in time measurement that the algorithm uses is given by (Poisel, 2005):

$$\Delta T = \begin{cases} \sqrt{\frac{3}{4\,\pi^2 \tau}}\ \frac{1}{\sqrt{SNR}}\ \frac{1}{(f+b)^3 - f^3}, & SNR > 1000 \\ \sqrt{\frac{1}{8\pi^2}}\ \frac{1}{SNR}\ \frac{1}{\tau\, b}\ \frac{1}{f}\ \frac{1}{\sqrt{\frac{1+b^2}{12\, f^2}}}, & SNR \leq 1000 \end{cases}. \tag{8}$$

Here, ΔT is the uncertainty in the time measurement, τ is the integration time, SNR stands for the signal-to-noise ratio (SNR), f is the frequency, and b is the bandwidth. There are other factors that could affect the time measurement uncertainty that are not outlined in Equation (8) such as digitization errors.

The algorithm uses this equation to estimate the uncertainty in measuring the frequency of the incoming radiation (Poisel, 2005):

$$\Delta f = \frac{\sqrt{3}}{2\,\pi\,\tau}\ \frac{1}{\sqrt{b\,\tau\,SNR}}. \tag{9}$$

Here, Δf is the uncertainty in the frequency measurement and the other variables are the same as in Equation (8).

Equations (8) and (9) demonstrate that both the frequency and time measurement uncertainties depend on the SNR of the SIGINT system. The algorithm uses two different formulas to find the SNR, as seen in Equations (10) and (11). The specification for many SIGINT systems will include a value called the receiver sensitivity. If the receiver sensitivity is available, the algorithm will use the following equation to estimate the SNR:

$$SNR = \frac{1000\ P\ G_r\ G_t\ \lambda^2}{(4\ \pi\ r)^2\ L_r\ L_t\ \frac{s}{SNR_{ref}}}. \tag{10}$$

Here, P is the power in watts, G_r is the receiver antenna gain, G_t is the transmitter antenna gain, λ is the wavelength, r is the range, L_r is the receiver loss, L_t is the transmitter loss, s is the receiver sensitivity, and SNR_{ref} is the reference SNR. All the values in Equation (10) must be linear and not in decibels (dB) (if dB values are used then the values in Equation [10] should be added and subtracted instead of multiplied and divided). If the sensor is an Electronic Intelligence (ELINT) system, then SNR_{ref} should be set to 8 Hz. For Communications Intelligence (COMINT) systems, SNR_{ref} should be set to 13 Hz.

If the receiver sensitivity is not available, the algorithm can use the following equation to estimate the receiver sensitivity divided by the reference SNR:

$$\frac{s}{SNR_{ref}} \approx 1000\ k\ T_n\ n_f\ b. \tag{11}$$

Here, k is the Boltzmann constant, T_n is the noise temperature, n_f is the noise figure, and b is the bandwidth. Once again, the parameters needed to calculate the SNR for a SIGINT system are usually given in the system's and target's specifications. Note that because the purpose of the MUSTC algorithm is to model the behavior of sensors in various hypothetical scenarios, the information about the scenario including the target's specifications should always be available.

2.3. Normalizing the Sensor Measurements

The MUSTC algorithm works by collecting the different raw measurements that the sensors make when a perturbation is added to a variable of interest. The algorithm then calls an optimization algorithm to move the location of the object of interest around until it produces a set of measurements that matches the one that was produced after it added the perturbation.

The optimization algorithm treats each measurement equally and it does not know or care where those measurements come from. An optimization algorithm is only concerned with minimizing the overall magnitude of the cost function.

Combining multiple types of measurements with different units together can cause a problem for the algorithm. If one sensor type measures the time it takes for something to happen in nanoseconds while another one measures in milliseconds, the nanosecond measurement will likely have a higher magnitude simply because of its units. Since the optimization algorithm strives to minimize the cost function and the nanosecond measurement will have a larger effect on the cost function, the optimization algorithm will naturally try to lower the nanosecond measurement before moving to the millisecond measurement. In fact, the optimization algorithm may reduce the error in the nanosecond measurement at the expense of the millisecond measurement. Therefore, before combining different measurements with different units in this way, it is important to find a way to normalize them so that their magnitudes are comparable.

Before the MUSTC algorithm feeds the sensor measurements to an optimization algorithm, it performs a test to find the maximum and minimum value that the sensors will make while it processes the scenario. The algorithm does this by adding the maximum positive and negative errors that the algorithm will add to the assets' locations and velocities (as well as any other variable that could affect the TLE besides uncertainties in the measurements themselves). Because the maximum and minimum values that the sensor finds during this test were found while the variables were at their extreme values, these maximum and minimum values are likely to be the maximum and minimum values that all of the sensors will measure during the entire TLE calculation.

Once the algorithm finds these maximum and minimum values, it can normalize the sensor measurements, so that they range between 0 and 1. To do this, the algorithm subtracts the minimum value of each sensor's measurement and then divides by the maximum value minus the minimum value. Once the sensors' magnitudes are normalized, the optimization algorithm will no longer give preference to one sensor's measurement over another.

While the default configuration is that each sensor's measurement is given the same consideration, there may be scenarios where this is not desirable. For example, many SIGINT systems cannot make as accurate an angle measurement as they can for the time and frequency difference of arrival measurements. These systems will often use TDOA and FDOA to find the exact location of the emitter and only use AOA to remove ambiguities that might arise from not having enough equations for the unknowns. In this case, it is preferable that the algorithm weigh the time and frequency measurement higher than the angle measurements.

The MUSTC program has a feature to add a weight to each sensor measurement in the scenario. By adding a weight of 0.1 to the angle measurement, instead of going between 0.0 and 1.0, the program will normalize the angle measurement, so that it ranges between 0.0 and 0.1. This feature will allow the optimization algorithm to try to get a better fit with the other measurements than it does with the angle measurement and may even make the angle measurement worse to get a better measurement with another sensor with a higher weight. This adds even more functionality to the MUSTC algorithm and allows users to test scenarios and configurations that may be difficult to predict using other methods of calculating the TLE of a location measurement.

2.4. Algorithm Inputs and Outputs

Before the program can find the TLE for a scenario, users must inform the program on the configuration of the scenario. Users must supply the algorithm with the following inputs to configure the scenario itself:

- One or more points in space to find the TLE of the item of interest. These locations may each be given in 2- or 3-D. If the asset is on the ground, a 2-D coordinate will suffice since the Z direction (height) will be 0.
- One or more scenarios. Each scenario must include one item of interest whose location is being determined via the system(s) described in the scenario. A scenario must also include one or more asset(s) whose location(s) are known and are to be used to determine the item of interest location. If the user supplies multiple scenarios, then the program will find the TLE for each location and for each scenario.
- Each asset contains a list of sensors and/or targets that are mounted on the asset. Either the sensor or the target list can be empty; however, the asset should have at least one sensor or target if it is to be of use to the MUSTC program. If the item of interest is not linked to a sensor or target, then it is impossible to find its location and the algorithm will not be able to find a TLE for this object.

In addition to overall inputs about the scenario, users may also have to supply input parameters for some of the targets and sensors that are within the scenario. These inputs allow the sensor and target pair to determine the magnitude as well as the uncertainty in the measurements that the sensor will be making. The required inputs for a sensor or target depend on the sensor or target type. The MUSTC program is highly modular and configurable; therefore, it is possible to add more sensor and/or target types to the program. More information on how to add a new sensor type to the MUSTC software is shown in Appendix A. Currently the only targets included with the MUSTC software are optical targets and RF and optical emitters. The only sensors currently

included with the MUSTC software are EO/IR sensors with LRF; photon counting detectors; and AOA, TDOA, and FDOA SIGINT sensors.

Currently optical targets do not require any additional inputs. The program assumes that every EO/IR sensor that measures the angle to the target can detect any optical target. The program also assumes that the configuration of the target does not change the uncertainty with which an EO/IR sensor with an LRF can measure a target location. The program can be expanded in the future to include factors that could affect the ability of an EO/IR sensor to accurately determine the location of an optical target. The following lists the additional inputs that are required for other targets/emitters:

- Omnidirectional optical emitters that photon counting detectors measure require the power and wavelength of the radiation of the emitter. The uncertainty of the power of the emitted radiation may also be provided. However, if the power uncertainty is not supplied the program assumes it is negligible.
- The RF emitters require the following inputs to find the SNR that the sensor will measure: power transmitted, transmitter antenna gain, transmitter signal loss, and the center frequency of the emitted radiation (the center frequency is only used to find the SNR and not considered when determining if a sensor can detect an emitter). Users also have the option of adding the following inputs to inform the program as to which emitters can be detected by which sensors: the maximum and minimum frequency that the emitter will emit as well as the type of emitter the target is (radio, radar, or jammer). If these inputs are not included the software will assume that any SIGINT sensor in the scenario will have no problem detecting the emitter.

The following inputs are required for sensors in the scenario:

- For EO/IR angle sensors, it is assumed that all EO/IR angle sensors can at least measure the azimuth angle to the target; therefore, the MUSTC software requires the uncertainty with which the sensor can measure the azimuth angle to the target be specified. Additionally, some EO/IR angle sensors can measure the elevation angle to a target, and some also have an LRF that can measure the range to the target. The uncertainty in elevation angle and/or the uncertainty in range that the EO/IR sensor with LRF can measure these things can also be provided as inputs. If one or more of these uncertainties are not included, the software will assume that detector cannot measure the range and/or elevation to the target.

There are three types of SIGINT sensor systems that are currently included in the MUSTC software: AOA, TDOA, and/or FDOA sensors. While a SIGINT sensor might be

able to use all of these algorithms to find the location of an object, the MUSTC software assumes that a sensor can only use one algorithm to find the emitter location. If the scenario includes a SIGINT system that can use multiple algorithms to find an emitter, it must be modeled as if there are multiple SIGINT sensors installed on the same asset. The following inputs are required for RF SIGINT sensors:

- RF sensors have optional inputs that inform the software as to which sensor can detect which emitter. In particular, users can specify the maximum and minimum frequency that the sensor can detect along with the type of sensor. The two possible SIGINT sensor types are ELINT and COMINT. ELINT systems are designed to detect things such as radar, while COMINT sensors are designed to detect things such as radios. It is assumed that they can also detect jammers. If these optional parameters are not given, the software assumes that the sensor can detect any emitter in the scenario.
- AOA SIGINT sensors require the uncertainty with which they can measure the angle to the sensor. The software assumes that any AOA SIGINT sensor can measure the azimuth angle to the emitter. If AOA sensors are added to the scenario, the uncertainty with which the azimuth angle can be measured must be included. Some AOA SIGINT sensors can also measure the elevation angle to the emitter. Users have the option to add the uncertainty with which the sensor can measure the elevation angle to the emitter. If the elevation angle uncertainty is not provided, the software will assume that the sensor cannot measure elevation angle.
- To model TDOA and/or FDOA SIGINT sensors, the MUSTC software must know the uncertainty with which the sensors can measure frequency and or time difference between sensors. Unlike angle and range uncertainties, these uncertainties are scenario dependent and therefore are not typically included on the specification sheets supplied by manufacturers. However, most reference books on SIGINT systems will include formulas to determine the uncertainty with which a SIGINT system can measure time and/or frequency difference as a function of the SNR of the SIGINT measurement. Therefore, TDOA and FDOA sensors require inputs that allow the MUSTC software to estimate the signal and noise of the SIGINT measurements. Receiver antenna gain, receiver signal loss, system bandwidth, and integration time are required parameters to estimate the signal part of the SNR of an RF measurement.
- Parameters to find the noise part of the SNR are also needed to model TDOA and FDOA SIGINT sensors. Some specification sheets for SIGINT sensors will include the receiver sensitivity of the sensors. If the receiver sensitivity as well as the reference SNR are supplied, the MUSTC software can model the noise in a SIGINT time or frequency difference measurement. However, since not all

specification sheets supply the receiver sensitivity, the noise figure of the SIGINT sensor may also be used as an input. The MUSTC software can also use this noise figure to estimate the noise in the measurement if receiver sensitivity and/or reference SNR are not given.

- Digitization errors are uncertainties caused by a sensor's analog-to-digital converter not having enough bits to measure a quantity with enough precision. These errors can affect a SIGINT system's ability to measure the uncertainty in the time and frequency difference measurement. While the formula that relates the SNR to time and frequency measurement uncertainty may suggest that systems should be able to measure these quantities with amazing accuracy when the sensor has extremely high SNR—in reality, it might be difficult to design an actual system that can measure time and/or frequency with that much precision. Therefore, the option to add a digitization error component to the TDOA and FDOA sensors in the MUSTC software is available.
- When it comes to photon counting detectors, inputs such as the visibility in miles of the atmosphere between the emitter and detector, the background radiation that the sensor will pick up, the dark current (a.k.a., the shot noise of the detector), the optical transmittance of the optics on the detector, the representative area of the sensor, the photon detection efficiency, and the integration time are needed. Optional inputs include the azimuth and elevation representing where the detector is pointed. If these angles are not supplied the system will assume that the detector is always perfectly aligned to pick up the maximum signal from each detector. The minimum and/or maximum wavelength of the radiation that the emitter can detect are optional inputs as well.
- In addition to the required parameters, there is a list of optional photon counting detector parameters that can be included. These parameters inform the program on the uncertainties of some of the supplied parameters. The list includes uncertainties in the following: visibility, background radiation, dark current, photon detection efficiency, and the azimuth and elevation where the detector is pointed. If the uncertainty in azimuth and elevation are provided but not the azimuth and elevations themselves, the system will assume that the operator tried to point the sensor to the optical emitter, but their aim was off by the given amount.

Uncertainties in knowing the location of reference objects can cause uncertainties in knowing the final location of the item of interest. Therefore, in addition to the uncertainties in the values that each sensor will measure, the algorithm must know the uncertainties in the locations of the reference items that will be used to find the location of the item of interest.

If there are no FDOA or other sensors in the scenario that depend on velocity, the software does not require the velocity of any of the assets to be set. However, if a scenario does use an FDOA SIGINT or similar sensor, then at least one asset must have its velocity set. In addition to the velocity value itself, the uncertainty in the velocity may also be entered. As with other uncertainties, the software will use the velocity uncertainty to determine how errors in the velocity measurements can lead to errors in the final TLE. If the velocity is not given the software will assume the asset will remain stationary, and if the uncertainty in the velocity is not given the software will assume that the uncertainty in velocity is negligible.

The MUSTC software expects that the location of all reference assets in the scenario will be supplied. However, the location for the item whose location the software is finding the TLE for (a.k.a., the object of interest) will not be supplied. Instead, the software will automatically set the nominal location of the object of interest to the locations where it is to calculate the TLE. The location uncertainty of the assets, except the item of interest whose location uncertainty will be the final TLE, should also be supplied. If the location uncertainty is not supplied for a reference asset, then the software will assume that the uncertainty in finding this location is negligible. However, there is an option to say that the uncertainty in location of a reference asset was based on an earlier location TLE calculation. The program can find this TLE ahead of time and then set this TLE to be equal to the uncertainty in the asset location.

For example, when an emitter location is to be found using a group of SIGINT sensors and the location of these sensors is not known ahead of time, the locations of the sensors must first be determined using a sensor such as an EO/IR sensor before they can be used to find the emitter location. It may be of interest to know how uncertainties in the initial EO/IR measurements of the location of the sensors will affect the final TLE of the emitter. For this type of scenario, the nominal location of the SIGINT sensors must still be provided. However, the user can specify that the uncertainty in the location of the SIGINT sensors is based on an earlier location measurement. The system will automatically examine the EO/IR sensors to determine the TLE in finding the location of the SIGINT sensors. The program will then carry over these TLE values as the uncertainties in knowing the location of the SIGINT sensors for the final calculation of the TLE of the emitter.

What if the scenario dictates that the sensors used to find the uncertainty in finding a reference asset cannot be used in the calculation to find the final TLE of the target of interest? For example, perhaps the SIGINT sensors also have EO/IR sensors, but those EO/IR sensors were not available during the first part of the experiment. For this reason, all targets have an optional inclusion number, and all sensors have an optional inclusion

number range. If the inclusion number or range are not specified, the program will assume the sensor will always have access to the target. However, if the target is given an inclusion number, and that number is outside of the range that was given to the sensor, then the software will not allow that sensor to detect that target—even if it would normally be able to be based on the other parameters the target and sensor were given.

Many sensors, such as SIGINT using TDOA and/or FDOA, take measurements that only make sense if their values are subtracted from each other. For example, SIGINT systems using TDOA cannot measure the exact time it takes for the radiation to travel from the target to the sensor. However, they can measure the additional time it takes to hit one sensor over another. If the MUSTC program encounters these types of sensors, it must determine which sensors are allowed to be paired together. When a scenario dictates that not all TDOA or FDOA sensors can communicate with each other, sensors in the MUSTC program can be assigned sensor inclusion numbers and ranges. These are similar to the target inclusion number as they restrict some sensors from pairing even if their parameters and types imply that they are compatible.

Once the information the program needs to perform one or more TLE calculation has been supplied, the program can perform these calculations. Once the calculations are finished, the program supplies the following outputs:

- The uncertainty (standard deviation) in measuring the location of the asset of interest in the X- and Y-directions is provided as an output. If this is a 3-D problem, the uncertainty in the Z-direction will also be given.
- The software also can turn the uncertainties in the X- and Y-directions into a 50% circular error probability (CEP50). The CEP50 is the size of a circle that would need to be drawn to ensure that 50% of all location measurements the system might make are inside the circle and 50% of the measurements are outside of the circle.
- The software can calculate the TLE at multiple locations and for multiple scenarios at once; therefore, the location where each TLE was calculated along with the parameters for each scenario are also outputs. Users have the option of naming each scenario to help reduce confusion; different scenarios can have similar parameters.

2.5. Possible Sources of Error

Depending on how the geometry is set up, it is possible there may be multiple locations where the target could be that would yield the exact same sensor measurements. For these scenarios, the sensors would not be able to determine the target location. If the algorithm determines this is the case, the uncertainty in all directions will be set to

positive infinity. When seeking the location uncertainty for many points, there could be some locations where the TLE will be set to infinity and others where they may not. This can happen if, for example, the user is using SIGINT AOA or EO/IR without an LRF and all of the sensors are in a straight line that intersects the target location being tested. Figure 5 demonstrates this scenario.

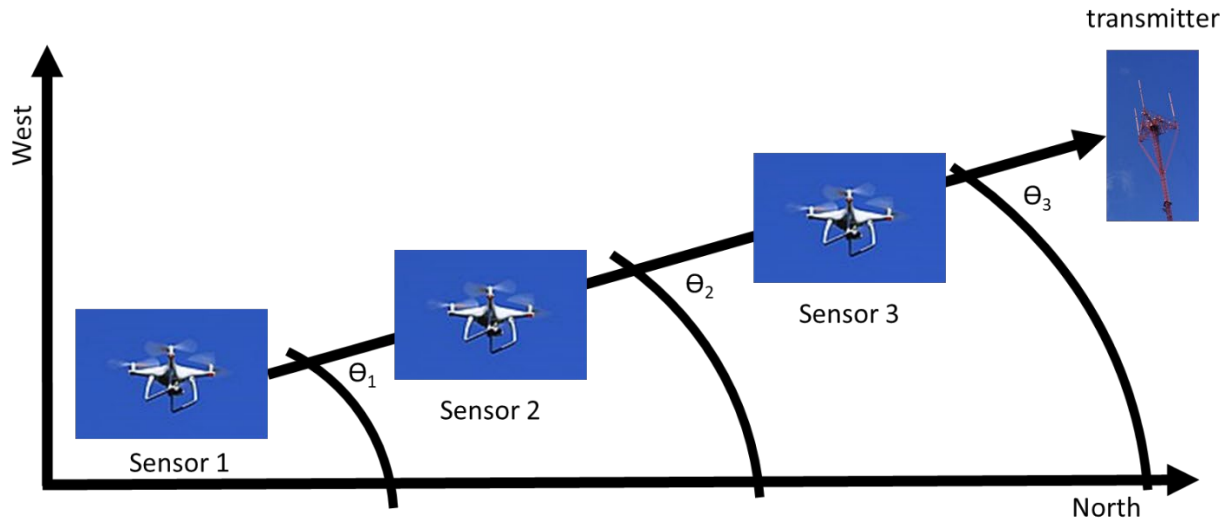

Figure 5. Configuration in which a SIGINT AOA cannot be used to find an RF emitter

Note in Figure 5 that if all three sensors and the transmitter are aligned, then no matter how far the transmitter is from Sensor 3, all three sensors will measure the exact same incoming angle. If the transmitter was not aligned with the sensors, the sensors in Figure 5 would have no problem using SIGINT AOA and/or an EO/IR with no LRF to find the location of the transmitter. The TLE may continue to increase as the transmitter moves closer to the alignment shown in Figure 5.

A similar phenomenon can occur with FDOA sensors with certain geometries as the frequency difference measurements also depend on the angles in the scenario. If the program detects that the scenario might be similar to what is shown in Figure 5, it will set the TLE for all measurements along the straight line to positive infinity.

Something similar to what is shown in Figure 5 can also happen when the systems are using SIGINT TDOA. Figure 6 highlights such a scenario. In Figure 6, the distance from Sensor 1 to Sensor 2 (listed as distance 1 in Figure 6) is equal to the distance between Sensor 2 and Sensor 3 (listed as distance 2 in Figure 6). No matter how far the transmitter is from Sensors 1 and 3, if the transmitter is on the 45° line, the sensors will measure the same TDOA. At a time equal to t_o, the radiation will hit Sensor 1 and Sensor 3 at once. Then, at a time equal to t_o plus the distance between the sensors divided by the speed of light, the radiation will hit Sensor 2. If the transmitter in Figure 6

were to move off the 45° line, the sensors would have no trouble using TDOA to find the target location. As before, if the program detects a scenario similar to what is shown in Figure 6, it will set the TLE equal to positive infinity for all TLE measurements along the 45° line.

