A COMPARISON OF TWO STOCHASTIC GLOBAL OPTIMIZATION METHODS FOR THE GENERATION OF ELECTRONIC COUNTERMEASURES TECHNIQUES

UNE COMPARAISON DE DEUX MÉTHODES STOCHASTIQUES D'OPTIMISATION GLOBALE POUR LA PRODUCTION DE TECHNIQUES DE CONTRE-MESURES ÉLECTRONIQUES

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Dean Theodore Vogelsang, CD, MSc, BEng, rmc
Captain

In Partial Fulfillment of the Requirements for the Degree of
Masters of Applied Science in Electrical Engineering

September, 2019

To my wife, Susan, whose unwavering support over the last two years made this work possible. Her strength, humour, and inspiration are what pushed me across the finish line.

Acknowledgements

This thesis was made possible through the unwavering support and guidance of my supervisor, Dr. Joey Bray. His expertise in the fields of electromagnetics and radar were complemented by an uncanny ability to provide just the right balance of motivation and praise to keep me moving forward. I would also like to thank Dr. Vincent Roberge, whose instruction and troubleshooting in the areas of global optimization and parallelization was crucial to the success of this work. Also deserving of my thanks is Major Randy Hartmann, whose experience in the field of airborne electronic warfare was vital to my understanding of key concepts. Finally, I would like to thank the Royal