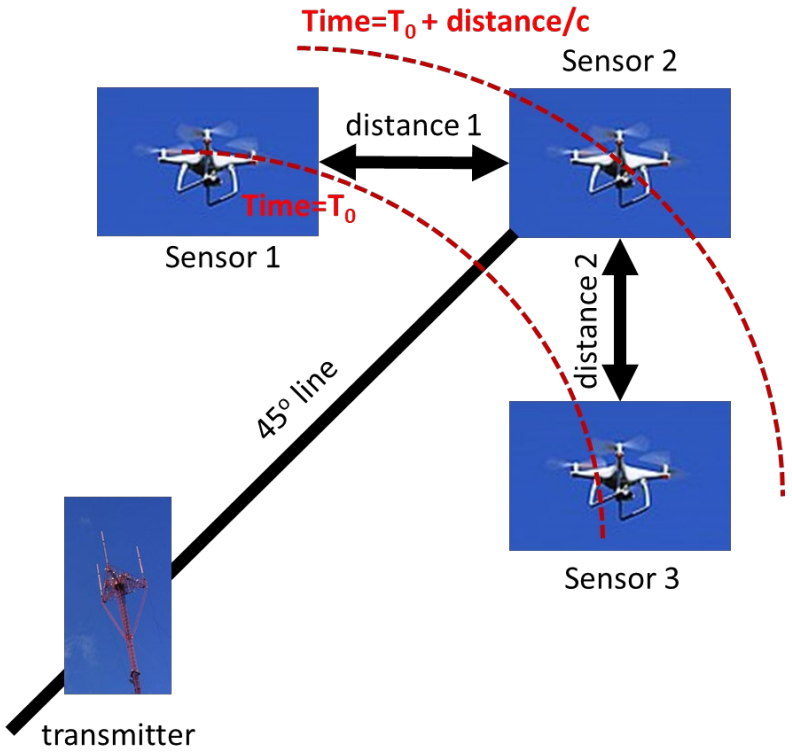

Figure 6. Scenario where TDOA cannot be used to find the target location

Therefore, if a user gets both finite and infinite results it is advisable to confirm the number of sensors being used and to recheck the geometry to ensure there are not multiple sensors making the exact same measurement.

There may be times when the geometry makes it impossible for the sensors to find the target, but the algorithm included with the MUSTC software is not able to detect this. Typically, when this happens, the optimization algorithm will become unstable. When the algorithm finds multiple locations that yield the same value for the cost function, it will start searching further away from the initial guess trying to find an area where the cost function does decrease. Often the algorithm will give up and return a random location that is far away from the target of interest's starting location.

When the algorithm enters a scenario where there are not enough measurements, and it is not able to automatically detect a problem, it will typically calculate a TLE that, while finite, is large enough that it can easily be flagged as unrealistic. Although this is what

the optimization algorithm typically does, there is no guarantee it will do so in every instance. There may be instances where an unusually large TLE is in fact the correct answer and this value is not due to instabilities in the optimization algorithm. It is therefore advisable to try to avoid performing calculations in areas where there are not enough measurements to determine the location of the item of interest.

2.6. Adding the MUSTC Algorithm to Combat Simulation Software

Any program can use the algorithm outlined in this report by using the DAC MUSTC Dynamic Linked Library (DLL). Appendix A demonstrates how to use the MUSTC DLL in another program. Sometimes it might be better for developers to forgo the DLL and instead to directly implement the algorithm into their program; specifically, combat simulation software developers may be among those who may prefer to add the TLE algorithm to their programs directly.

The TLE algorithm walks through all possible errors that each variable is likely to have. The algorithm then calculates the probability of each error and uses that to calculate the final TLE—as opposed to many combat simulation programs that use more of a Monte Carlo approach.

Monte Carlo-type algorithms test how the uncertainty of one variable can affect another by adding a group of random values to the variable whose uncertainty is being explored. The programs will then examine the statistics of the values that are affected by the random error it placed on the variable.

While Monte Carlo-type algorithms are performing calculations quite similar to the ones the MUSTC algorithm calculates, the difference is that the MUSTC algorithm systematically goes through a set of errors on each variable while Monte Carlo-type algorithms pick the errors to add to each variable randomly. The MUSTC algorithm can therefore be modified to work in a combat simulation program that performs Monte Carlo-type calculations.

Combat simulation programs can use an optimization algorithm similar to the one that the MUSTC algorithm uses when adding perturbations to each variable to see how errors in measuring an object's location are affected by random values added to other variables. The main disadvantages to using the Monte Carlo approach are that it does not systematically examine each possible value for the variable being explored, and it cannot use a previous result from the optimization algorithm as an initial guess for the next result the optimization algorithm is charged with finding. However, choosing an extremely robust optimization algorithm and testing for a distribution of variable errors

that is large enough to be statistically significant allows for a Monte Carlo-type approach to finding a sensor TLE.

3. CONSTRAINTS, LIMITATIONS, AND ASSUMPTIONS

3.1. Constraints

The major constraints for this model are as follows:

- The algorithm must be able to find the uncertainty when locating the object of interest in a reasonable amount of time.
- There are a finite number of optimization algorithms available to try and they usually can only find one minimum.
- It may not be known how the uncertainty in one measurement might be correlated to the uncertainty in another measurement. For example, when analyzing a set of SIGINT sensors that are trying to locate an emitter using TDOA, the error in measuring the time for Sensor 1 may have the same root cause as the error in measuring the time for Sensor 2. If this were to happen, their errors might be correlated in an unknown way.
- Items like digitization errors can have an unknown effect on the sensor's ability to measure the time or frequency at which the incoming radiation is measured. These types of errors are normally not included on sensor specification sheets.
- Because there is limited time that can be spent on updating the algorithm, there are limited models and scenario configurations that can be added.

3.2. Limitations

The following limitations arise because of the constraints:

- The number of perturbation calculations the algorithm performs must be limited to produce finite processing times.
- The calculated SNR is mainly used to find the uncertainty in measuring the time or frequency at which the radiation hits the SIGINT sensors using TDOA and FDOA. While it is possible for users to set an additional digitization error, most users probably will not.
- The standard deviation of a Poisson distribution is equal to its mean. For photon counting detectors, the algorithm currently uses a link budget analysis to calculate the mean and therefore, the standard deviation of the photon counting detector measurements. The program is limited such that the calculated mean is assumed to be the correct mean; therefore, that is the only standard deviation Poisson distributions can use.
- There may be times when there are multiple global minimums. For example, there may not be enough equations for the number of unknowns that arise when the algorithm finds the target location. Multiple locations may yield the same

measurements as the perturbed setup, but the optimization algorithm can only return one location. While the algorithm does test for this, there may be scenarios where the algorithm does not successfully detect this possible source of error.

- The effects of multipath uncertainties, signal loss due to the ground or objects in the scenario, or differences in the way the beam will propagate in the near field versus the far field for SIGINT systems are not modelled. The algorithm does not consider the circumstance where an EO/IR angle sensor cannot detect an optical sensor or that the uncertainty in measuring the angle or range to the target might change because of circumstances in the scenario.

3.3. Assumptions

Because of the limitations, the following assumptions are made:

- The distance between each perturbation is small enough to avoid the possibility of hitting a local minimum instead of the global minimum during optimization. Several tests were performed to ensure that the perturbation size is small enough. For example, MUSTC was tested to ensure that a small change in the perturbation size that is used will not have a significant impact on the calculated TLE. A basic validation and verification (V&V) was performed to ensure that the final calculated values make sense. The results of these tests are shown in Appendix D. It is assumed that the precautions taken ensure that the default perturbation size is small enough to produce accurate TLE estimations.
- Each variable is assumed to be independent.
- The standard formula for finding the uncertainty in measuring the time or frequency of the radiation based on the SNR of the sensor is assumed to be valid. If digitization errors are not stated in any system specification, it is assumed that their contributions to the final TLE are negligible.
- The algorithm has checks in place so that the sensors in the scenario would have enough measurements to find a unique location for the target. If the algorithm determines that a unique target location could not be found, the algorithm will set the uncertainty of the measurement at that location to infinity. It is assumed that these checks will catch all instances where there are not enough equations for the unknowns.
- The algorithm assumes that all RF emitters are far enough from the SIGINT sensors that the target can be considered a far field emitter. It also assumes that uncertainties caused by multiple paths to the receiver will not seriously affect the SIGINT sensors' measurements. Finally, the algorithm assumes that attenuation caused by the ground and environment around the RF emitter and SIGINT

sensors will not seriously reduce the intensity of the radiation the sensors measure.

- While different optical targets may be easier to find than others, it is assumed that all EO/IR angle sensors can find all optical targets and measure the angle and range to all targets with the same accuracy.
- Currently we must assume that the models included in the algorithm can do a reasonable job of estimating the performance of the sensors. However, the MUSTC program is designed to be highly modular and thus can easily be updated in the future. As time allows, future additions and updates can be made to model everything that could affect the performance of photon counting detectors, SIGINT sensors, or EO/IR sensors with LRF. Appendix A outlines how to add new sensor types to the MUSTC software.

4. RESULTS AND DISCUSSION

4.1. Sample Results

The V&V section (Appendix D) outlines how the program successfully calculates the TLE values for an object of interest for several simple scenarios. The tests in Appendix D were used to confirm that the MUSTC program returns the expected values for all test scenarios. While these scenarios are useful for V&V purposes, they are not realistic and do not match the type of calculations that the algorithm would produce during real-world calculations.

This section describes how the MUSTC program was able to find the TLE for an object in an unusual, but still realistic, scenario. Most scenarios that involve SIGINT sensors have sensors whose locations are known with an emitter whose location is not. The MUSTC program can be used for this type of scenario. However, the program is so adaptable it can also be used in a scenario where the locations of the emitters are known but the locations of the sensors are not. That is the type of scenario that will be detailed here.

The scenario involves two SIGINT sensors using TDOA. The first sensor (SIGINT Sensor 1) is located at X = 0 and Y = 0. The location of SIGINT Sensor 1 is assumed to have been determined by two EO/IR with LRF sensors. The first EO/IR with LRF is at X = 5 m and Y = –10 m, and the second EO/IR with LRF is at X = 0 m and Y = –20 m. The location of both EO/IR with LRF sensors was determined ahead of time with an uncertainty of 1.5 m in both the X- and Y-directions. The EO/IR with LRF sensors can find the azimuth to an optical target with an accuracy of 10 mrad and the range to a target with an accuracy of 5 m.

The location of SIGINT Sensor 1 was found using other sensors; therefore, before the program can find the TLE for the final target of interest, the MUSTC program must perform calculations to determine the TLE for the location of SIGINT Sensor 1. Once the MUSTC program finds a TLE for SIGINT Sensor 1 using the EO/IR with LRF sensors, these TLE values will automatically be used as uncertainties in the location of SIGINT Sensor 1 for the final TLE calculation.

The asset carrying the second SIGINT sensor (SIGINT Sensor 2) is the object of interest and the location of the asset is unknown. The MUSTC program moves the object of interest to multiple locations and then calculates the final TLE for the asset at each location. In this way, the MUSTC program calculates the TLE as a function of location for the object of interest (the asset with SIGINT Sensor 2).

Even though the location of the second SIGINT system is unknown, it is assumed that the two SIGINT systems can communicate with each other and can measure the additional time it takes for RF radiation to hit one SIGINT sensor and then another; therefore, they are able to use TDOA. The antenna gain for each SIGINT system is assumed to be 0 dB, the system loss is set to 0 dB, the bandwidth is 100 Hz, integration time is 10 s, receiver sensitivity is –70 dB, and the reference SNR is set to 13 dB.

To find the location of SIGINT Sensor 2, some RF emitters with known locations are needed. The first two RF emitters whose locations are known are assumed to be domestic radio stations. The first radio station (Radio Station 1) has a tower height of 500 m and is located at X = –3 km and Y = 5 km. Although the location of the radio station is known, because of vibrations in the tower, wind, and so forth, the location uncertainty is set to 1 cm in all directions. The station frequency is 800 kHz, it outputs 20 kW of RF radiation, has an antenna gain of 6 dB, and a system loss of 0 dB.

This second RF emitter will be Radio Station 2. It will be located at X = 2 km, Y = –4 km, and will have a 600 m tower. The uncertainty of the location in the X- and Y-directions for this tower will be set to 3 cm; the uncertainty in the tower's height will be 2 cm. This station will emit radiation at 100 MHz, the power will be 1 kW, the antenna gain is 3 dB, and the system loss is again assumed to be 0 dB.

Even though this is a 2-D problem (the asset of interest with SIGINT Sensor 2 is assumed to be on the ground), a third emitter is needed to avoid possible location ambiguities. The third emitter will be a portable radio named "Portable Radio." Portable Radio will output 60 W of radiation at 1 GHz and both the antenna gain and system loss are assumed to be 0 dB. The nominal location of Portable Radio will be X = –20 m and Y = –100 m. However, as with SIGINT Sensor 1, the location of the radio will be determined using the two EO/IR with LRF sensors. The MUSTC program must find the TLE for the location of Portable Radio before it can find the final TLE for the asset of interest (SIGINT Sensor 2).

Figure 7 shows the relative locations of all assets in this sample experiment.

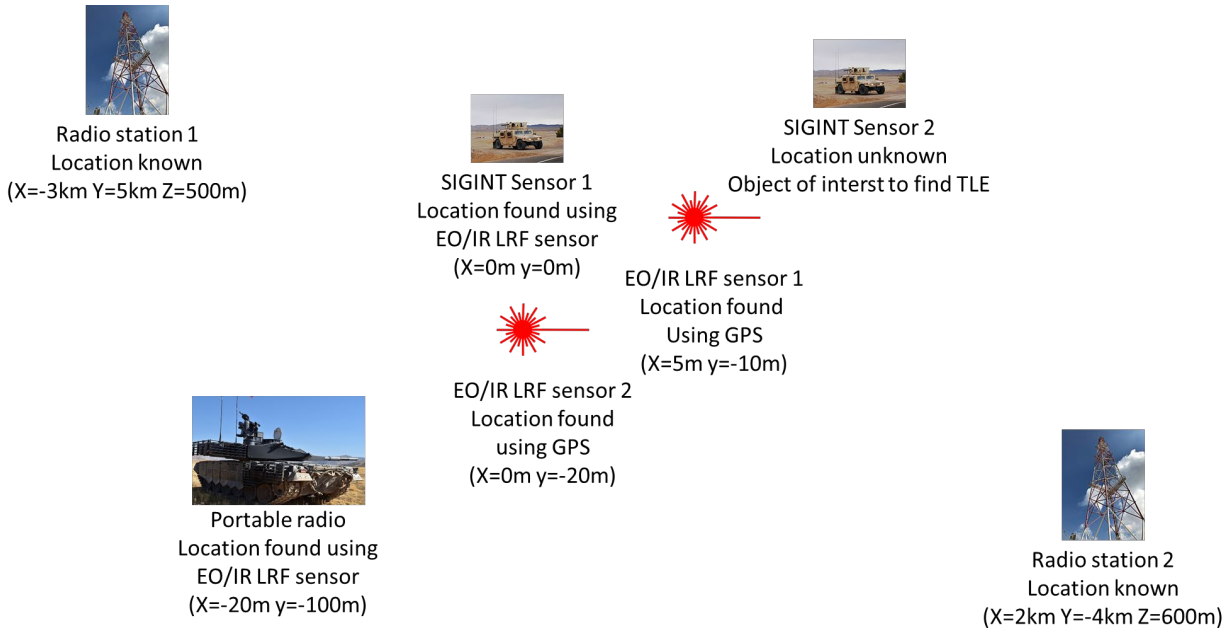

Figure 7. Assets used for the sample experiment

Now that the parameters for the test experiment have been established, they are fed into the MUSTC program to find the 2-D TLE as a function of location. It is 2-D because SIGINT Sensor 2 is assumed to be on the ground; if it were on an aircraft the MUSTC program would find the 3-D TLE.

Figure 8 shows the results of this experiment.

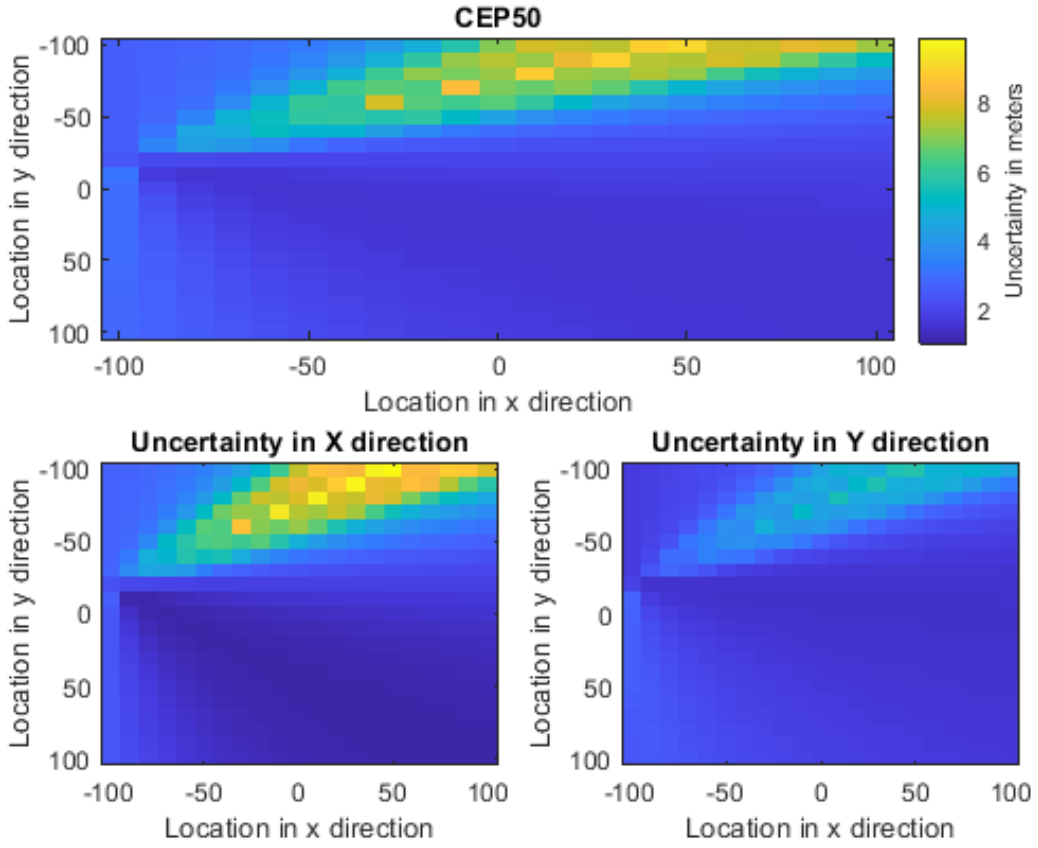

Figure 8. TLE in the X- and Y-directions along with CEP50 for the sample experiment at several locations

This experiment setup is somewhat complicated; therefore, it can be difficult to estimate what the TLE as a function of location should be. However, based on the uncertainties that were fed into the experiment, the order of magnitude of the final TLE values, between approximately 0–10 m, seems reasonable. The scenario just described is closer to a real-world scenario than the ones tested in Appendix D.

4.2. Discussion

DAC has developed an algorithm that estimates the TLE as a function of location for a wide variety of assets of interest using a variety of sensors and techniques to find the asset's location. The algorithm can be expanded to include almost any source of error that could affect the ability of a set of sensors or systems to find the location of an object of interest.

Because the technique that DAC's MUSTC algorithm uses is fundamentally different from most combat simulation algorithms, it may not be practical to directly use the MUSTC DLL with combat simulation software. However, Section 2.6 demonstrates how

the basic algorithm can be modified so it can be directly added into various combat simulation programs. Even if combat simulation programs are not modified to directly use the MUSTC algorithm, they can be modified to ingest the output of the MUSTC program and to then use this output to inform how various sensors and systems can perform when determining location.

The main disadvantage to this algorithm is that it is computationally intensive and therefore time consuming. To address this, Appendix C outlines efforts to speed up computation times. The use of faster computers will further reduce computation times. Individual TLE calculations do not depend on each other so they can be completed in parallel; making the process multi-threaded (enabling the use of multiple processors and computers at once) dramatically sped up the average computation time. In addition, the parameters of the program can be updated to decrease computation time but at the expense of some accuracy.

5. CONCLUSION AND RECOMMENDATIONS

DAC developed a novel TLE algorithm. The inputs to the algorithm are the location of targets and sensors installed on the reference assets along with the sensor(s) and/or target(s) installed on the object of interest whose location is to be determined. The algorithm will step through each of the supplied sources of error and then determine a final TLE for the object of interest as a function of location. The algorithm can be expanded to investigate almost any source of error that can be encountered when trying to find a location.

The main disadvantage to this algorithm is that it can be computationally intensive and therefore slow. However, efforts have been taken to make the program work as fast as possible. The program can be forced to work even faster by modifying the program parameters to favor speed over some degree of accuracy. In addition, since the different TLE calculations are independent, multiple TLE calculations can be performed at the same time in parallel.

The flexibility of the algorithm means that it has a variety of uses. DAC can use this algorithm to augment data sent to combat simulation models and to inform the DOD when decision makers are choosing optimal PNT sensors and systems for the U.S. Army of the future.

6. REFERENCES

Poisel, R. A. (2005). *Electronic warfare target location methods*. Artech House.

Appendix A – Using the Multipurpose Universal Simplified Target Location Error (TLE) Calculator Program's Dynamic Linked Library

A C# Dynamic Linked Library (DLL) project called MultiPurposeTLEDll was written to implement the Multipurpose Universal Simplified TLE Calculator (MUSTC) algorithm. Users can reference this DLL in their visual studio projects so they can have these target location error (TLE) calculations available to their programs.

To use the DLL to perform TLE calculations, users will need to create a new instance of the TLECalculator class. Users will then be able to use one of the following methods of the TLECalculator class to calculate one or more TLE value:

- public TleResult getOneTleResults(double tlocX, double tlocY, double? tlocZ, Scenario scenario)
- public TleResult getOneTleResults(OneLocation location, Scenario scenario)
- public TleResult[] calculateAllResults(OneLocation[] locations, Scenario[] scenarios)
- public List<TleResult> calculateAllResultsList(List<OneLocation> locations, List<Scenario> scenarios).

As seen in the previous methods, to calculate one or more TLE result, users will need one or more instances of the Scenario class to configure the targets and sensors for the scenario. Users may also need one or more instance of the OneLocation class to tell the program where to calculate the TLE values (unless they explicitly tell the system where to calculate a single TLE value using the tlocX, tlocY, and tlocZ variables).

The OneLocation class has the following members:

```
public class OneLocation
  {
     public double locX_meters;
     public double locY_meters;
     public double? locZ_meters;
       }.
```

To find the TLE for an asset at one or more locations, a new instance of OneLocation for each location must be created. The location in the x-, y-, and possibly z-direction

must be set by hand, or one of the methods that are included with the OneLocation class can be called to have the program automatically set these values.

The Scenario class has the following members:

```
public class Scenario
   {
      /// <summary>
      /// Asset whose location uncertainty will translate into the final calculated TLE
      /// </summary>
      public Asset itemToCalcTle;
      /// <summary>
      /// Other assets with sensors or targets that are being used to find the location of the TLE asset
      /// </summary>
      public Asset[] otherTargetsAndSensors = null;
   }.
```

The Scenario class has one member of the Asset class that is used to designate the object whose TLE is being calculated. The Scenario class also has an array that houses all reference assets being used to find the location of the object of interest. These values can be set by hand; however, the Scenario class also includes many methods that will make it easier for users to set these values. Even if these methods are used, new instances of the Asset class must be created to inform the class as to the assets in the scenario. The Asset class has the following members:

```
public class Asset
   {
      public Location location;
      public Velocity velocity;
```

```
    public List<ATarget> targets;

    public List<ASensor> sensors;

}.
```

When defining the asset that will be used to find the TLE, the Location member in the Asset class can be set to null. For all other instances, the location and/or the velocity of the asset's platform must be set. The Location and Velocity classes have the following members:

```
public class Location

{

    public double x_meters;

    public double y_meters;

    public double? z_meters;

    public double? xUncertainty_meters;

    public double? yUncertainty_meters;

    public double? zUncertainty_meters;

    public bool locationNotTLEbutCanStillMove;

    public bool? uncertaintyDeterminedByIniTleCalc;

    }
public class Velocity

{

    public double xVel_mpers;

    public double yVel_mpers;

    public double? zVel_mpers;
```

```
    public double? xVelUncert_mpers;

    public double? yVelUncert_mpers;

    public double? zVelUncert_mpers;

    }.
```

For both the Location and Velocity classes, the location and velocity of the platform is in 2-D or 3-D as are the uncertainty of each.

For the Location class, the locationNotTLEbutCanStillMove member is used to specify that in this scenario, this asset's platform will move to its optimal location depending on how the other assets are arranged. The uncertaintyDeterminedByIniTleCalc member in the Location class can be set to specify that the uncertainty in the location of this platform will be set by another TLE calculation.

Before estimating the location of the primary asset, it may be necessary to determine the location of other asset(s) that have targets and/or sensors that are used in the final measurement. By setting the uncertaintyDeterminedByIniTleCalc member in the Location class to true, the program will automatically perform an initial TLE calculation to find the uncertainty in the location of these intermediate assets. The program will carry this uncertainty over when finding the final TLE of the primary object. If some targets and/or sensors are only used for these initial TLE calculations and not for the final TLE calculation, then inclusion numbers can be used to specify which targets and/or sensors will be used in each part of the experiment. More discussion about inclusion numbers appears in the targets and sensors section.

Once the location and/or velocity of each asset's platform has been set, the target(s) and/or sensor(s) that are on each asset must be configured. Assets have members named targets and sensors that are lists of objects of the ASensor and ATarget abstract classes. As before, these lists can be set by hand or by using methods in the Asset class to add new targets and sensors to the asset. Regardless of how targets or sensors are added to an asset, instances of classes that extend the ASensor and ATarget abstract classes must be defined. The ASensor and ATarget abstract classes have the following members:

```
public abstract class ASensor

  {

    public int? maxTargetInclusionNumberSensorCanDetect = null;
```

```
        public int? minTargetInclusionNumberSensorCanDetect = null;
        public int? sensorInclusionNumberCanCompareDiff = null;
        public int? minSensorInclusionNumberCanCompareDiff = null;
        public int? maxSensorInclusionNumberCanCompareDiff = null;
        public double measurementWeight = 1;
        }
public abstract class ATarget
   {
      public int? targetInclusionNumber = null;
      }
```

The ASensor class has the measurementWeight member, which by default is set to 1. If one sensor has a much higher accuracy than all other sensors in the scenario, it is possible to weigh this sensor's measurements so that its value takes precedence over the values other sensors may measure. For example, to set a sensor's measurement to have 10× more influence over the final location calculation than any other sensor in the scenario, set this sensor's measurementWeight member value to 10. Similarly, if a sensor has a far lower accuracy than any other sensor's measurement, this sensor's measurements can be set to have one 10th the influence on the final location measurement as any other sensor's influence in the scenario by setting the sensor's measurementWeight member value to 0.1.

The ATarget class has the targetInclusionNumber member. When this member is set to null, the inclusion number will not influence whether a sensor can detect a target. However, setting the targetInclusionNumber member of the ATarget class along with the maxTargetInclusionNumberSensorCanDetect and/or minTargetInclusionNumber SensorCanDetect member of the ASensor class prohibits the sensor from detecting the target when the target's inclusion number is outside of the range defined by the sensor, even if the sensor would normally be able to detect the target.

Some sensors, such as time difference of arrival (TDOA) and frequency difference of arrival (FDOA) Signals Intelligence (SIGINT) sensors, only work if the results of one sensor are subtracted from another. For example, TDOA sensors measure the additional time it took for the radiation to go from one sensor to another; they typically

cannot measure the absolute amount of time it takes for the radiation to go from the emitter to one of the sensors. These sensors need to be paired together to take measurements that make sense. There are situations where multiple TDOA or FDOA sensors could be paired together but the scenario dictates that they cannot. For example, when sensors cannot communicate with each other so they cannot compare their measurements. This is when the sensorInclusionNumberCanCompareDiff, minSensorInclusionNumberCanCompareDiff, and maxSensorInclusion NumberCanCompareDiff members are used; like the targetInclusionNumber member, these members can be used to explicitly say that one sensor cannot be paired with another.

Although it is important to understand the members of the ATarget and ASensor abstract classes, to define the scenario to make TLE measurements, classes that extend the ATarget and ASensor abstract classes are needed to allow for simulation of real sensors and targets. The following classes extend the ATarget class to allow for target definition in the DLL:

```
public class OpticalTarget : ATarget

{

}

public class LightSourceOmnidirectional : ATarget

{

    public double wavelengthNanoMeters;

    public double? optionalUncertaintyInPowerWatts;

    public double powerWatts;

}

public class RFemitter : ATarget

{

    public double powerTransmittedWatts;

    public double transmitterAntennaGainDb;
```

```
    public double transmitterSignalLossDB;

    public double frequencyHz;

    public double? optionalMinFreqHz;

    public double? optionalMaxFreqHz;

    public RFemitterType? optionalEmitterType;

}
```

If the scenario includes an optical angle sensor, one or more optical targets using the OpticalTarget class must be defined. There are currently no members in the OpticalTarget class; therefore, the assumption is that any optical angle sensor can detect any optical target with the same accuracy and reliability.

Likewise, if the scenario includes a photon counting detector, one or more LightSourceOmnidirectional classes must be defined. To create such a class, the power and wavelength of the emitter via the powerWatts and wavelengthNanoMeters variables must be supplied. The uncertainty in the power can be set using the optionalUncertaintyInPowerWatts variable.

Finally, if the scenario includes RF emitters that will be detected by SIGINT sensors, one or more instances of the RFemitter class must be defined. Since there are scenarios where not all SIGINT sensors can detect all RF emitters, the program can check to see if a SIGINT system can detect an RF emitter by using the optionalMinFreqHz, optionalMaxFreqHz, and optionalEmitterType members. By setting the optionalMinFreqHz and optionalMaxFreqHz members of the RFemitter class as well as the range of frequencies that the sensor will detect in their classes, the program will confirm that the sensor and target frequencies are in the correct range to be detected.

Electronic Intelligence (ELINT) SIGINT sensors are designed to detect emitters such as radar systems and Communications Intelligence (COMINT) sensors are designed to detect things like radios. The optionalEmitterType member is used to specify the type of emitter that each target is to ensure that ELINT systems are not detecting radios. Setting the values of all members to null allows all RF emitter types to be detected by all SIGINT sensors.

Unlike optical sensors, the uncertainty in the measurements that SIGINT systems make depends on the signal-to-noise ratio (SNR) of the SIGINT sensor. Because of this, RF emitters have members that must be defined for the program to estimate the SIGINT

system's measurement SNR. The members are as follows: powerTransmittedWatts, transmitterAntennaGainDb, transmitterSignalLossDB and frequencyHz.

The following classes extend the ASensor abstract class:

```
public class OpticalAngleSensor: ASensor

   {

      public double azimuthUncertRadians;

      public double? optionalElevationUncertRadians = null;

      public double? optionalRangeUncertaintyMeters = null;

       }


public class PhotonCountingIntensitySensor : ASensor
{
      public double? optionalMinWavelengthNanometers = null;
      public double? optionalMaxWavelengthNanometers = null;

      public double visibilityMiles;
      public double? optionalUncertaintyVisibilityMiles = null;

      public double backgroundRadiationWatts;
      public double? optionalUncertaintyBackgroundRadiationWatts = null;

      public double darkCurrentHertz;
      public double? optionalDarkCurrentUncertaintyHertz = null;

      public double systemOpticalTransmittance;

      public double sensorAreaCmSq;

      public double photonDetectionEfficiencyPercent;
      public double? optionalUncertPhotonDetEffPercent = null;

      public double integrationTimeSeconds;

      public double? optionalAzimuthAngleSensorPointedDegrees = null;
      public double? optionalAzimuthPointedUncertaintyDegrees = null;
      public double? optionalElevationAngleSensorPointedDegrees = null;

      public double? optionalElevationPointedUncertaintyDegrees = null;
```

```
}
public class RFangleSensor : ArfSensor
   {
      public double azimuthUncertRadians;
      public double? optionalElevationUncertRadians = null;
      }
public class RFsensorFDoA : A_TDOA_FDOA_RfSensor
   {
   }
public class RFsensorTDoA : A_TDOA_FDOA_RfSensor
   {
       }
```

The RFangleSensor, RFsensorFDoA, and RFsensorTDoA classes all extend the ArfSensor class. The members for this abstract class are as follows:

```
public abstract class ArfSensor : ASensor
   {
      public double? optionalMinFreqHz;
      public double? optionalMaxFreqHz;
      public RFsensorType? optionalSensorType;
    }
```

In addition to the ArfSensor abstract class, the RFsensorFDoA and RFsensorTDoA classes also extend the A_TDOA_FDOA_RfSensor abstract class. The members of the A_TDOA_FDOA_RfSensor abstract class are as follows:

```
   public abstract class A_TDOA_FDOA_RfSensor : ArfSensor
```

```
    {
        public double? optionalNoiseFigure = null;
        public double? optionalReferenceSNRinDB = null;
        public double? optionalReceiverSensitivityDBm = null;
        public double receiverAntennaGainDB;
        public double receiverSystemLossDB;
        public double bandwidthHz;
        public double integrationTimeSec;
        public double? optionalDigitizationError = null;
}
```

Both the optical angle sensor and the angle of arrival (AOA) SIGINT sensors can measure the azimuth to the target. Therefore, both the OpticalAngleSensor and RFangleSensor classes have an azimuthUncertRadians member that allows the uncertainty to be set in measuring the azimuth to the target. In addition, some AOA SIGINT and optical sensors can also measure the elevation to the target; therefore, both classes also have an optional optionalElevationUncertRadians member. When this member is set to null, the sensors cannot measure the elevation to the target. However, when they can measure the elevation angle to the target, this member is used to set the uncertainty in measuring this angle. Finally, many optical angle systems have a laser range finder (LRF) that can measure the range to the target. Therefore, the OpticalAngleSensor class has an optionalRangeUncertaintyMeters member that can either be set to null if the sensor has no LRF or to a value that represents the uncertainty in measuring this range.

Unlike optical angle sensors, it is not assumed that all SIGINT sensors can detect all RF emitters. All RF sensors therefore extend the ArfSensor abstract class. This class has the optionalMinFreqHz, optionalMaxFreqHz, and optionalSensorType members that determine the maximum and minimum frequencies that the sensor can detect along with whether the sensor is an ELINT or COMINT sensor; all of these values are optional. Setting each to null allows the SIGINT sensor to detect any RF emitter in the scenario.

Because uncertainty in measuring the time and frequency for TDOA and FDOA SIGINT systems depends on the SNR of the incoming RF radiation measurements, the RFsensorFDoA and RFsensorTDoA classes need to estimate the SNR of their measurement before they determine the uncertainty in their time and frequency measurements. These classes extend the A_TDOA_FDOA_RfSensor abstract class that calculates the SNR for both TDOA and FDOA sensors. The A_TDOA_FDOA_RfSensor class has the following members that are used to estimate noise when finding the SNR of the radiation measurement: optionalNoiseFigure, optionalReferenceSNRinDB, and optionalReceiverSensitivityDBm. Many specification sheets for SIGINT systems will include the receiver sensitivity for the sensor; if this value is known and the reference SNR is supplied, the program will use these values to estimate the noise of the incoming RF radiation measurement. If the receiver sensitivity is not available, the system's noise figure must be supplied so that the algorithm can estimate the noise in the SNR. To calculate the signal for the SNR estimation, the A_TDOA_FDOA_RfSensor class has the following members: receiverAntennaGainDB, receiverSystemLossDB, bandwidthHz, and integrationTimeSec. When the program has all values, it can determine the SNR and the uncertainty in measuring the time or frequency for TDOA and FDOA sensors. The A_TDOA_FDOA_RfSensor class also has an optionalDigitizationError member. Digitization errors happen when the analog-to-digital conversion circuit the sensor is using does not have enough bits to measure the desired value with enough precision/accuracy. This variable allows for modeling the uncertainty in measuring time and frequency based on the digitization error. Nothing happens when this variable is set to null.

The RFsensorFDoA and RFsensorTDoA classes do not have any members of their own, but once values are established for all the abstract classes that these extend to, there will be enough information to create new instances of these classes. Some SIGINT systems will be able to make angle, frequency, and time measurements at the same time; however, to model such a sensor using this program requires defining an asset that has three separate sensors installed.

Photon counting detectors have many variables that could affect the system's measurement. Using the optionalMinWavelengthNanometers and optionalMaxWavelengthNanometers members of the PhotonCountingIntensitySensor class, it is possible to set the wavelength of the emitters that the sensor can detect. Once the system determines that a photon counting detector can detect an emitter, the program uses the power and wavelength of the emitter along with the following variables to help determine the magnitude of the measurement: visibilityMiles, backgroundRadiationWatts, darkCurrentHertz, systemOpticalTransmittance, sensorAreaCmSq, photonDetectionEfficiencyPercent, and integrationTimeSeconds.

The direction the detector is pointed to relative to the axis used to define locations in the scenario can be set using these variables: optionalAzimuthAngleSensorPointedDegrees and optionalElevationAngleSensorPointedDegrees. If these values are null the detector will always be pointed to the emitter.

To explore how uncertainties in the amount of power the emitter will emit will lead to errors in the location, the following members of the PhotonCountingIntensitySensor class can be set: optionalUncertaintyVisibilityMiles, optionalUncertaintyBackground RadiationWatts, optionalDarkCurrentUncertaintyHertz, and optionalUncertPhoton DetEffPercent. Uncertainties in the photon detector's direction are set using the following members: optionalAzimuthPointedUncertaintyDegrees and optionalElevationPointedUncertaintyDegrees. If the uncertainty in the angles is given but the angle itself is not, then the program assumes that the operator is pointing the emitter towards the target but that their aim is off by the given amount.

There are now enough classes and information to get the program to estimate TLE values for scenarios that involve SIGINT, photon counting, and electro-optical/infrared (EO/IR) with LRF sensors. The next section discusses how to add another type of target or sensor to the program.

Adding Another Type of Target or Sensor to the DLL

Although the software that implements the algorithm is currently only able to find the TLE for location measurements made with SIGINT systems, photon counting detectors, and/or EO/IR angle sensors with LRF, the software was designed with maximum modularity in mind; therefore, it is easy to modify the program to allow it to find the TLE for another type of location sensor. To add another type of target and sensor, the ATarget and ASensor abstract classes must be extended. Although the ATarget class is defined as being abstract, there are no abstract methods that need to be overridden when extending it; however, it is necessary to ensure that the sensor and targets work together when defining the sensor. There is a virtual method in the ATarget class that may need to be overridden:

```
internal virtual List<ABasePertCalcs> getAllPertCalcs()
```

This function returns all perturbations, or variables that may affect the final TLE, for this target. The uncertainty in the power emitted by an emitter that will be detected using a photon counting detector is an example of a perturbation on a target that may affect the sensor's measurement. Most targets will not have perturbations; therefore, leaving this method to its default value is acceptable. Perturbations will be discussed in detail later.

When creating new sensors, the ASensor abstract class must be extended. The abstract methods that need to be implemented when extending the ASensor abstract class are as follows:

```
internal abstract double[] getDirectMeasurementValues(LocVelForCalc thisLocation,
        LocVelForCalc targetLocation, ATarget target, bool is2dProblem);
internal abstract double[] getDirectMeasurementValues(LocVelForCalc thisLocation,
        LocVelForCalc targetLocation, ATarget target, bool is2dProblem,
        ABasePertCalcs targetPert, double tarPertAmount,
        ABasePertCalcs sensorPert, double senPertAmount);
internal abstract bool canOnlyMeasureDifferenceOfArrivalFromSensors();
internal abstract bool canSeeTarget(ATarget target);
internal abstract bool canHandle3dScenarios();
internal abstract int numberOfValues(bool is2dProblem);
internal abstract List<ABasePertCalcs> getAllPertCalcs(LocVelForCalc sensorLoc,
LocVelForCalc targetLoc, ATarget target);
```

The first method to override is the getDirectMeasurementValues method. The parameters that are passed to this function are the locations of the target and the sensor (identified as “this location”), the target the sensor is sensing, and whether this is a 2-D problem. In another instance of the getDirectMeasurementValues method the target and sensor perturbations are also passed as parameters. The return value is an array of measurements this sensor will make on this target.

The next method to override to create a new sensor class is the canOnlyMeasure DifferenceOfArrivalFromSensors method. There are some values that a sensor cannot measure directly; for example, SIGINT systems cannot directly measure the time it takes for radiation to travel from the target to the sensor. However, it is possible for the SIGINT systems to measure the additional time for the radiation to hit one sensor as opposed to another sensor. When the canOnlyMeasureDifferenceOf ArrivalFromSensors method returns a true value, the software assumes that the sensor system cannot directly use the results of the getDirectMeasurementValues method for a single sensor and will instead pair the measurements from this sensor to the

measurements of another sensor to measure the difference between these values. When the method returns a false value, the software assumes that the system will be able to directly use the numbers from the getDirectMeasurementValues method that each sensor returns. A sensor that is directly measuring the angle to the target will have their getDirectMeasurementValues method set to always return false.

Next up for override is the canSeeTarget method. This determination is based on it being theoretically possible for the sensor to see the target. Thus, this function will return true even when the target is so far away that the signal from it is indistinguishable from noise and it is not practical to take the measurement. It is best practice to ensure that the incoming target is of the expected type. For example, when implementing an optical sensor and the program asks if the sensor can see an RF target the answer should be false. Once ensuring the target type is correct, check whether the target is the right type for the sensor (e.g., ensure that COMINT SIGINT systems are not detecting a radar) and whether the target emits radiation within the frequency band the sensor is looking for.

Not all sensors can be used to detect 3-D. When this is the case, the canHandle3dScenarios method must be overridden to return a false. Likewise, the number of measurements the system will output may change as a function of whether this is a 3-D problem. This may require an override of the numberOfValues method so that the correct number of measurements are returned when the program calls the getDirectMeasurementValues function.

Finally, there is the getAllPertCalcs method, which outputs all perturbations that are defined for this sensor. Uncertainty in sensor measurement could affect a location measurement, and the MUSTC program works by adding perturbations to different variables that might affect the accuracy of a location measurement. The program will add errors or perturbations to what the sensor should measure and then determine how much the system's location measurement will be off because of this perturbation. When there are variables or measurements the sensor or target makes or keeps track of that might affect the final location measurement, this is communicated by overriding the target or sensor's getAllPertCalcs method. The getAllPertCalcs method returns an array of elements that belong to the ABasePertCalcs class. For each variable that could affect the location measurement, a new instance of the ABasePertCalcs class is needed.

While an instance of any class that extends the ABasePertCalcs abstract class can be returned from the method, it is best practice to use the getNewPertCalc static method from the PertCalcFactory class to get a perturbation for a variable. The parameters that this method requires are as follows:

```
internal static ABasePertCalcs getNewPertCalc(DistributionType distroType, object
distroParams, object scenarioInformationIn, UncertaintyType type, OtherVariableType?
otherVarTypeIn, object originObject)
```

The distroType variable will inform the program as to which distribution to use, and distroParams are the parameters that will be sent to the distribution. Currently, there are only two possible distribution types: Poisson and normal. For a normal distribution, distroParams should be a double precision variable that houses the standard deviation of the distribution. When the distroType is Poisson, distroParams should be a long integer with the distribution's lambda value (the mean sensor count). For Poisson distributions, the standard deviation is always the square root of the mean value; if the lambda is sufficiently large, the program might automatically turn the Poisson distribution into normal.

When a sensor has more than one perturbation to keep track of, the scenarioInformationIn object can be set to any value in order to differentiate between perturbations; otherwise, it can be left null. For example, if a sensor can measure both azimuth and elevation angle, and there can be uncertainties in both measurements, the scenarioInformationIn object can be used to inform the sensor as to which angle measurement is currently perturbing. The program may need to track the type of uncertainty this perturbation is when it is stepping through the perturbation values. This is done by setting the type variable to a member of the UncertaintyType enumerator. The valid options for this variable are as follows:

```
public enum UncertaintyType
   {
      location, velocity, directionSensorMeasurement, otherVariable
   }
```

Perturbation type can be defined as the uncertainty in the location or velocity of a reference object, a direct sensor measurement, or some other type of variable that could affect the TLE (e.g., the power emitted by a source whose intensity at a given location is used to estimate the range to the emitter). If the perturbation is a variable other than a direct sensor measurement or an uncertainty in the location or velocity of a reference asset the program needs to know what type of variable it is dealing with. Although more options can be added to the OtherVariableType enumerator, currently the following are valid options:

```
public enum OtherVariableType
   {
      opticalPower, visibility, backgroundOpticalRadiation, opticalSensorDarkCurrent,
opticalPhotonEfficiency, angleUncert, unKnown
```

```
}
```

The originObject parameters house an instance of the target or sensor that created this perturbation. Sensors or targets that create the perturbation information can send instances of themselves for the originObject parameter.

If the program wants the sensor class to add these perturbation values to its measurement values it will call the version of the getDirectMeasurementValues method that has the targetPert, tarPertAmount, sensorPert, and senPertAmount parameters. The targetPert and sensorPert parameters are the instances of the ABasePertCalcs class that were created when the program called the sensor or target's getAllPertCalcs method. The tarPertAmount and senPertAmount parameters are the magnitude of the perturbation that the program is currently asking for.

An example detailing the addition of a 0.1-radian error to the azimuth measurement is explained here. The program will call the getDirectMeasurementValues method with the parameter sensorPert set to the instance of the class that informed the program of this azimuth uncertainty. The senPertAmount parameter will be set to 0.1 and the program expects to add 0.1 radians to the sensor's final azimuth measurement. If the sensor can make multiple angle measurements, then using the scenarioInformationIn object ensures that the error is being added to the correct angle. The program will only add a perturbation to a single variable at a time. The targetPert and/or sensorPert parameter will always be set to null and the values that the program sends the getDirectMeasurementValues function via the tarPertAmount and senPertAmount parameters can be negative or positive.

The program will often use serialization to log the calculations progress or to make clones of a particular instance of a class for multithreading purposes. Therefore, if a new function is defined for use with the program's calculation, it is important to declare the class serializable.

Using the instructions in this section, a C# program can be configured to use the MUSTC DLL to find the TLE for almost any scenario that includes the sensor types that have already been defined. Likewise, a C# programmer can use these instructions to add new sensor types.

Appendix B – Choosing an Optimization Algorithm

The Multipurpose Universal Simplified TLE Calculator (MUSTC) program requires an optimization algorithm to determine how the uncertainty in one position, navigation, and timing (PNT) sensor variable can affect the final target location error (TLE). The literature is full of optimization algorithms that the MUSTC algorithm could use. Optimization algorithms move parameters around (in this case, the parameters are the location of the target of interest) to minimize (or maximize, depending on how they are configured) a cost function. For the MUSTC algorithm, the cost function is defined as the average of all the differences between the sensor measurements at a new test location and the sensor measurements at the original location but with a perturbation added to one of the variables that could affect the TLE; the input parameter for this cost function is simply the new test location of the item of interest.

Optimization algorithms have different techniques for finding a new test location, determining if the new test location is closer or farther away from the final location that minimizes the cost function, and figuring out how to avoid local minima in favor of a global minimum. The algorithms will continue trying new test locations until a final location that minimizes the cost function is found.

Using a simplistic brute-force optimization algorithm that sets up a grid pattern and then finds the cost function at each location in the grid is one way to show that algorithms can find a minimum of the cost function. The upside to this type of algorithm is that it is less likely to find a local minimum over a global minimum. The downside is that the precision of the final location result will be limited by the distance between each point in the grid pattern, and that the algorithm can be extremely time consuming to perform because it requires the system to find the cost function at many locations. Therefore, while this would be a less than ideal algorithm to use for any real-world calculations, it does make for a good test because if this algorithm cannot find a global minimum it is unlikely that any algorithm can. Additionally, if the algorithm works, it adds confidence to the idea that other optimization algorithms can also find a good minimum for the cost function.

Gradient descent is a faster and more advanced optimization algorithm than the brute-force algorithm previously mentioned. The gradient-descent algorithm finds the gradient of the cost function at the initial guess and then multiplies the gradient by a predetermined constant before adding it back to the initial guess. The algorithm identifies this new location as a better guess as to the location that minimizes the cost function and will repeat this process until it finds a location where the gradient is small enough that it assumes that it has reached an inflection point. This algorithm has the advantage of requiring significantly fewer calls to the cost function than the simplistic brute-force algorithm did; however, since the MUSTC program does not have a closed-

form solution to the gradient of the cost function, the algorithm must estimate the gradient numerically. The gradient is estimated numerically by adding a small delta to each location component to see how this change in delta changes the cost function. Thus, finding the gradient still requires the program to find a value of the cost function at many locations. Also, if the constant that the gradient is multiplied by to find the next value is too small it may take a long time to reach the cost function's minimum; conversely, if the constant is too large it might bounce past the minimum and may never find it. Finally, this algorithm does not have any way of ensuring that it finds the global minimum instead of a local minimum.

A popular algorithm that improves on gradient descent is Nelder–Meade Simplex. A simplex is a shape that has one more side than the number of dimensions in the space where the shape is defined. For a 2-D space, a simplex would be a triangle; for a 3-D space, a simplex would be a triangular pyramid with four sides. Instead of a single initial guess, Nelder–Meade Simplex requires enough initial guesses to form a simplex. Thus, for a 2-D problem, the algorithm requires three initial guesses.

Based on the magnitude of the cost function at those initial guesses, the Nelder–Meade Simplex algorithm will determine an additional guess at a point where the cost function should have a smaller magnitude. Using the actual cost function, the algorithm will see if the cost function at this new location does have a smaller magnitude than the cost function at the other locations. If the magnitude is lower, it will replace one of the other guesses to create a new simplex. If it is not lower, it will move in the opposite direction to find another guess that does in fact lower the value of the cost function as expected. As the algorithm works, the simplex will morph and change its shape as it walks down a hill in the cost function until it converges on the minimum. Like gradient descent, Nelder–Meade Simplex can find a local instead of a global minimum. However, Nelder–Meade Simplex is usually faster than gradient descent and is therefore usually preferred.

Additional algorithms are available in the external libraries from Microsoft Visual Studio's NuGet package manager. One of those libraries is the "libOptimization" library by "tomitomi3." This library is distributed under the Massachusetts Institute of Technology (MIT) software license. The MIT software license allows anyone to use this library free of charge, and the library includes a large collection of optimization algorithms that the MUSTC program can try. A series of tests were performed to see which algorithm would work best for this application.

For the first test case, each optimization algorithm was fed a problem where the answer could be easily and independently verified. Normally, the MUSTC algorithm will add one perturbation to one variable that might affect the TLE to see how an error in this variable

can create an error in the location measurement. For example, the MUSTC algorithm will add a perturbation to the location of one of the scenario's reference assets to see how an error in the reference location can lead to an error in the final location measurement. However, we cannot use this type of test to find the correct optimization algorithm to use because it may be difficult to determine the location error that we should get for a perturbation on only one reference location.

When the MUSTC algorithm works in its stock configuration, it can be difficult to determine how a perturbation on one variable can affect the error in the final measurement; however, what would happen if we were to perform a test where we modify the behavior of the algorithm so that several locations were perturbed at once? In particular, what if simultaneous perturbations were added to all locations in the same direction and by the same amount for all assets used to measure the location of the object of interest? Such a configuration would not be of much use when finding the TLE of a target. However, it would make a useful test of the optimization algorithms. This is because the optimal location of the target after all the perturbations have been added can be easily determined ahead of time. When the optimization algorithm moves the location of the object of interest in the same direction and by the same amount as the perturbation that was added to all the other objects, the cost function should be minimized. Figure B-1 illustrates this setup.

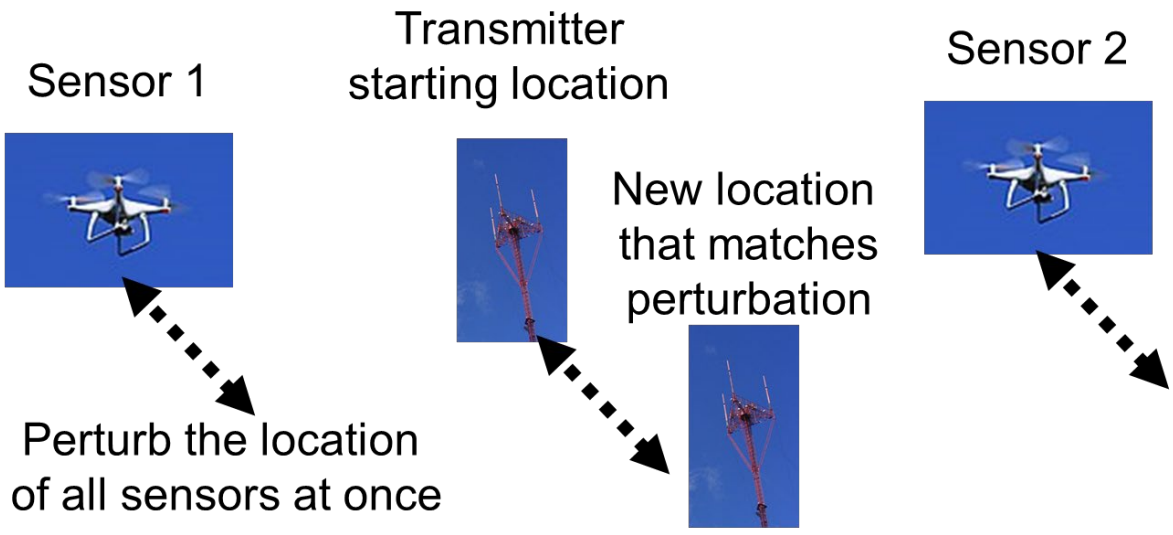

Figure B-1. Experimental setup where the location of each sensor is perturbed by the same amount to see if the location of the sensor moves by the same amount

The initial set of tests examined just the Nelder–Mead Simplex algorithm and the brute-force algorithm. The first test scenario had four Signals Intelligence (SIGINT) sensors flying in formation and were using time difference of arrival (TDOA), frequency difference of arrival (FDOA), and angle of arrival (AOA) at the same time to find the

emitter location. After adding a small perturbation to the location of all the sensors in the formation at once (0.56 m in the X-direction, –0.56 m in the Y-direction, and 0.5 m in the Z-direction), the algorithms were evaluated to see if they correctly determined that the cost function would be minimized when the target location was moved in the same direction by the same amount. The experiment was then repeated with a larger perturbation on all locations (13 m in the X-direction, 33 m in the Y-direction, and 7 m in the Z-direction). Finally, the two experiments were repeating as a 2-D problem. In the 2-D problem, the location of all assets in the scenario (both the sensors and the target itself) were limited so that their location in the Z-direction had to be 0. The results of this initial test case are shown in Table B-1.

In particular, Table B-1 shows the name of the algorithm, the name of the test (near move, far move, in 2- or 3-D), and the error in the algorithm's calculation for each direction. This error is the difference between the algorithm's expected location (a movement from the starting location equal to that which all the other assets were perturbed by) subtracted from the actual location that the algorithm came up with. Note that a far move test for the brute-force algorithm could not be performed as that distance would be outside the grid that it used.

Table B-1. DAC Implemented Optimizations Algorithm Performed on the Initial Algorithm Test Cases

Algorithm	Test Name	Error in the X-Direction (m)	Error in the Y-Direction (m)	Error in the Z-Direction (m)	Time (ms)
NM Simplex	2-D near move	–1.68e-13	8.55e-15	NA	71
NM Simplex	2-D far move	–2.05e-12	2.84e-13	NA	3
NM Simplex	3-D near move	–0.94	0.093	0.12	5
NM Simplex	3-D far move	–4.30	0.56	0.30	7
Brute force	2-D near move	4.00e-13	8.44e-13	NA	439
Brute force	3-D near move	4.00e-13	8.44e-13	–1.07e-14	96,000

As seen in Table B-1, the Nelder–Mead Simplex was fast and performed well for 2-D problems (errors smaller than 10^{-3} are likely due to round off errors and are therefore acceptable); however, the algorithm did poorly when faced with 3-D problems. The brute-force grid algorithm performed well for both 2- and 3-D problems, and the small error resulted from the spacing in the search grid. As expected, the brute-force algorithm was extremely slow and could not be used if the perturbation was outside of the test grid; it was only included in the test case as a proof of concept.

Using the "libOptimization" library, the test in Table B-1 was expanded to include many more optimization algorithms. Table B-2 shows which optimization algorithms from the library were tested. The 2-D near move test was discontinued as it did not yield any additional useful information on the performance of the algorithms.

Table B-2. How the Optimization Algorithms in the "libOptimization" Library Performed in the Initial Algorithm Test Case

Algorithm	Test Name	Error in the X-Direction (m)	Error in the Y-Direction (m)	Error in the Z-Direction (m)	Time (ms)
CS	2-D far move	0	0	NA	16,000
CS	3-D near move	–5.46e-14	–5.66e-15	2.31e-12	16,000
CS	3-D far move	2.27e-13	–5.66e-15	2.31e-12	16,000
DE	2-D far move	0	0	NA	410
DE	3-D near move	5.46e-14	–6.66e-15	–7.57e-13	660
DE	3-D far move	0	0	–3.39e-12	630
DEJADE	2-D far move	0	0	NA	1,400
DEJADE	3-D near move	–1.68e-13	8.55e-15	1.82e-12	72,000
DEJADE	3-D far move	0	0	–2.15e-12	2,300
ES	2-D far move	43.1	–5.66	NA	600
ES	3-D near move	8.49	–0.843	4.89	600
ES	3-D far move	–27.7	3.64	–12.2	600
FA	2-D far move	3.21e-5	–1.35e-5	NA	89,000
FA	3-D near move	–0.0072	0.00082	0.0011	82,000
FA	3-D far move	0.0037	–0.00050	–0.0071	77,000
hillClimbing	2-D far move	7.65e-4	–1.1e-4	NA	5,800
hillClimbing	3-D near move	4.34e-3	–5.22e-4	–0.0100	5,800
hillClimbing	3-D far move	–0.0165	0.0022	–0.00045	6,000
nmSimplex	2-D far move	13.1	–1.724	NA	4
nmSimplex	3-D near move	–1.57	0.156	–0.479	3
nmSimplex	3-D far move	65.4	–8.59	–10.54	4
nmSimplexWiki	2-D far move	1.61e-11	–2.04e-12	NA	8
nmSimplexWiki	3-D near move	2.195e-6	–1.95e-7	–3.20	5
nmSimplexWiki	3-D far move	–25.9	3.40	–49.8	8
patternSearch	2-D near move	–0.503	0.0500	NA	4
patternSearch	3-D near move	–0.503	0.0500	3.71e-3	7
patternSearch	3-D far move	13.7	–1.80	–0.26	9
PSO	2-D far move	0	0	NA	1,200
PSO	3-D near move	–1.68e-13	8.55e-15	1.50e-12	1,200
PSO	3-D far move	2.27e-13	–2.84e-14	–2.45e-13	1,200

Table B-2. How the Optimization Algorithms in the "libOptimization" Library Performed in the Initial Algorithm Test Case

Algorithm	Test Name	Error in the X-Direction (m)	Error in the Y-Direction (m)	Error in the Z-Direction (m)	Time (ms)
PSOAIW	3-D near move	–5.46e-14	–5.66e-15	1.46e-12	300
PSOAIW	3-D far move	2.27e-13	–2.84e-14	–4.44e-12	800
PSOChaoticIW	2-D far move	0	0	NA	1,800
PSOChaoticIW	3-D near move	–5.56e-14	–5.66e-15	–5.42e-13	1,900
PSOChaoticIW	3-D far move	0	0	–3.56e-12	1,600
PSOCPSO	2-D far move	0	0	NA	1,500
PSOCPSO	3-D near move	–5.46e-14	–5.66e-15	–2.13e-13	1,300
PSOCPSO	3-D far move	0	0	–6.12e-13	1,400
PSOLDIW	2-D far move	0	0	NA	5,400
PSOLDIW	3-D near move	–1.68e-13	8,55e-15	2.20e-12	5,000
PSOLDIW	3-D far move	0	0	–3.93e-12	5,400
RealGABLX	2-D far move	–9.09e-13	1.14e-13	NA	14,000
RealGABLX	3-D near move	–1.68e-13	8.55e-15	2.00e-12	12,000
RealGABLX	3-D far move	2.27e-13	–2.84e-14	–2.88e-12	17,000
RealGAREX,	2-D far move	0	0	NA	17,000
RealGAREX,	3-D near move	–5.46e-14	–5.66e-15	4.28e-13	24,000
RealGAREX,	3-D far move	1.02e-12	1.42e-13	–3.80e-12	24,000
RealGASPX	2-D far move	–9.09e-13	1.14e-13	NA	850
RealGASPX	3-D near move	–5.46e-14	–5.66e-15	3.52e-13	1,000
RealGASPX	3-D far move	0	0	–2.77e-12	1,300
RealGAUNDX	2-D far move	–2.066e + 140	2.78e + 139	NA	1,400
RealGAUNDX	3-D near move	-682	74.6	150	14,000
RealGAUNDX	3-D far move	–1.65e + 70	2.22e + 69	–1.96e + 69	15,000
SimulatedAnnealing	2-D far move	13.8	-13.4	NA	402
SimulatedAnnealing	3-D near move	–0.227	0.0231	–0.0117	420
SimulatedAnnealing	3-D far move	14.7	–13.6	6.90	410
SteepestDescent	2-D far move	5.82	–0.729	NA	440
SteepestDescent	3-D near move	–0.133	–0.0889	0.0358	664
SteepestDescent	3-D far move	7.29	–0.99	–0.15	664
Template	2-D far move	16.4	–28.2	NA	100
Template	3-D near move	–3.03	0.240	–0.169	100
Template	3-D far move	15.0	–28.2	5.55	100

As seen in Table B-2, there are several optimization algorithms in the “libOptimization” library that performed well in this test. In particular, the following algorithms did a good job at finding the correct location of the target that minimized the cost function: CS, DE, DEJADE, PSO, PSOChaoticIW, PSOAIW, PSOCPSO, PSOLDIW, RealGABLX, RealGAREX, and RealGASPX. The FA and hillClimbing algorithms also did a reasonable job although the accuracy was not as high as it was for some of the other algorithms. The remaining optimization algorithms did not perform well for this application although they no doubt perform well for other applications.

While the first test case did a good job of determining which algorithms work well for this application and which do not, it does not show how well these algorithms will perform when the MUSTC program is tasking them with performing real-world calculations. Thus, a second test was performed where the MUSTC algorithm uses the different optimization algorithms to find a single TLE measurement for a more realistic scenario.

The scenario was similar to the previous scenario. However, instead of adding perturbations to the locations of all sensor locations at once, the algorithm went back to its default configuration of only perturbing one variable at a time. To ensure that each perturbation will have a larger effect on the final TLE, the number sensor assets were reduced to two. The sensors were configured to use TDOA, AOA, and FDOA SIGINT measurements as well as electro-optical/infrared (EO/IR) with laser range finder (LRF) measurements to find the TLE of a single object of interest. The object of interest has both an RF emitter and an optical target installed. Table B-3 shows the TLE in the X-, Y-, and Z-direction that the algorithm calculated along with the time it took for the MUSTC algorithm to complete all of its calculations.

Table B-3. Results of a Test Designed to Demonstrate How Different Optimization Algorithms Will Perform in Real-World MUSTC Calculations

Algorithm Name	Time (ms)	X-Direction Uncertainty (m)	Y-Direction Uncertainty (m)	Z-Direction Uncertainty (m)
CS	8,000,000	0.0953	0.0953	0.0500
DE	940,000	0.0953	0.0953	0.0500
DEJADE	3,500,000	0.0953	0.0953	0.0500
ES	1,700,000	4.13	4.17	0.0579
hillClimbing	15,000,000	0.0870	0.0975	0.00988
nmSimplex	19,000	5.60	6.13	1.35
PSO	1,100,000	0.0953	0.0953	0.0500
PSOAIW	450,000	0.0850	0.0999	0.0500
PSOChaoticIW	1,300,000	0.0953	0.0953	0.0500
PSOCPSO	1,100,000	0.0953	0.0953	0.0500
RealGASPX	1,700,000	0.0953	0.0953	0.0500

Although it is not easy to independently determine the TLE that the algorithms should find for this test, based on the performance of the algorithms that did well in the first test, it can be inferred that the uncertainty in the X- and Y-directions should be 0.0953 m, and the uncertainty in the Z-direction should be 0.0500 m. Thus, the optimization algorithm that successfully passed both tests and had the shortest computation time was the one called DE. Interestingly, PSOAIW was even faster than DE but its results from the second test did not exactly match the results that the other algorithms calculated. Therefore, if speed is the only concern the PSOAIW algorithm may be a viable choice. However, the DE algorithm appears to combine both speed and accuracy.

Even though the DE algorithm was the fastest algorithm that correctly found the answer, it still took 940,000 ms, or 16 min, to find a single TLE measurement. Depending on the number of TLE calculations that the user wants to perform and the time in which the algorithm can perform these calculations, this may be unacceptable. Appendix C outlines the improvements that were made after this test was performed to improve the speed of the calculations.

Appendix C – Optimizing the Algorithm for Speed

Optimize Computational Time Overview

The main drawback of the Multipurpose Universal Simplified TLE Calculator (MUSTC) program, as configured during the verification and validation (V&V) process, is that it was time consuming to run. To make the program more useful, it was important to try to overcome this drawback. There were some simple changes made to improve the computational time. For example, because one target location error (TLE) calculation does not typically depend on another, the application was made to be multi-threaded so that more than one TLE value could be calculated at once. Another change was to tailor the settings in Visual Studio so that the final application is better optimized to run on the U.S. Army Combat Capabilities Development Command (DEVCOM) Analysis Center hardware. However, these simple fixes can only do so much. To have a significant decrease in computational time, changes were needed that would optimize the MUSTC program code itself.

There is a general rule in computer science that says that a program will typically spend around 90% of its time running around 10% of the code. The most efficient way to speed up a program is to find the 10% of code that the program is spending most of its time running and then to optimize that. Optimizing the other 90% of code is a waste of time and may be counterproductive as it may make the code more complicated, error prone, and harder to maintain.

For the MUSTC algorithm, it is easy to determine which bit of code is spending the most time calculating. For each variable that might affect the final TLE calculation, the MUSTC algorithm will set a perturbation to this variable. The algorithm will then ask an optimization algorithm to find the error in the location of the target of interest that relates to the specified variable perturbation. Finding this value requires the optimization algorithm to make repeated calls to the cost function. As a result, the program spends most of its time calling this cost function at the request of the optimization algorithm.

The first step in speeding up the program was to find the best optimization algorithm to meet the requirements. A good optimization algorithm should be able to find the final location error while calling the cost function method as few times as possible. In Appendix B, various optimization algorithms were tested to see which could be used for this application and which return TLE values that may be in error. The speed it took for each algorithm to perform the calculations was also a factor in determining which was the best optimization algorithm for the MUSTC algorithm to use. While the chosen optimization algorithm consistently found the correct answer and was faster than all the other optimization algorithms, it still must repeatedly call the cost function before calculating the final TLE value. This leaves room for additional improvements.

Another change that could be done to speed up the algorithm would be to increase the speed of the code that implements the cost function. Since calling the cost function is the process that takes the most time, speeding up that code could speed up the optimization algorithm. However, this might not be a good idea as the cost function is already pretty fast and making it even faster could affect the program's expandability and modularity. This leads to the conclusion that the best way to decrease computational time is to decrease the number of times that the optimization algorithm is called in the first place. For example, if the program can be modified so it only needs to call the optimization algorithm 1/10th of the number of times, then the program will run 10× faster. However, decreasing the number of times that the program calls the optimization algorithm could decrease the accuracy of the final TLE estimate.

While developing the algorithm, DEVCOM Analysis Center (DAC) erred on the side of making the calculation more accurate at the expense of computational time. If errors or problems were found while debugging the program or performing V&V, DAC wanted to be sure that the errors were from problems in the underlying code and not because of optimizing the algorithm for speed before it was ready. After performing V&V and debugging the code, the program seems to be working as expected and now is the time to become more aggressive with the optimization. Now we can try to speed up the processing time while only imparting a nominal impact on the accuracy of the calculation.

The first way to decrease the number of times that the optimization algorithm is called is by increasing the step size between perturbation values. When finding how a change in value of a single variable can affect the TLE of the object, the program steps through a bunch of errors on the variable. The program adds these errors to the variable of interest in the form of perturbations. The perturbations range from –4 to +4 standard deviations from the variables starting value. If the step size between perturbation values is set to 0.01 standard deviations, the program would calculate the error in location for the target of interest when the variable has a perturbation that is –4.00, –3.99, –3.98, . . . , +3.98, +3.99, +4.00 standard deviations from the starting value. This means that the optimization algorithm will be called 800 times (there is no need to find the error when the perturbation is set to zero). However, if the step size between perturbation values were set to 0.1 standard deviations, the program would call the optimization algorithm to find the error in the location when the variable has a perturbation that is –4.0, –3.9, –3.8, . . . , +3.8, +3.9, +4.0 standard deviations from the starting value of the variable. In this case, it would only need to call the optimization algorithm 80 times. With this change, the program could run 10× faster.

Before increasing the step size between perturbations in the name of decreased computational time, it is important to examine how this change will affect the final TLE value. The first way in which the step size could affect the final TLE value is that it could interfere with the optimization algorithm itself and cause it to select a local instead of a global minimum. This is because the program actually adds these perturbations to the variable in a very specific order. If the step size between perturbation values is set to 0.01 standard deviations, the program would start by adding a perturbation that is 0.01 and then –0.01 standard deviations from the starting value of the variable. For these calculations, the program would feed the starting location of the object of interest as the initial guess for the optimization algorithm. This is a good initial guess as 0.01 and –0.01 represent a very small perturbation to the variable. This makes it much less likely that the optimization algorithm will find a local instead of a global minimum. Once the program finds the error when the variable is 0.01 and –0.01 standard deviations from the starting value, it will use those values to try to find the error in the target of interest's location when the perturbation is set to 0.02 and –0.02 standard deviations from the starting value. This process will continue until it uses the errors at 3.99 and –3.99 as the starting guesses when computing the error at 4 and –4 standard deviations from the starting value.

When the program adds 4 and –4 standard deviations of error to the variable it is likely that the optimization algorithm will find that this will equate to quite a large error in the location of the target of interest. Even with this large error, the optimization algorithm can still avoid a local minimum as it will feed the error when the variable was at 3.99 standard deviations from the starting value as the initial guess.

As the step size becomes large, the optimization algorithm may be more likely to return location errors that are incorrect because of local minima in the cost function. An optimization algorithm that is less prone to such errors was selected, but it is still possible for even the best algorithm to make such an error if it is fed a bad starting guess. Because the next calculation is somewhat dependent on the last value, an error at a smaller perturbation value can propagate and cause even greater errors at perturbations with larger magnitudes.

Another problem with making the step size between perturbations too large is that it can still interfere with the final TLE accuracy even if it is not large enough to confuse the optimization algorithm. Selecting a step size between perturbations of 0.01 is essentially stating that the error when the perturbation is set to 0.005 is basically the same as when the perturbation is set to 0.015. This might be a good assumption if the step size between perturbations is 0.01 standard deviations, but what if the step size is set to 1 standard deviation? Is the error at 0.5 standard deviations basically the same as at 1.5

standard deviations? Setting the step size to 1 standard deviation would mean that only eight calculations are needed to find the TLE; how accurate would the TLE calculation be in this case? The best way to answer these questions is with an empirical test.

Another question that may need an empirical test to answer is as follows: does the algorithm need to keep the step size between perturbations constant while it is calculating how this variable will affect the final TLE? Perhaps the algorithm could decrease the step size in the areas that most contribute to the final TLE while increasing the step size in areas that contribute the least to the final TLE value. Figure C-1 highlights this point.

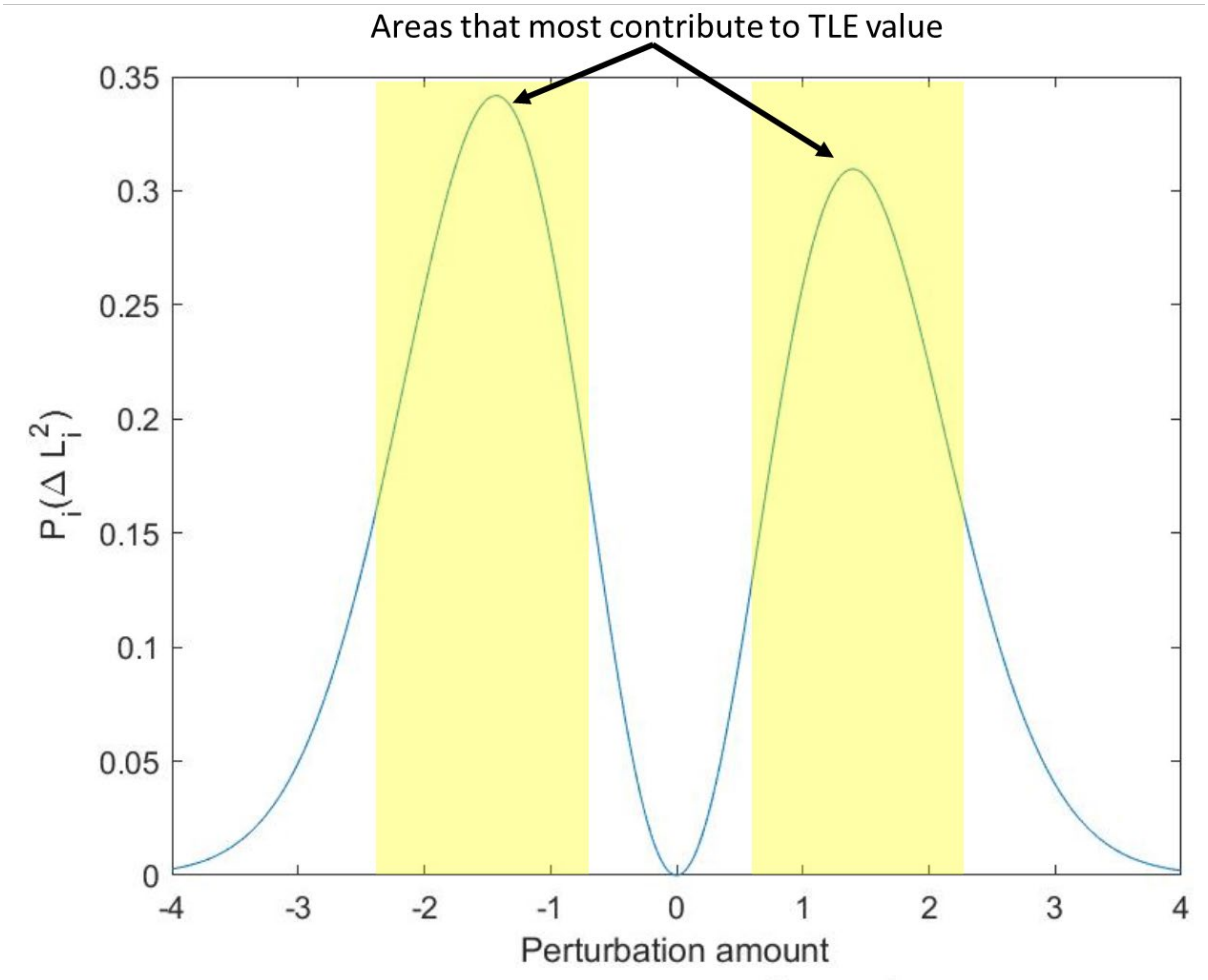

Figure C-1. Perturbation values that contribute most to the final TLE value

Figure C-1 shows the probability that a perturbation will happen multiplied by the square of the error that the optimization algorithm found when adding the perturbation to the variable. Remember, the amount of TLE a single variable will add to the total TLE of the object of interest in a single direction will be the square root of the sum of all values in

Figure C-1. The algorithm performs a similar calculation in each direction and for each variable that could affect the final TLE value. Once the program determines how each variable affects the final TLE, it gathers this information together to find the final TLE value in each direction. See Equation (7) and Figure 4.

The shaded areas in Figure C-1 are areas that contribute the most to the final TLE value. The areas to the far right and left of the graph do not contribute much to the final TLE value because there is a small likelihood that the error in the variable will be that large. Even though the error in location these perturbations will cause is likely to be quite large, the probability of the error is so small that the probability multiplied by the square of the error in location quickly drops its contribution to zero. Similarly, the area in the center of Figure C-1 does not contribute much to the final TLE value because the error in location from such a small perturbation to the variable is also small. Thus, although the probability in the center part of Figure C-1 is extremely high, the probability multiplied by the square of the error is small.

Is it possible to keep the perturbation step size constant in the areas that are shaded in Figure C-1 but then start increasing the step size for areas further away from the shaded parts of the graph? Increasing and decreasing the perturbation step size throughout the calculation could make the probability calculation more complicated; it is much simpler to increase the step size by assuming that the location error from adjacent perturbation values are basically the same. In this way we can skip calculations and essentially increase the perturbation step size in different areas of the TLE calculation.

The damage that could come from skipping perturbation values at the start of the TLE calculation can be controlled only by starting to skip perturbation values when it has been determined that the error caused by these perturbations will be small. At that point, setting the initial guess to locations very close to the starting location of the object of interest would still be acceptable for the optimization algorithm. The following are questions to ask:

- What is the nominal perturbation step size?
- At what point is the error in location caused by small perturbations small enough that it can be skipped without the likelihood of causing the optimization algorithm to pick local instead of global minima and thus making the final TLE value inaccurate?
- At what point are the probabilities of large perturbations from the starting value small enough that they can start to be skipped without adversely affecting the accuracy of the final TLE value?

Algorithms are needed to find answers to these questions. These algorithms will likely need parameters to determine how aggressive the program should be when setting the step size, skipping calculations, and determining the optimal balance between the speed of calculation and the accuracy of the final TLE value.

Algorithms to Determine the Perturbation Step Size and When Calculations Can Be Skipped

The first step in maximizing the accuracy of the TLE calculation for a given computational time is to find an algorithm that finds the starting perturbation step size. Sometimes the correct perturbation step size can directly relate to a parameter that the user may have access to. For example, one of the variables that affects the TLE calculation is the uncertainty in location or velocity. The perturbation step size for these variables would represent the minimum precision in location or velocity for this scenario. The user can set these values, and therefore the starting perturbation step size for these variables.

However, how would the program find the perturbation step size for other variables that do not directly relate to a location and/or velocity precision parameter? Is there a way to set the step size for these variables based on parameters that we could set such as the precision in location or velocity? Luckily, the program already performs a set of calculations that can be used to determine the step size for additional varieties based on the location and velocity precision parameters that the user can supply. Section 2.3 outlined how each sensor measurement should be normalized so it goes from zero to one before it is fed into the optimization algorithm. This normalization algorithm can also be used to determine the step size for additional variables.

To normalize the measurement, the algorithm will examine each variable that the program might need to perturb in order to find the TLE. The program will perturb each variable by its maximum and minimum amount to determine the maximum and minimum measurements that each sensor should make while the system is performing its TLE calculation. Once the algorithm finds these maximum and minimum values, it can normalize the sensor measurements. The maximum and minimum sensor value calculations can also be used to determine how a perturbation in location or velocity will cause a change in the raw sensor measurements. This is because the program will set the location and velocity of all the objects in the scenario to their maximum and minimum values to see how those changes affect all the sensor’s raw measurement values. In this way, the program will determine how a given change in velocity and or location can affect the raw measurement value and can therefore find a perturbation step size for all the raw sensor measurements that are matched to the user-supplied location and velocity-precision parameters. However, there are variables other than the

location, velocity, and the raw-sensor measurements. The program needs an algorithm that finds the step size for these variables as well.

During normalization, the program will also need to consider how changes to all the variables will affect the raw measurements. Therefore, not only will the program know how changes in velocity and location will affect the raw measurements, the program will also know how changes in all variables of interest will affect the raw measurements as well. By examining how location and velocity changes affect the raw measurements, and then examining how changes in other variables will also change the raw measurements, the program can determine a perturbation step size for any variable of interest that is matched to the desired location and velocity precision that was supplied by the user.

The program is now able to calculate a nominal perturbation step size for any variable that might affect the TLE calculation based on a location or velocity precision parameter that is supplied by the user. However, there may be times when the algorithm should use a different perturbation step size than the one it found using this procedure.

For example, if the user set the location precision parameter to 1 mm and added a reference object whose location uncertainty is 1 km, would the program really need to go from –4 standard deviations to +4 standard deviations (–4 km to +4 km) from the starting value of the sensor with a step size equal to 1 mm? Such a calculation would take an extraordinary amount of time and produce a calculation with a significantly larger precision than is necessary. Allowing the user to supply a maximum number of perturbations as a parameter would essentially set a limit to the number of calculations that the program would need to make for any single TLE calculation. This could significantly reduce the run time.

Similarly, allowing the user to set a minimum number of perturbations needed to perform a single TLE calculation is worth considering. If a reference object is added whose location is defined within 1 mm, and the location precision is set to 1 mm, then the program would only perform eight calculations (from –4 mm to 4 mm with a step size of 1 mm and no need to perform a calculation at 0 mm). While the TLE from this variable is likely to be small, it would still be a good idea to perform more than eight calculations when analyzing this variable. Therefore, it is also a good idea to set the minimum number of perturbations.

Now the user has a set of parameters that will allow them to easily inform the program as to the perturbation step size it should be using while performing TLE calculations. These parameters will match the TLE calculation to the user’s desired precision. The parameters are as follows: location precision, velocity precision, and minimum and

maximum number of calculations. These parameters can speed the computational time significantly by ensuring that the program does not spend a large amount of time performing calculations with precisions that are much higher than what the user actually needs.

Recall from Figure C-1 there is another way to speed up the program's calculation. If the program were to skip some of the perturbation calculations in the non-shaded areas of Figure C-1 the computational time could be sped up even more with only a nominal impact on the accuracy of the final TLE measurement. This is because the calculations in the non-shaded areas in Figure C-1 will not have much effect on the final TLE value.

When the program is performing calculations that are inside the shaded area, the program should not skip any calculations at all. Therefore, the first two parameters the user needs to supply are the ones that define the starting and stopping points where the program should not speed up the calculations by skipping values.

Once the program makes it into an area where it can start to skip values, it needs to determine how aggressive it can be with the calculation skipping. One instance may be when the error probability is extremely small and therefore the calculations do not affect the final TLE value very much. The program needs to know what the value of the probability should be before it starts skipping calculations, and it may even skip multiple calculations at once as the probability gets smaller.

The program does a quick calculation to see if it can avoid sending a given perturbation value to the optimization algorithm to find the actual location error. Because the optimization algorithm can be time consuming, this quick calculation can save time. In particular, the quick calculation will determine if the probability of the perturbation value on the variable is so low that it will not have much effect on the final TLE value. To perform this quick calculation, the program defines the variable N_p. This variable is found using the following formula:

$$N_p = \frac{p_i}{\bar{p}}. \tag{C-1}$$

Here, p_i is the probability of the measurement the algorithm is considering skipping because it will not affect the TLE much and $\bar{p}$ is the mean of the probabilities for all measurements. To use this variable to decide whether a calculation should be skipped, the program needs a probability reference parameter from the user. The algorithm takes the probability reference parameter and divides it by the N_p variable it found using Equation (C-1). The program will then round the resultant value to the nearest integer. This integer represents the number of calculations that the program should skip because these values will not affect the final TLE much anyway. The user can also

define a maximum number of skips parameter. If the integer that it calculates is larger than the maximum number of skips given by the user, then the program will set the number of values to skip to the maximum number.

As an example, assume that the program is about to determine the error in the location of the object of interest caused by the next perturbation to a variable. Normally the program would just feed this problem to the optimization algorithm. However, to save time it might be advantageous to skip this calculation and assume the error for this perturbation is more or less equal to the error from the previous perturbation. Even if this assumption is not ideal, it might not matter since the probability of this perturbation is low enough as to not affect the final TLE calculation very much anyway.

To see if it can skip this perturbation because of lack of probability, the program will first compare the probability of this perturbation happening to the mean of the probabilities for all perturbations. Assume that the probability of this perturbation happening is one quarter of the mean probability for all perturbation values. In this case, N_p would be equal to 0.25. Assume that the user set the probability reference parameter to 0.5. This would imply that the user would like the program to start skipping calculations when the probability of the perturbation is roughly half that of the mean probability of all perturbations. The program would divide 0.5 by 0.25 and then round the value to the nearest integer. In this case, 0.50/0.25 = 2. The program would conclude that it can save time by skipping two calculations in a row. It would find the error value from the previous perturbation calculation and assume that it will be the same as the error value for this as well as the next perturbation values. Remember that the user does have the option to define the maximum number of skips.

If the user set the maximum number of skips to 1 the program would only be able to save time by skipping the calculation for this perturbation value. When the program gets to the next perturbation value it would have to send that perturbation value to the optimization algorithm to find the actual error in location caused by the perturbation on the variable. As an alternative, the user could have set the maximum number of skips to 0 (or clear a flag that tells the program that it should perform these skip calculations) and the program will just find the error value for each perturbation no matter the probability of this perturbation.

The program will perform a similar calculation when considering whether to start skipping calculations because the error caused by a perturbation in this region is too small. As with the algorithm designed to skip calculations because of low probability, the algorithm needs a variable similar to N_p to rate how likely it is that the program can skip this calculation as compared to other calculations. In the case of skipping because of low error values, this variable is just equal to the magnitude of the error caused by the

previous perturbation. As with the algorithm that skips because of low probability, the algorithm will divide the “small error skip” parameter given by the user by the magnitude of the error from the previous calculation to determine how many subsequent calculations the program can skip.

As an example, assume the program adds a small perturbation to the variable that it is calculating the TLE for. Because the perturbation is small, the error is also small. For this example, assume the error is equal to 10^{-4} m. Before the program calculates the error from the next perturbation on the variable, the program will check if the previous error of 10^{-4} m is small enough that the program can skip this calculation. The program will determine this by looking at the small error skip parameter that the user supplied. If the user set this parameter to 10^{-3} m, the program will divide 10^{-3} by 10^{-4} and round that value to the nearest integer. In this case, the nearest integer is 10. The program would conclude that it can skip this calculation and the nine subsequent ones.

When skipping calculations because of a small error, be careful not to skip too many calculations. Recall that the optimization algorithm uses the error calculation from the previous perturbation as an initial guess for the error for the next perturbation. If too many values are skipped the initial guess can be off, leading to the optimization algorithm returning error values from local instead of global minima. This problem could not only affect this calculation, but also all subsequent error calculations. This is why the user is given the option to set the maximum number of calculations that can be skipped; if the user set that value to 2 then the program would skip only 2 calculations even if the algorithm concludes that it might be able to skip 10.

The user has four parameters to use when deciding how aggressive the program should be when it is setting the step size between perturbation values. Those parameters are as follows: location precision, velocity precision, maximum number of perturbations, and minimum number of perturbations.

The user also has five variables that will inform the algorithm as to how aggressively it will skip calculations because the probability of the perturbation is too small or because the magnitude of the calculated error in a location is too small. Those parameters are as follows: the maximum and minimum magnitude values that define the area where the calculations should not be skipped, the probability reference parameter, the small error skip parameter, and the maximum number of calculations the program can skip in a row. There is also a flag that can tell the program to not skip calculations at all.

As the user becomes more aggressive with these parameters, the computational time for the TLE measurements will decrease. However, as the computational time decreases the accuracy of the final TLE calculation may decrease as well. A sweet spot

exists where the calculation times are as short as possible while still yielding valid TLE calculations. The next section explores how to determine the optimal values for each parameter.

Experiment to Determine Optimal Performance Parameters

Now that the parameters that can speed up the TLE calculations at the expense of some accuracy in the calculated values have been defined, it is necessary to determine how aggressive to be with these parameters. The goal will be to increase the computation speed as much as possible while trying to keep the errors caused by this speed up to an acceptable level.

To find the optimal parameters, a series of test calculations were performed. Sample scenarios were set up that would match the type of calculations the program would perform while doing more realistic calculations. For each scenario, location, and variable that might affect the TLE the program performed multiple calculations showing how the uncertainty in the variable would affect the final TLE. For the first calculation the program performed for each variable, the parameters were set to match the values used when debugging the program and performing V&V. In other words, the initial parameters produced accurate TLE calculations, but at the expense of computational time. The program then repeated each calculation multiple times with new parameters that could improve the computational time. Each time the program tried a new set of parameters, the parameters became more aggressive.

It is important to compare the TLE values the program calculated when the parameters were set to their least aggressive value to TLE values the program calculated as the parameters became more aggressive. This comparison shows how more aggressive parameter values lead to decreases in overall TLE calculation accuracy. Figure C-2 shows the result of this comparison.

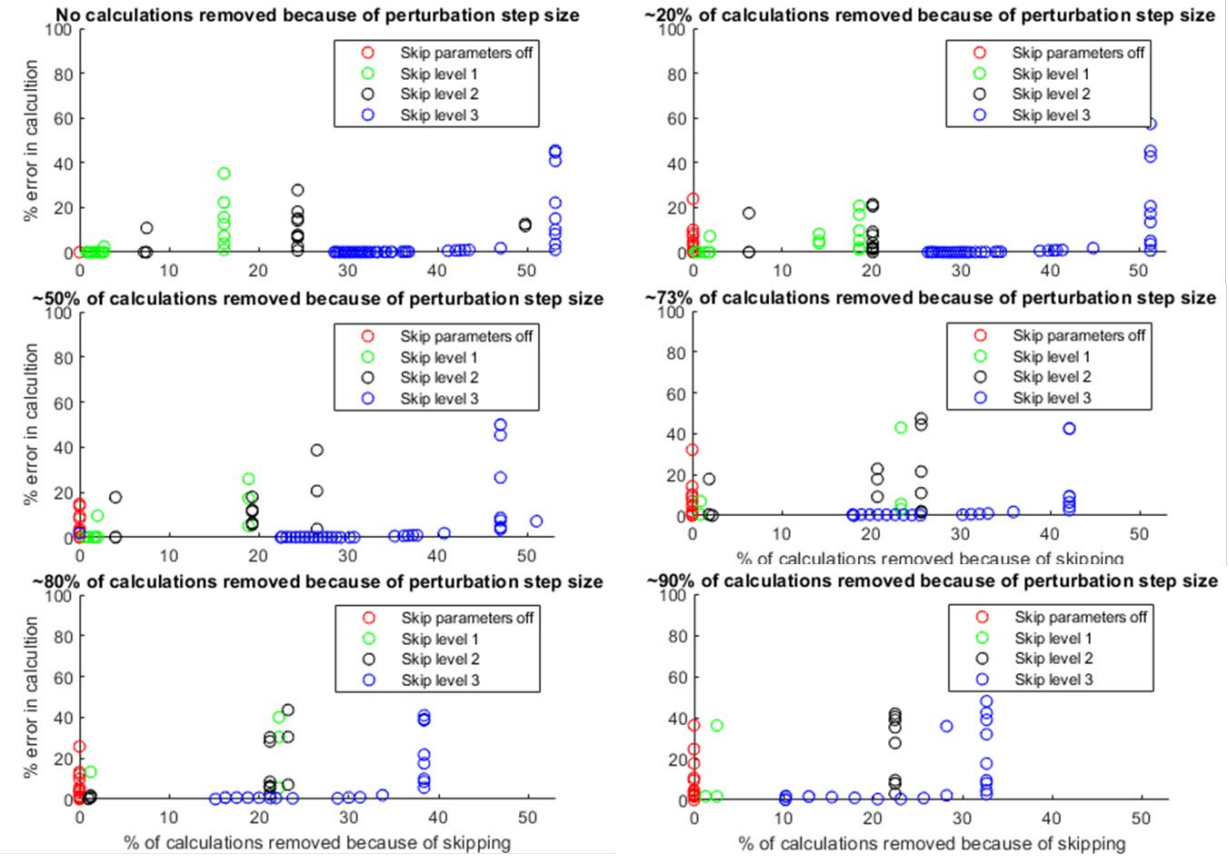

Figure C-2. Accuracy in the TLE estimation drops as the algorithm speeds up calculation time by not performing as many calculations

Each subplot in Figure C-2 shows the percent error in a set of TLE calculations for a different level of aggressiveness when it comes to the step size between perturbation calculations. The colors in Figure C-2 represent the additional percent error in a set of TLE calculations that resulted because the program was configured to skip calculations when they are outside the shaded areas shown in Figure C-1.

For the upper-left graph of Figure C-2 the step size between perturbations was set to their least aggressive value. The red circles in the upper-left graph were performed when the program did not skip calculations and the step size was set to its least aggressive value. All red circles in the upper-left graph show a 0% error in calculation. This is because the red circles in the upper-left graph were the reference TLE values.

In addition to red circles in the upper-left graph, there are also green, black, and blue circles. The changes in color represents an increase in the aggressiveness the program used when skipping calculations because they were outside the shaded areas in Figure C-1. Therefore, the green dots in the upper-left graph do show some error in TLE as seen by looking at the Y-axis. However, they also had some speed increase caused by skipping calculations as shown in the X-axis. As the program cycled through the

different colors, it skipped more calculations because they were outside the shaded area in Figure C-1 however, they also yielded ever-increasing errors in the TLE calculation.

As the subplots moved from the upper left to the bottom right in Figure C-2, the program became increasingly aggressive regarding the baseline step size between perturbations. Therefore, the bottom-right graph had a base step size that was significantly larger than what is shown in the upper-left graph. Thus, the red circles in the bottom-right graph show errors in the TLEs even though they are not skipping calculations because they are outside the shaded areas in Figure C-1. However, as the program cycles through the colors in the bottom-right graph, the TLE errors further increase as the program skips more calculations because they are outside the shaded areas. The aggressiveness of the parameters that dictate the perturbation step size was increased until the program removed up to 90% of its calculations.

The red circles in Figure C-2 graph the percent error in the TLE calculation when calculations were removed based on the aggressiveness of the perturbation step size parameters. These red circles show that becoming extremely aggressive when picking the perturbation step size will only impart a nominal hit on the accuracy of the calculation. Even when around 90% of the calculations were removed, most TLE calculation errors were under approximately 10%. In fact, there were only two calculations where the error was between approximately 30% to 40%. Having a percent error approaching 40% may be deemed unacceptable, but it is still possible to remove 80% of the calculations due to the perturbation step size parameters without reaching a 40% error in the TLE calculation.

The plots in Figure C-2 are only a sample of possible TLE calculations. As the sample size increases, calculations will have larger errors because of the aggressiveness of the perturbation step size parameters. However, the examples in Figure C-2 are reasonably representative of the types of calculations the MUSTC algorithm will perform.

The parameters that skipped calculations because the location measurement error or the probability of the location error were too low were not as successful as the one that increased the step size between perturbations. When the skip parameters were at their most aggressive setting (level 3), as many as 40% of the calculations were skipped. This is in addition to the calculations that were already skipped because of perturbation step size.

However, even at its least aggressive value (level 1), the percent error that is imparted on the final calculation could be as high as 40%. If the goal is to decrease the computational time to make it is as low as possible while still creating TLE

measurements that are at least of the correct order of magnitude, then it would be acceptable to set both the skip parameters and the perturbation step size parameters at their most aggressive values. However, if the goal is to get the best result with the smallest effort when removing calculations without affecting the TLE accuracy, then Figure C-2 suggests it is better to be more aggressive with the perturbation step size parameters than the parameters that skip calculations.

Appendix B outlined the test used to find the best optimization algorithm for the application. It took around 16 min for the best optimization algorithm to produce a single TLE value for a single variable. The time to calculate a single TLE value could be much higher if the program has to investigate many variables. A long calculation time may be acceptable if the algorithm only needs to produce a few TLE values at a time. However, if a multitude of TLE values are needed to map out TLE as a function of location over a wide area, such a calculation time might not be acceptable.

If many separate TLE calculations are required, the first step to make the program faster is to make it multithreaded. Modern computers typically have 16 or more processor cores. By making the program multithreaded, and by ensuring that the individual TLE calculations can be done out of order and possibly simultaneously, the average time it takes for the program to calculate a single TLE value can be shortened to 1 min or less (it is still 16 min per TLE calculation but since the computer has 16 cores, it can calculate 16 TLE values at once). A more powerful computer may use high-performance computing (HPC) to calculate even more TLE values at once.

If the average of 1 min per TLE calculation on a 16-core computer is not sufficient, the user can sacrifice some accuracy to speed up the program. The program can skip up to 90% of its calculations by making the parameters that govern the perturbation step size more aggressive. The results from Figure C-2 suggest that this might impart an error of up to 40% on the final calculation values. If only an order of magnitude estimation of the TLE is needed, then an additional 30% of calculations can be removed by adjusting the parameters that govern the number of values that should be skipped to be more aggressive.

While processing time may be a main drawback of this algorithm, users have options available to make useful TLE estimations with less processing time. The algorithm still creates reasonable estimations for the TLE even as the parameters that govern processing time get more aggressive, which adds confidence that the program is working correctly. If it were not working correctly, because of instabilities in the calculation, a small change in the parameters that govern processing time could result in a huge change in the TLE calculations.

Appendix D – Verification and Validation

After identifying an algorithm that claims to be able to find the target location error (TLE) of an object of interest, it must be proven that it can do what it claims.

Appendix B outlined two tests that were used to identify the best optimization algorithm to use. The first test involved simultaneously moving all reference assets that are used to find the location of the asset of interest in the same direction and by the same amount. If the algorithm is properly modeling all sensors and targets of interest, and the optimization algorithm is working as expected, then it is anticipated that the optimization algorithm will find that the cost function is minimized when it moves the object of interest in the same direction and by the same amount that the reference assets were moved. Appendix B demonstrated that there were several optimization algorithms that could correctly pass this test. One such algorithm was the brute-force optimization algorithm. This algorithm finds the location that minimizes the cost function by setting up a grid of locations and then testing the cost function at every point in that grid. The fact that the brute-force optimization algorithm passes this test adds confidence that the algorithms were working as intended and their positive results were not based on a coincidence.

Although this test demonstrated that the optimization algorithm as well as the sensor and target models appear to be working correctly, it does not prove that the model can accurately estimate the TLE of a location using that algorithm. Therefore, a scenario where the TLE that the algorithm calculates can be verified by another means is needed.

One scenario where the expected TLE can easily be estimated is when Signals Intelligence (SIGINT) systems using angle of arrival (AOA), or electro-optical/infrared (EO/IR) angle sensors that do not have laser range finders (LRFs), are used and there are exactly two sensors to find the location of the target. If the sensors are orthogonal to each other, then the uncertainty in one direction would only be from one sensor and the uncertainty from the other direction will only be from the other sensor. Figure D-1 illustrates this experimental setup.

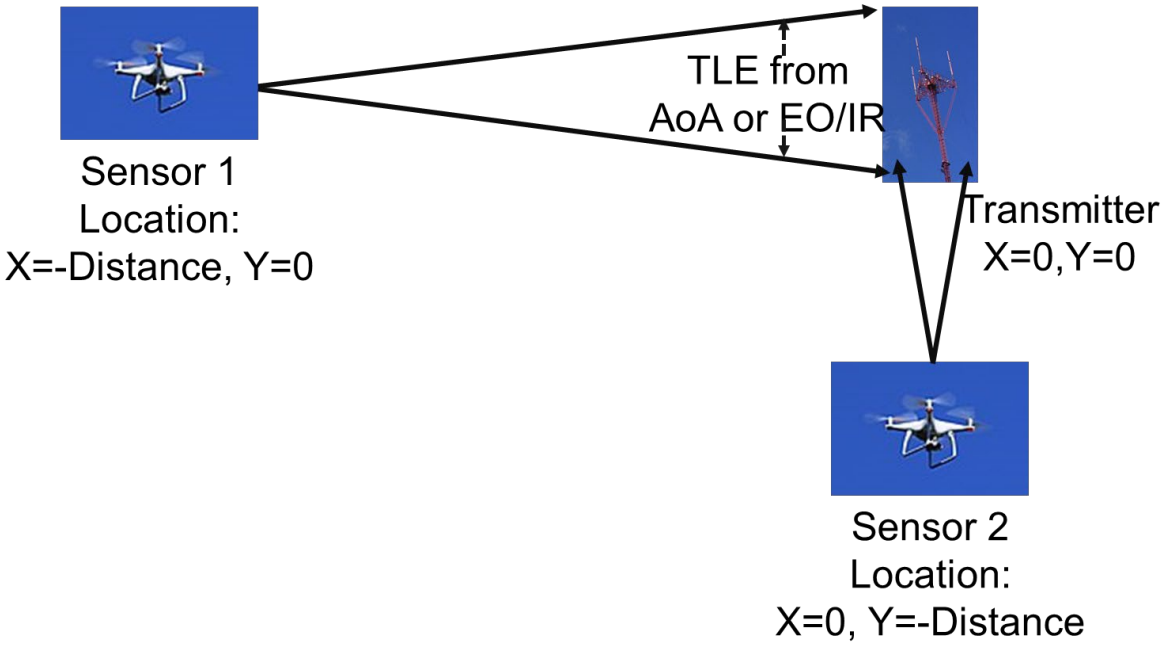

Figure D-1. Experimental setup where the uncertainty in one direction is only caused by a measurement uncertainty of a single sensor

As can be seen Figure D-1, the target was placed at the origin (X = 0 and Y = 0). Sensor 1 was set at some distance from the target in the X-direction, and Sensor 2 was set at some distance in the Y-direction. The uncertainty in measuring the location of the target in the Y-direction will be equal to the distance from Sensor 1 to the target multiplied by the uncertainty that Sensor 1 has in measuring the azimuth angle (assuming that the azimuth uncertainty is small enough for the small angle approximation to be valid). Likewise, the uncertainty in the X-direction will be equal to the distance from Sensor 2 to the target multiplied by the uncertainty in using Sensor 2 to measure the azimuth angle. Table D-1 shows the results of feeding the scenario shown in Figure D-1 into the Multipurpose Universal Simplified TLE Calculator (MUSTC) software when SIGINT with AOA is used to measure the angle. Table D-2 shows the results when EO/IR angle sensors are used.

Table D-1. Results of Feeding the Scenario in Figure D-1 into the MUSTC Program When the Operator is Using SIGINT AOA to Find the Azimuth Angle

Azimuth Uncertainty (radians)	Sensor 1 Distance (m)	Sensor 2 Distance (m)	Uncertainty from Sensor 1 (m)	Expected from Sensor 1 (m)	Uncertainty from Sensor 2 (m)	Expected from Sensor 2 (m)
0.0001	1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	1,000	–1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	–1,000	0.0999	0.1	0.0999	0.1
0.005	1,000	1,000	4.9961	5.0	4.9961	5.0
0.0001	70,000	1,000	6.9944	7.0	0.0991	0.1
0.0001	1,000	70,000	0.0999	0.1	6.9910	7.0
0.0001	70,000	7,000	6.9944	7.0	0.6994	0.7

Table D-2. Results of Feeding the Scenario in Figure D-1 into the MUSTC Program and the Operator Using EO/IR Angle Sensors to Find the Azimuth Angle

Azimuth Uncertainty (radians)	Sensor 1 Distance (m)	Sensor 2 Distance (m)	Uncertainty from Sensor 1 (m)	Expected From Sensor 1 (m)	Uncertainty from Sensor 2 (m)	Expected from Sensor 2 (m)
0.0001	1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	1,000	–1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	–1,000	0.0999	0.1	0.0999	0.1
0.005	1,000	1,000	4.9961	5.0	4.9961	5.0
0.0001	70,000	1,000	6.9944	7.0	0.0991	0.1
0.0001	1,000	70,000	0.0999	0.1	6.9910	7.0
0.0001	70,000	7,000	6.9944	7.0	0.6994	0.7

As seen in Tables D-1 and D-2, when the MUSTC algorithm models SIGINT sensors using AOA, or EO/IR sensors without LRF, the program returns uncertainties in each direction that are almost identical to what is expected. It does not matter if there is an increase to the azimuth uncertainty, an increase in the range to each sensor, or if we flip the scene so the locations of the sensors are in the opposite direction, the algorithm always returns a TLE that is basically equal to the expected value. The results from the AOA SIGINT sensors are identical to the results from the EO/IR angle sensor. This is expected as the program uses much of the same code to estimate the angles of interest for both sensor types.

The fact that the program finds the expected result even if the scenario is flipped is significant as it demonstrates that the program handles angles correctly. The program must decide if the optimization algorithm might have to deal with angles at around 0° or 180°. If the optimization algorithm might deal with angles at around 0° it is important that

the program define the angles in the scenario, so they go from 0° to 180° and 0° to –180°. If the program defines the angle so it goes from 0° to 360° the optimization algorithm will get confused because of the huge bump in the measured angle value (0° to 360°) from such a small change to the actual value (small positive angle to a small negative angle). If the optimization algorithm may have to deal with angles at around 180° the program must do the opposite and define the angle to go from 0° to 360°. Once the program decides how to define the angle it must remain consistent for each sensor throughout the calculation. Tables D-2 and D-3 demonstrate that the program does this correctly.

The experiment shown in Figure D-1 can be modified so that instead of being in the X- and Y-directions, the problem can be defined in the X- and Z-directions. Then testing can be performed to see if the algorithm can correctly perform this experiment when elevation angle measurements are used instead of azimuth angles, and when the scenario is expanded so it is in 3-D instead of 2-D. Table D-3 shows the results of this modified experiment in the 2-D of interest (X- and Z-directions). Table D-4 shows the results of this experiment for 3-D (Y-direction). Once again, the results were identical whether the program models EO/IR angle sensors or SIGINT AOA sensors. Therefore, the SIGINT AOA sensors were not shown because they were redundant.

Table D-3. Results of the Experiment in Figure D-1 When the Scenario is Changed to be in the X- and Z-Directions and the Operator is Using EO/IR Angle Sensors to Measure the Elevation Angle to the Target

Elevation Uncertainty (radians)	Sensor 1 Distance (m)	Sensor 2 Distance (m)	Uncertainty from Sensor 1 (m)	Expected from Sensor 1 (m)	Uncertainty from Sensor 2 (m)	Expected from Sensor 2 (m)
0.0001	1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	1,000	–1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	–1,000	0.0999	0.1	0.0999	0.1
0.005	1,000	1,000	4.9961	5.0	4.9961	5.0
0.0001	70,000	1,000	6.9944	7.0	0.0991	0.1
0.0001	1,000	70,000	0.0999	0.1	6.9910	7.0
0.0001	70,000	7,000	6.9944	7.0	0.6994	0.7

Table D-4. Results from the Elevation Angle Experiment Shown in Table D-3 with the Results Shown for 3-D (Y-Direction)

Elevation Uncertainty (radians)	Sensor 1 Distance (m)	Sensor 2 Distance (m)	3-D Uncertainty (m)	Expected 3-D Uncertainty (m)
0.0001	1,000	1,000	6.4E-13	0
0.0001	1,000	–1,000	7.4E-13	0
0.0001	–1,000	1,000	5.2E-13	0
0.0001	–1,000	–1,000	6.0E-13	0
0.0050	1,000	1,000	6.6E-13	0
0.0001	70,000	1,000	6.6E-04	0
0.0001	1,000	70,000	4.6E-13	0
0.0001	70,000	7,000	4.2E-04	0

As seen in Tables D-3 and D-4, the program seems to perform this 3-D calculation correctly. As with the 2-D azimuth experiment, the program finds values that are almost identical to the expected ones for the uncertainties in the X- and Z-directions. Also as expected, the program finds uncertainties that are basically zero for the Y-direction in this 3-D problem.

This experiment demonstrates that the program can correctly model EO/IR angle sensors as well as SIGINT sensors using AOA. However, is there a way to expand this experiment to test SIGINT systems using frequency difference of arrival (FDOA) or time difference of arrival (TDOA) to find the location of the sensor?

It may be possible to expand this experiment to FDOA sensors because when FDOA sensors measure frequency shift, they are actually measuring values that depend on the angles between the sensors and the emitters. Therefore, instead of measuring the azimuth from the sensor to the emitter directly, FDOA measurements are indirectly measuring something that is related to the angle between the velocity vector and the vector from the emitter to the sensor. Because the sensors are measuring something that only indirectly relates to the angles of interest, there are still some issues that we must address to make the scenario similar to what is shown in Figure D-1.

The first problem with trying to make a scenario similar to Figure D-1 for FDOA sensors is that FDOA sensors cannot directly measure a shift in frequency. Instead, FDOA sensors measure the difference in frequency shifts as measured by one sensor and compared to another. This makes isolating different FDOA measurements difficult. But what would happen if one of the FDOA sensors was actually mounted on the asset that has the RF emitter? Such a scenario probably would not make sense in real life; however, the MUSTC algorithm is so versatile that it can work with this type of scenario anyway. If a sensor were to be mounted on the same asset as the emitter the difference

in frequency between the external sensor's measurement and the measurement of the sensor that is mounted on the emitter would be given by

$$\Delta F = f_o \frac{|v| \cos(\alpha)}{c}. \quad \text{(D-1)}$$

Here, ΔF is the change in frequency between the sensor mounted on the emitter and the sensor that is mounted elsewhere, f_0 is the base frequency, v is the velocity of the sensor, α is the angle between the vector from the sensor to the target and the velocity vector, and c is the speed of light.

Additionally, in creating an FDOA scenario similar to the AOA scenario shown in Figure D-1, the frequency difference as measured by Sensor 1 must be independent of the frequency difference as measured by Sensor 2. Not only should they be independent, but they should be set up so that one sensor only measures the TLE in one direction while the other sensor only measures the TLE in the other direction. This type of geometry can be achieved by setting the velocities of each sensor such that they are perpendicular to each other and to the location vectors between the sensors and the emitter. The velocity vector for Sensor 1 should only be in the Y-direction and the velocity vector for Sensor 2 should only be in the X-direction.

We would like to add a small perturbation to the azimuth angle between the location of Sensor 1 and the location of the emitter, as was done in the AOA and EO/IR angle examples. However, what frequency measurement perturbation should be chosen to create the desired azimuth angle perturbation? Figure D-2 shows the geometry when adding a small perturbation angle ($\Delta\theta$) between the transmitter and Sensor 1.

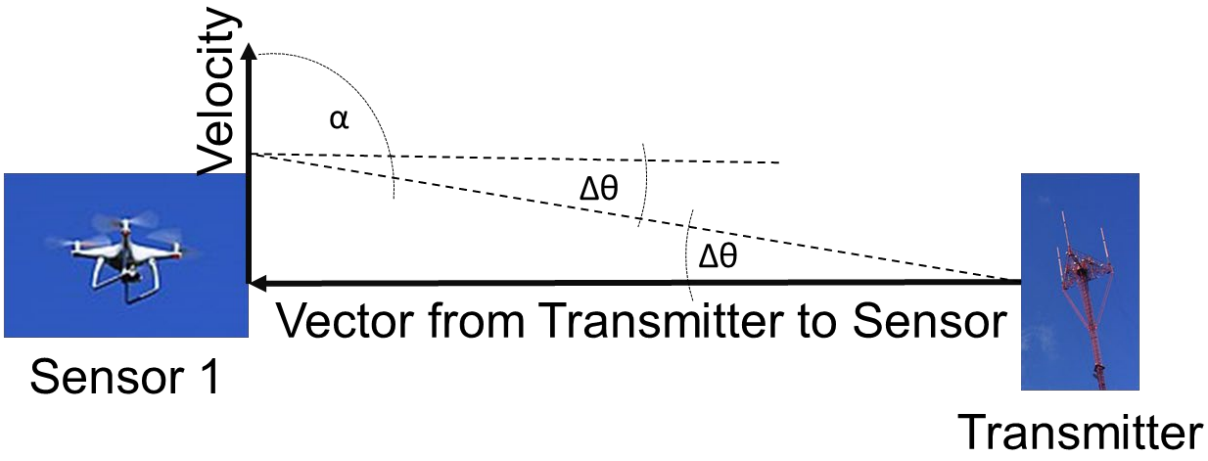

Figure D-2. Geometry when adding an uncertainty in azimuth angle to the test FDOA setup

As seen in Figure D-2, when adding an uncertainty to the angle between one of the external sensors and the item of interest ($\Delta\theta$), the angle α that must be used in

Equation (D-1) to find the frequency that the external sensor will measure will be equal to 90° + $\Delta\theta$. Since the velocity was perpendicular to the vector from the sensor to the target, the change in frequency before the uncertainty angle was added will be 0. A similar diagram could be made for Sensor 2. The only difference with Sensor 2 being that the vector between the external sensor and the transmitter would be in the Y-direction and the velocity would be in the X-direction.

Therefore, we must determine a frequency shift to go with the desired azimuth angle perturbation ($\Delta\theta$) to use in the FDOA experiment. Remember that the TLE for the target of interest would be equal to the perturbation angle multiplied by the distance between the transmitter and the external sensor. Also, because the geometries are perpendicular, the azimuth perturbation for Sensor 1 should only affect the TLE in one direction and the azimuth perturbation for Sensor 2 should only affect the TLE in the other direction. What frequency measurement uncertainty should be added to force the desired azimuth angle uncertainty? Knowing that α for the scenario should be equal to 90° + $\Delta\theta$, those values can be plugged into Equation (D-1) to get a formula for the desired frequency measurement uncertainty ($d\Delta F$):

$$d\Delta F = f_o \frac{|v| \cos(90^o + \Delta\theta)}{c}. \quad \text{(D-2)}$$

Therefore, if we want to set an azimuth angle uncertainty in one direction for our FDOA experiment, all we need to do is specify that the frequency uncertainty as measured by one of the external sensors should be equal to Equation (D-2).

This experiment does have one more complication to address. In a real-life situation, the SIGINT systems would be able to compare the frequency difference measurement from Sensor 1 directly to the Sensor measurement of Sensor 2 in addition to measuring the difference between the sensor mounted on the emitter and the measurements made by Sensors 1 or 2. Normally, this additional measurement would be beneficial as more measurements can create a more accurate location measurement. However, it is not beneficial to this specific scenario as it would cause the TLE in one direction to no longer be independent of the geometry from the sensor that is supposed to measure the TLE in the other direction. Fortunately, the MUSTC algorithm is versatile enough that this problem can be fixed. The sensor inclusion number parameter can be used to inform the program that Sensor 1 cannot directly communicate with Sensor 2. This means that the program will understand that the frequency difference between Sensor 1 and Sensor 2 will not be available for this test.

With these issues addressed, it is possible to perform a test for SIGINT sensors using FDOA. As was the case for the AOA SIGINT experiment, an azimuth uncertainty was

selected ahead of time for each sensor. Unlike the AOA SIGINT, for the FDOA experiment, the angle uncertainty was indirectly added via an uncertainty in frequency as calculated by Equation (D-2). This azimuth uncertainty was the only thing that drove the TLE in each direction. Table D-5 shows the results for this test.

Table D-5. Results from the FDOA Verification and Validation (V&V) Test

Azimuth Uncertainty (radians)	Sensor 1 Distance (m)	Sensor 2 Distance (m)	Uncertainty from Sensor 1 (m)	Expected from Sensor 1 (m)	Uncertainty from Sensor 2 (m)	Expected from Sensor 2 (m)
0.0001	1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	1,000	–1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	1,000	0.0999	0.1	0.0999	0.1
0.0001	–1,000	–1,000	0.0999	0.1	0.0999	0.1
0.005	1,000	1,000	4.9961	5.0	4.9961	5.0
0.0001	70,000	1,000	6.9944	7.0	0.0991	0.1
0.0001	1,000	70,000	0.0999	0.1	6.9900	7.0
0.0001	70,000	7,000	6.9944	7.0	0.6994	0.7

As seen in Table D-5, the uncertainties in the two directions matches the expected values. This adds confidence that the FDOA algorithm is performing as expected and demonstrates that the program's sensor-inclusion function is working. The only major sensor types left to examine are SIGINT systems using TDOA and photon counting detectors.

Is it possible to make a similar test for TDOA SIGINT systems? In some ways, performing TDOA is easier than FDOA. This is because, unlike FDOA and AOA that either directly or indirectly measure the angles in the scenario, TDOA sensors indirectly measure the distance between the transmitter and sensors. Therefore, to add an uncertainty in location in one direction, can an uncertainty in time measurement be added to the sensor located in that direction that is equal to the desired distance divided by the speed of light? Unfortunately, there are a couple of complications that must be addressed first.

As with the FDOA scenario, the first thing that must be done for this scenario is to place a sensor on the asset that has the transmitter. In this way, the SIGINT systems can directly measure the time it takes for the radiation to travel from the transmitter to both sensors. The second thing is to use the sensor-inclusion number parameter to once again inform the algorithm that Sensor 1 cannot directly communicate with Sensor 2. The only time differences that will be available in this scenario will be the distance from the transmitter to Sensor 1 divided by the speed of light and the distance from Sensor 2 to the transmitter divided by the speed of light.

To increase the TLE of the object of interest in the X-direction, it is only necessary to add a time uncertainty to the sensor responsible for this part of the TLE equal to the TLE value divided by the speed of light. Unfortunately, this will also impart a TLE in the Y-direction. Figure D-3 highlights this point.

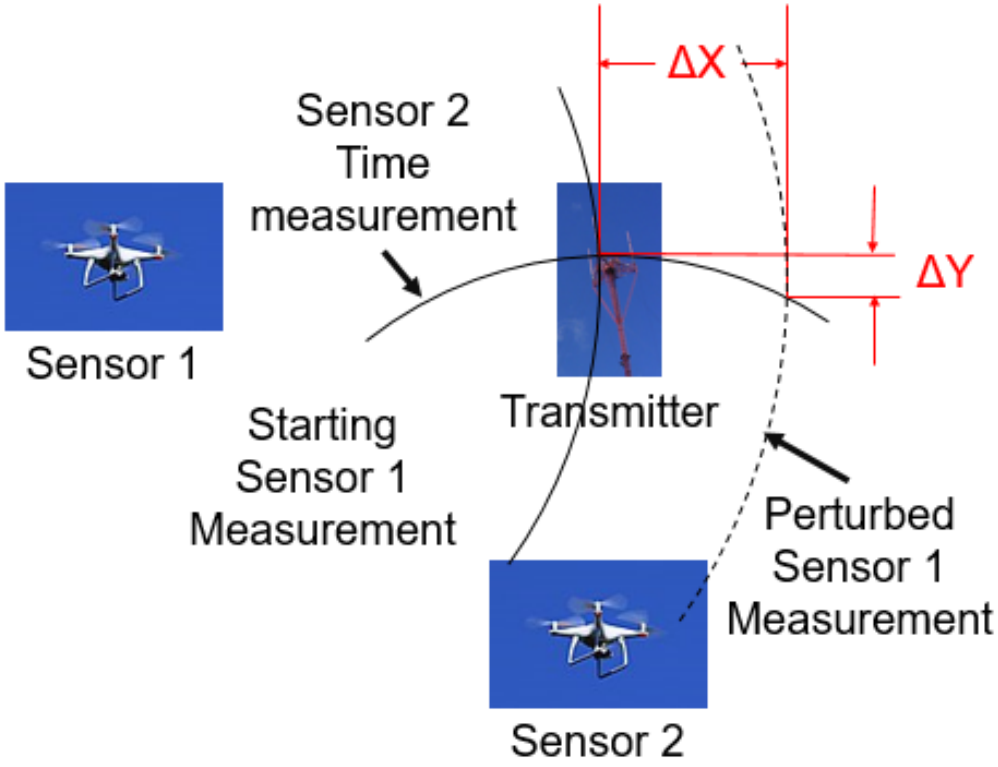

Figure D-3. Perturbation in time measurement for one sensor can add a TLE in both directions

As seen in Figure D-3, a perturbation to the Sensor 1 measurement can add a TLE in both the X- and Y-directions. However, the uncertainty in the Y-direction in Figure D-3 is much smaller than the uncertainty in the X-direction. By keeping the perturbation to each sensor measurement small, it should still mostly only affect the TLE in one direction. While this issue is easy to mitigate, there is another issue that presents more of a problem. Figure D-4 highlights this second issue.

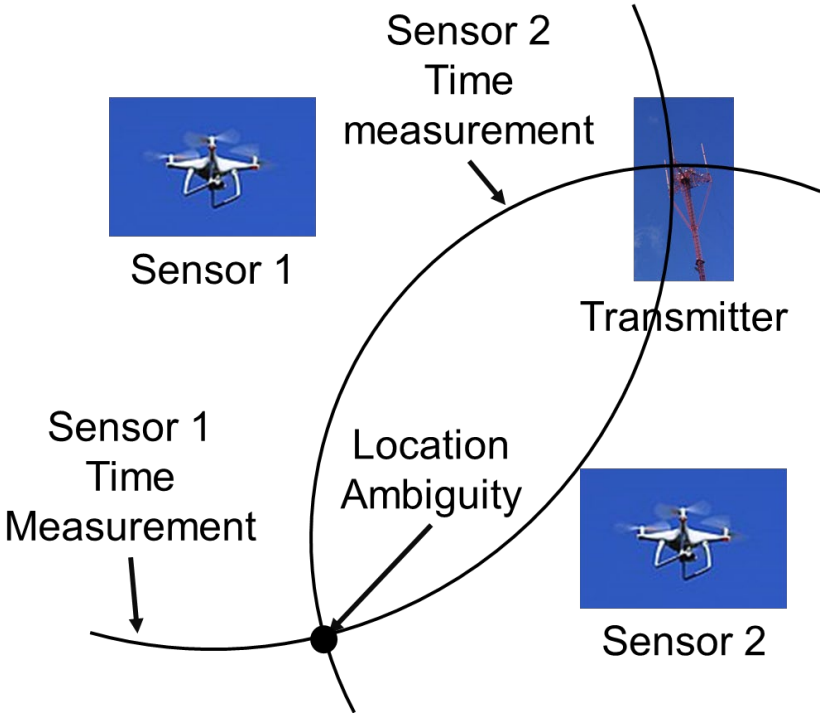

Figure D-4. Location ambiguity that could affect the TDOA experiment

As seen in Figure D-4, because a TDOA sensor was installed on the transmitter, each external sensor can now directly measure the distance between itself and the transmitter. All locations that are possible based on the Sensor 1 measurement can be represented as a circle centered on Sensor 1; a similar circle can be drawn around Sensor 2. A SIGINT system using TDOA in this scenario would determine that the emitter is located where the circles from Sensors 1 and 2 intersect. Figure D-4 shows that, because there are two circles, they intersect at two locations. This represents a location ambiguity that could confuse the algorithm.

In a real-world situation, SIGINT systems might add AOA measurements to remove the ambiguity. Such an addition would successfully remove the ambiguity, but it would also make it more difficult to estimate the TLE for this scenario. Real SIGINT angle sensors typically have much less precision than SIGINT time or frequency measurements. Real systems will often use the AOA measurement to remove the ambiguity while using the TDOA and/or FDOA measurements to find the exact sensor location. Because of this, the MUSTC program allows users to add weights to the different sensors and inform the program as to which sensor(s) should take priority. In this way, the MUSTC algorithm can be configured so that the TDOA and/or FDOA measurements take priority over the less accurate AOA measurements.

To complete this test, a single AOA sensor was added but the weight of the sensor was set to 0.01. This means that TDOA will have 100× more influence on the TLE than the

AOA measurement. The AOA may have a small effect on the final TLE, but the TDOA measurements should dominate the results. Table D-6 shows the results of this test.

Table D-6. Results from the TDOA Test

Perturbation Added Sensor 1 (m)	Uncertainty from Sensor 1 (m)	Perturbation Added Sensor 2 (m)	Uncertainty from Sensor 2 (m)
0.0	0.0001	0.1	0.0991
0.1	0.0991	0.0	0.0001
0.1	0.0991	0.1	0.0991
0.0	0.0022	–0.5	0.4996
–0.5	0.4996	0.0	0.0022
–0.5	0.4996	–0.5	0.4996

As seen in Table D-6, despite the AOA sensor, the TLE in each direction matches the time perturbation that was added to each external sensor divided by the speed of light. This test demonstrates that the program can successfully estimate the TLE for SIGINT systems using TDOA, and that the sensor weight function is working as expected.

The program might have been able to properly work even without the AOA sensor because the actual location and not the alternate location is used for the initial guess when the algorithm starts to use the optimization algorithm. However, as the algorithm adds larger perturbations it might be possible for the system to select the alternate location instead of the desired location for the asset of interest, making the results unstable. It is a good idea to try to avoid situations where there are location ambiguities. While the program tries to detect these scenarios, it might not always be successful.

Having demonstrated through testing that the MUSTC program can be used to find the TLE for EO/IR angle sensors and SIGINT systems, the next step is to demonstrate that the program can find TLE for photon counting detectors measuring the intensity of light from a known optical emitter. The MUSTC program assumes that the photon counting detectors measure the distance by measuring how the intensity of the radiation that the counters detect decreases as the distance between the sensor and the emitter increases. Because the photon counting detectors indirectly measure the distance from the sensor to the emitter, a test similar to the test for TDOA SIGINT sensors is needed.

One issue with performing a similar test with the photon counting detector as was done with the SIGINT TDOA detector is that it is not possible to set the uncertainty in making a photon counting measurement to an arbitrary number. For TDOA SIGINT sensors, the uncertainty in measuring the time depends on the signal-to-noise ratio (SNR) of the RF measurement. To change the time uncertainty, the power that the target transmits must be increased or decreased to increase or decrease the SNR. However, counting

photons follows a Poisson distribution. In Poisson distributions, the standard deviation is always equal to the square root of the mean. The uncertainty cannot be set to an arbitrary number; therefore, another way was needed to set the total uncertainty in the photon counting to an arbitrary number without changing the way Poisson distributions work.

The solution was to create a flag in the program that turns off the uncertainty in counting the photons. When this flag is set, the program assumes that the detector can somehow correctly count the number of photons and the algorithm will no longer automatically add a somewhat arbitrary (as far as this V&V test is concerned) uncertainty to the scenario. This flag has been marked as being for testing purposes only and should not be used for real-world problems. By using this flag, the affects the Poisson distribution has on the TLE can be avoided. However, a second test was needed to ensure that for more realistic scenarios, the program handles Poisson distributions correctly. An additional test was therefore performed to analyze the performance of the Poisson distribution versus a normal one; the results of that test are discussed later on in this section.

Once the Poisson distribution is overridden so that the uncertainty in counting photons is arbitrarily set to zero, the program will no longer automatically add this uncertainty to the scenario. However, we still need to find a way to explicitly add a desired uncertainty to the photon counting detector's measurement so we can confirm that such an uncertainty creates a TLE that makes sense. Unlike SIGINT and EO/IR angle sensors with LRF, there are variables other than the direct sensor measurement or the location of the reference assets that can affect the final location measurement that can have uncertainty added.

For example, uncertainty can be added to the efficiency of the photon counting detector. If the efficiency of the detector is lower than expected the measurement from a known light sensor would be less than expected for a light source from that distance. This might confuse the system and make it assume that the light source was farther away than it actually was. In this way, an uncertainty on the measurement can be add so that it creates a TLE that can be independently calculated. At the same time, this will also prove that the software can handle uncertainties on how the sensor measures something other than direct uncertainties in the sensor's final measurement.

How much uncertainty should be added to the photon efficiency to produce a given TLE? Recall from Equations (3) and (4) that this uncertainty depends on parameters such as the distance, the power transmitted, the visibility, and the dark current/shot noise. If the noise is set to zero and the visibility to an arbitrarily high number, the photons the system will count will be equal to

$$p = \frac{E\,P_t\,B}{d^2}. \tag{D-3}$$

Here, P_t is the power transmitted, d is the distance, E is the efficiency of the detector, and B is a constant that includes the energy of the photon, the transmittance of the optics attached to the detector, the square of the effective area of the detector, and so forth.

To ensure a given error in the distance between the emitter and the detector, what uncertainty should be added to the efficiency of the detector? To answer that question, use this equation:

$$\frac{E\,P_t\,B}{(d+\Delta_d)^2} = \frac{(E+\Delta_E)\,P_t\,B}{d^2}. \tag{D-4}$$

Here, Δ_d is the desired or set error in distance and Δ_E is the uncertainty in the detector efficiency. Solving for the uncertainty in detector efficiency leads to the following equation:

$$\Delta_E = E\left(\frac{d^2}{(d+\Delta_d)^2} - 1\right). \tag{D-5}$$

Equation (D-5) calculates the uncertainty in the photon detector efficiency that would equate to a set error in the distance. If a test were run where all uncertainties were set to zero except the photon detector efficiency, then the uncertainty in each direction should be equal to the Δ_d value that was fed into Equation (D-5). Table D-7 shows the results of such a test.

Table D-7. Uncertainty in the X- and Y-Directions for the Errors in Each Direction Set Via the Uncertainty in the Photon Counting Detector Efficiency

Set Error in X-Direction (m)	Calculated Uncertainty in X-Direction (m)	Set Error in Y-Direction (m)	Calculated Uncertainty in Y-Direction (m)
0.10	0.099944839	–0.10	0.099944839
–0.10	0.099944839	0.10	0.099944839
1.00	0.999459461	1.00	0.999459461
3.00	2.981984229	3.00	2.981984229
5.00	4.965296625	5.00	4.965296625
10.00	9.838557183	10.00	9.838557183
1.00	0.999488633	3.00	2.981974452
1.00	0.999686443	5.00	4.965250931
1.00	1.00297647	10.00	9.838199269
3.00	2.981974452	1.00	0.999488633
5.00	4.965250931	1.00	0.999686443
10.00	9.838199269	1.00	1.00297647

As seen in Table D-7, the uncertainty in each direction as calculated by the MUSTC program closely matches the absolute value of the error that was set by changing the uncertainty in the photon detection efficiency. The differences in Table D-7 are likely due to round-off errors and the need to add an AOA SIGINT sensor with a small weight as was required for the SIGINT TDOA test.

Table D-7 demonstrates that the photon detection sensor model seems to be working as expected and that the program can handle a variable that indirectly produces an uncertainty on the measurement, and not just an uncertainty on the measurement itself. However, can the program handle an uncertainty on a variable related to the target instead of the sensor?

Unlike the other sensor types that have been explored, for photon counting detection sensors there is a variable related to the target that can affect the sensor measurement. That variable is the uncertainty in the amount of power that the light source emits. By how much should the power emitted change to try to set an error in location? Adding the error to the power instead of the efficiency changes Equation (D-4) into:

$$\frac{E\,P_t\,B}{(d+\Delta_d)^2} = \frac{E\,(P_t+\Delta_P)\,B}{d^2}. \tag{D-6}$$

Here, Δ_P is the added error to the amount of power the transmitter transmits. Equation (D-5) will therefore become

$$\Delta_P = P_t\left(\frac{d^2}{(d+\Delta_d)^2} - 1\right). \tag{D-7}$$

A similar test to the one shown in Table D-7 was performed but the uncertainty was added to the power and not the photon detection efficiency. The results of this test are shown in Table D-8. Unlike Table D-7, independent errors to the X- and Y-directions cannot be added. Because power affects both sensors, the same error must be induced in both directions.

Table D-8. Uncertainty in the X- and Y-Directions for the Errors in Both Directions Set Via the Uncertainty in the Power that the Transmitter Transmits

Set Error Both Directions (m)	Calculated Uncertainty X-Direction (m)	Calculated Uncertainty Y-Direction (m)
0.10	0.099944839	0.099944839
–0.10	0.099944839	0.099944839
1.00	0.999459461	0.999459461
3.00	2.983650474	2.983650474
5.00	4.963629499	4.963629499
10.00	9.855268966	9.855268966

As seen in Table D-8, the uncertainties in each direction match the error that was set using the uncertainty in power. Once again, this confirms that the photon counting detection model in the MUSTC program is working as expected. This also proves that targets that have parameters that could affect the TLE can be correctly modeled.

The tests thus far demonstrate that the program can handle uncertainties added to each sensor measurement as expected. The uncertainty in the sensor's ability to make an accurate measurement can affect the TLE; however, in the real-world uncertainties the location of the reference assets often has a greater impact on the total TLE of a position, navigation, and timing (PNT) measurement. The final test will show how an increase in location uncertainty can drive the TLE in a more realistic scenario.

For this final test, an experiment with two reference assets is set up to allow the location uncertainty to have as large of an effect on the TLE as possible. The uncertainty of all other parameters except location uncertainty in this experiment were set to zero. The transmitter/target of interest was located a fair distance away from the sensors. This represents a much more realistic scenario. Figure D-5 shows this experimental setup.

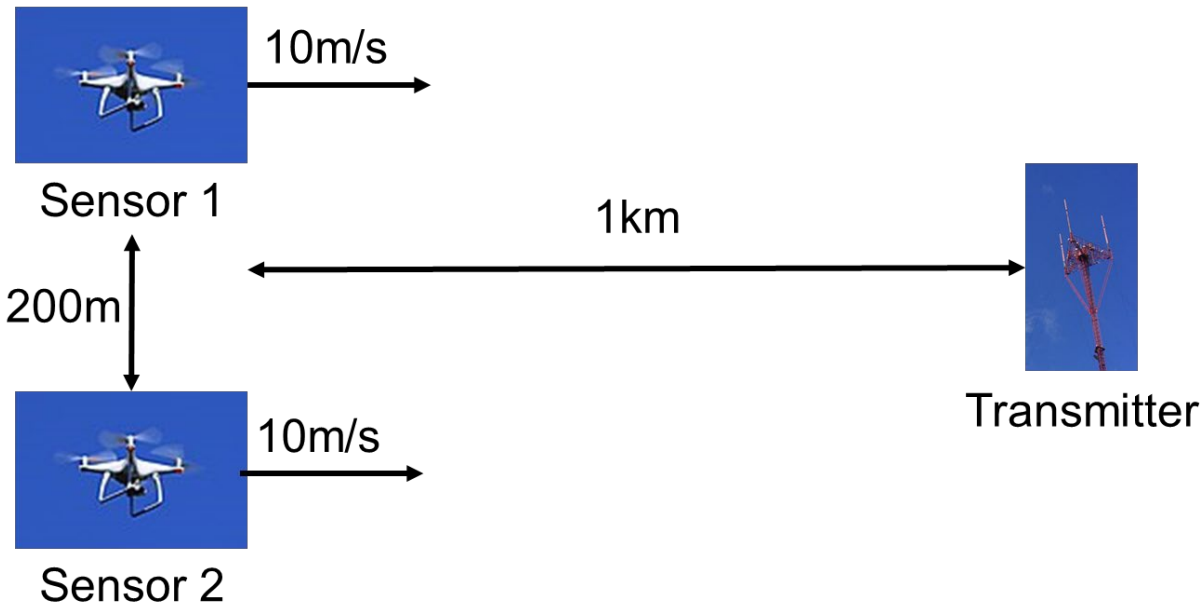

Figure D-5. Location uncertainty experiment

As seen in Figure D-5, the scenario includes two sensors that are 200 m apart and moving towards an emitter that is 1 km away at 10 m/s. There are only two sensors, so the emitter was placed on the ground to remove possible ambiguities (although the sensors are at an altitude of 10 m), and the systems are using TDOA, FDOA, and AOA to find the emitter location.

To see how an ever-increasing uncertainty in the location of a sensor can lead to an increasing TLE, a series of increasing uncertainties were added to the location of one or both sensors. For the first test, uncertainties in the location of Sensor 1 in just the X-direction were added. For the second test, uncertainties in the location of Sensor 1 were added in just the Y-direction. After that, uncertainties in both the X- and Y-directions for Sensor 1 were added. Finally, uncertainties in both the X- and Y-directions for Sensors 1 and 2 were added. Figure D-6 plots the results of these experiments.

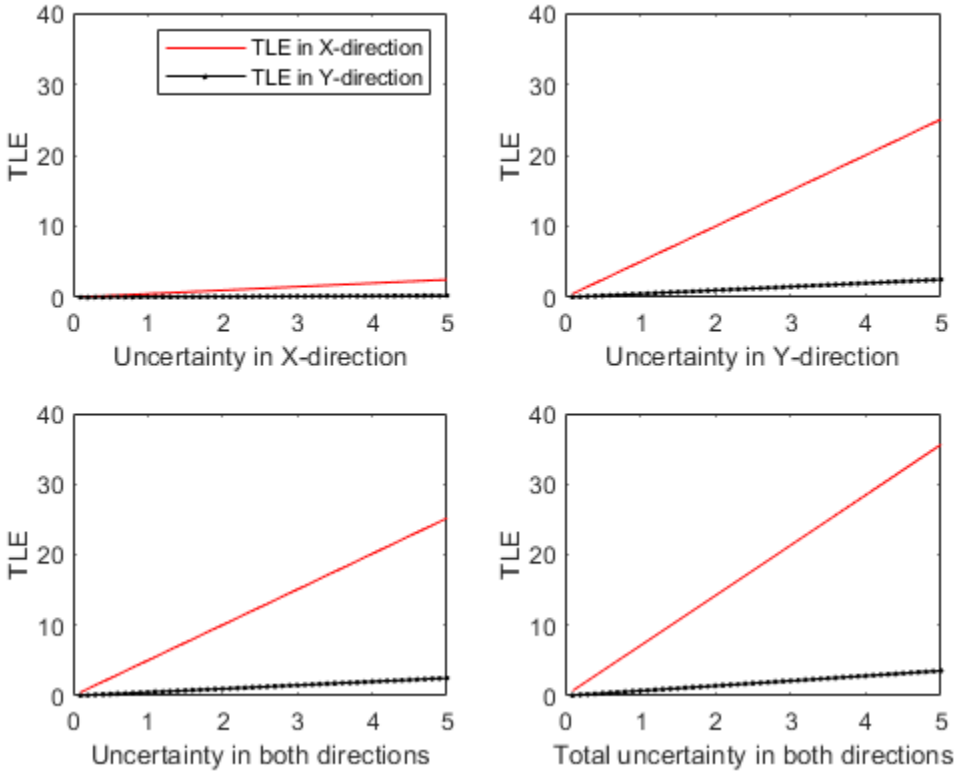

Figure D-6. Various results from the test for the scenario shown in Figure D-5

The upper-left graph in Figure D-6 shows the TLE in the X- and Y-directions as a function of the uncertainty that was added to the location of Sensor 1 in the X-direction. The solid-red-line is the TLE in the X-direction while the dotted-black-line is the uncertainty in the Y-direction. The TLE should increase as more uncertainty is added to the sensor location.

The upper-right graph in Figure D-6 plots the TLE in both directions as uncertainty is added to the location of Sensor 1 in the Y-direction. The algorithm determined that adding uncertainty in the Y-direction yields a higher TLE than was seen when an uncertainty was added in the X-direction. As with the upper-left graph, the upper-right graph shows that the TLE increases as more uncertainty is added—this is expected.

The lower-left graph in Figure D-6 plots the TLE in both directions as uncertainty in the location of Sensor 1 is added in both the X- and Y-directions. Since the TLE from uncertainties added in the Y-direction was larger than those added in the X-direction—and because the uncertainties in each direction are assumed to be independent—it is expected that the uncertainties added to the Y-direction will dominate the total uncertainties plotted in the bottom-left graph of Figure D-6. Indeed, it is the case that the

uncertainties plotted in the lower-left graph are similar but slightly larger than the uncertainties in the upper-right graph.

The graph in the bottom right in Figure D-6 shows the TLE when uncertainties in the X- and Y-directions are added to both sensors. The setup as shown in Figure D-5 is symmetrical; therefore, the same TLE is expected whether the uncertainties are being added to Sensor 1 or Sensor 2. Since these uncertainties are assumed to be independent, the total uncertainty should be equal to the square root of the sum of the square of either uncertainty. It is expected that the numbers plotted in the lower-right graph in Figure D-6 should be a factor of $\sqrt{2}$ larger than the numbers plotted in the lower-left graph, and this seems to be the case.

The graphs in Figure D-6show that the TLE in the X-direction is around 10× larger than the TLE in the Y-directions in all cases. Further, the TLE in both the X- and Y-directions is 10× higher when adding an uncertainty to a reference asset in the Y-direction than if it were added to the X-direction. This scenario is more complicated than the others, and it can be more difficult to tell what TLE the algorithm should find; however, a back-of-the-envelope calculation can be performed to see if these results make sense.

The SIGINT sensors are using AOA, TDOA, and FDOA at the same time to find the emitter location. AOA involves directly measuring the angle to the emitter while FDOA involves indirectly measuring the angle to the sensor via a change in the Doppler shift. TDOA is the only algorithm that involves indirectly measuring the distance to the sensors via measuring the time difference between measurements. There are twice as many angle-dependent measurements than distance measurements for this scenario; therefore, the difference in angle should dominate the final TLE. How do the angle measurements change when adding an uncertainty to the sensor location in the X- or Y-directions? Figure D-7 shows how the azimuth angle between the emitter and the sensor changes (θ to θ') as uncertainty is added in the X- and Y-directions.

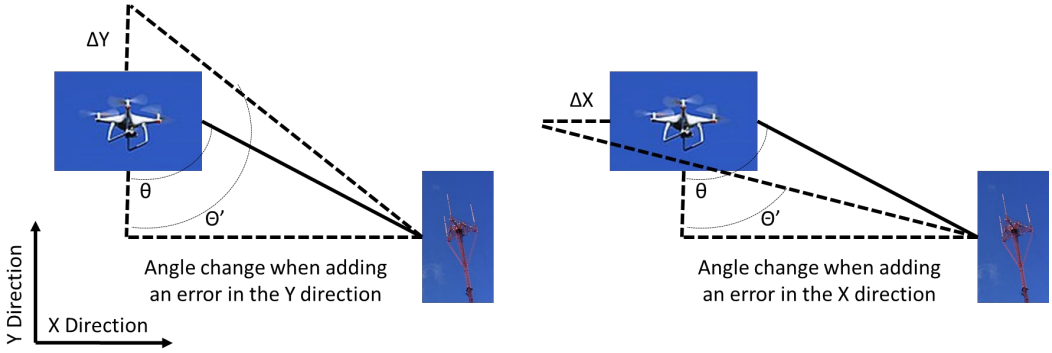

Figure D-7. Angle changes as errors are added in the X- and Y-directions to the sensor location

When adding an uncertainty equal to 5 m to the sensor's location in the X-direction, the new angle (θ') becomes equal to $tan^{-1}\left(\frac{1000+5}{100}\right) \cong 84.32^o$. When adding that uncertainty to the Y-direction, then $tan^{-1}\left(\frac{1000}{100+5}\right) \cong 84.01^o$. The starting angle is equal to $tan^{-1}\left(\frac{1000}{100}\right) \cong 84.29^o$. Thus the change in angle is larger when uncertainty is added in the Y-direction than in the X-direction. It makes sense that, in general, the TLE should be larger when adding an uncertainty in the sensor location in the Y-direction than when adding it to the X-direction, but does it make sense that it is about 10× larger?
Figure D-8 shows the error in the emitter location that is incurred because of an uncertainty in an angle measurement.

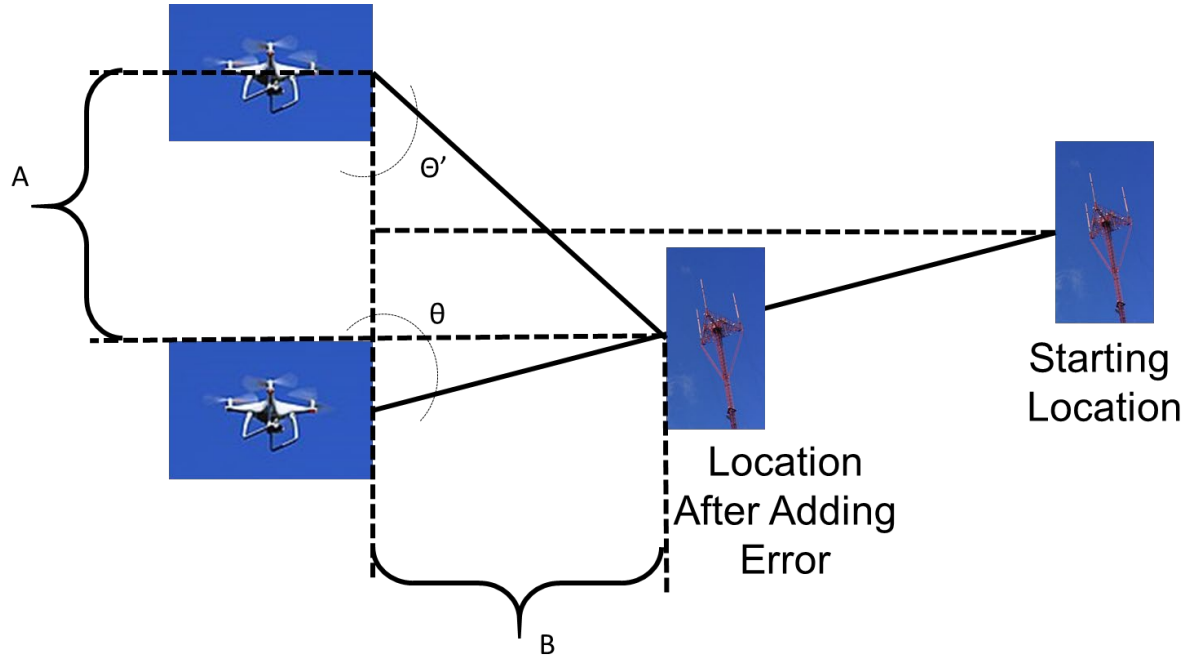

Figure D-8. Error in the measured emitter location after adding an error to the angle measured by one sensor

As seen in Figure D-8, variable A is defined as being equal to the distance in the Y-direction from the erroneous location of the emitter and the location of the sensor whose angle measurement has an uncertainty. Variable B is this distance in the X-direction. This means that $\tan(\theta') = \frac{B}{A}$. Likewise, $\tan(\theta) = \frac{B}{s-A}$ where, s is the separation between the sensors. Solving these equations for *A* and *B* will give Equations (D-8) and (D-9):

$$A = \frac{s}{1+\frac{tan\,(\theta')}{tan\,(\theta)}}, \tag{D-8}$$

$$B = A\,tan\,(\theta'). \tag{D-9}$$

Plugging in the angles found for an uncertainty in the X-direction leads to $A \cong 99.75$ and $B \cong 1002.5$. For an uncertainty in the Y direction, $A \cong 102.44$ and $B \cong 975.6$. Based on this back-of-the-envelope calculation, it is possible to estimate that when adding an uncertainty of approximately 5 m to the location of a sensor in the X-direction, the error in the emitter location is around 0.25 in the Y-direction and around 2.5 in the X-direction. When adding this uncertainty to sensor location in the Y-direction, the target location error is around 2.44 in the Y-direction and around 24.4 in the X-direction.

Using these calculations, it can be concluded that the smallest TLE would be the TLE in the Y-direction when an uncertainty is added to the sensor location in the X-direction. The TLE in the X-direction when adding an uncertainty to the sensor location in the X-direction is around 10× larger than the TLE in the Y-direction. When the uncertainty is added to the Y-direction the TLE in both the X-and Y-directions increases further. This means that the TLE in the Y-direction after adding an uncertainty in the Y-direction is approximately the same as TLE in the X-direction when an uncertainty is added to the X-direction. However, the TLE in X-direction after adding the uncertainty to the Y-direction is a further 10× larger than the TLE in the Y-direction. These back-of-the-envelope calculations help to validate the TLE values that the algorithm calculates when an uncertainty is added to a reference asset's location.

Thus far, the tests demonstrate how program can model uncertainties with a normal distribution. However, is it possible to model uncertainties with different distributions? As previously mentioned, photon counting detectors follow a Poisson distribution. The two key differences between normal and Poisson distributions are that in Poisson distributions, the standard deviation is equal to the square root of the mean and the probability density functions are different.

The software the MUSTC algorithm uses to calculate the probability density function for Poisson distributions is slower than for normal distributions when the mean, and therefore the standard deviation, becomes large. Fortunately, as the mean increases, the Poisson density function should more closely match a normal distribution density function. In response, for large means, the MUSTC function will switch to a normal distribution even when modeling an uncertainty that matches a Poisson distribution. How large does the mean have to become before it should switch to a normal distribution? Can this functionality be used to highlight the difference between values when using a Poisson versus a normal distribution?

To answer this, a new test scenario is needed. For this test, all uncertainties were set to zero except the uncertainty in the sensor's ability to count the photons. The program was then allowed to model how uncertainties in the number of photons the detector counts can affect the TLE.

Since the standard deviation is the square root of the mean for Poisson distributions, the only way to change the standard deviation is to increase the mean number of photons detected. By setting a minimum mean count value for the program to switch from Poisson to normal distribution, for each mean count value the program was forced to use Poisson and then normal distribution. The program will not use a normal distribution if the photon count is less than 16. This is because the program will calculate the error when the measure goes from –4× the standard deviation to +4× the standard deviation from the mean. If the mean counts were less than 16, then –4× the standard deviation from the mean would be less than zero. Because the program cannot handle negative photon counts, such an exercise would not make sense.

Figure D-9 shows the uncertainty in location in a single direction (the uncertainty in location in the other direction show the same results) as a function of the mean number of counts the photon counter would measure. The mean number of counts was varied by changing the integration time.

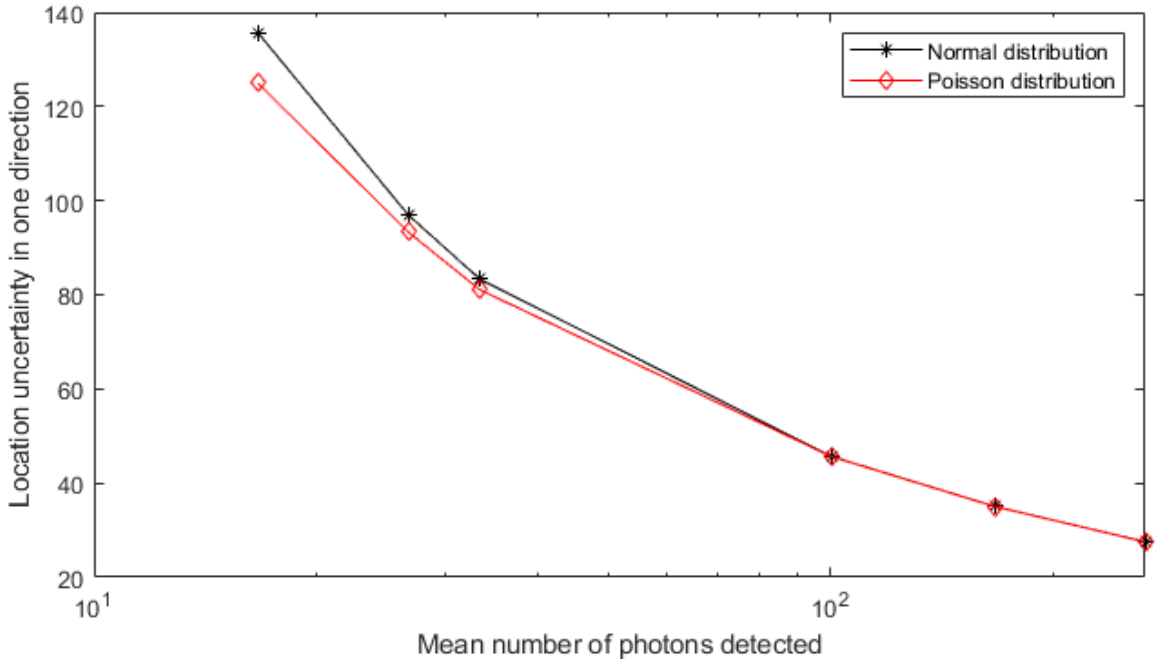

Figure D-9. Uncertainty in location caused by the uncertainty in using photon detector sensors to find the emitter location as a function of the mean number of photons the detector would measure

As one can see in Figure D-9, when the mean count is set to around 16 (the smallest mean number of photons detected that was plotted on the graph), the program calculates a lower TLE when using the Poisson distribution than when using normal distribution. As the mean number of counts increases, the value using a Poisson distribution becomes closer to the value the program gets when using a normal

distribution. When the mean number of counts becomes equal to 100 or more, the program can safely use a normal distribution instead of a Poisson distribution and still get the same TLE estimation. The behavior shown in Figure D-9 is expected and is evidence that the program is working.

Although it is possible that the MUSTC program still contains errors, the results from this Appendix D add confidence that the program is working correctly. In the future and as time allows, it is recommended that additional algorithm testing be performed to ensure calculations are still being performed as expected.

LIST OF ACRONYMS

2-/3-D	two-/three-dimensional
AOA	angle of arrival
CEP	circular error probability
CEP50	50% Circular Error Probability
COMINT	Communications Intelligence
DAC	DEVCOM Analysis Center
dB	decibel
DEVCOM	U.S. Army Combat Capabilities Development Command
DLL	Dynamic Linked Library
DOD	Department of Defense
ELINT	Electronic Intelligence
EO/IR	electro-optical/infrared
FDOA	frequency difference of arrival
GPS	global positioning system
HPC	high-performance computing
INS	inertial navigation system
LRF	laser range finder
MIT	Massachusetts Institute of Technology
MUSTC	Multipurpose Universal Simplified TLE Calculator
PNT	position, navigation, and timing
RF	radio frequency
SIGINT	Signals Intelligence
SNR	signal-to-noise ratio
TDOA	time difference of arrival
TLE	target location error
V&V	verification and validation

ORGANIZATION

DEVCOM Analysis Center
FCDD-DAE-I/M. Banta
6896 Mauchly St.
Aberdeen Proving Ground, MD 21005-5071

DEVCOM Army Research Laboratory
FCDD-RLD-DCI/Tech Library
2800 Powder Mill Rd.
Adelphi, MD 20783-1138

Defense Technical Information Center
ATTN: DTIC-O
8725 John J. Kingman Rd.
Fort Belvoir, VA 22060-6218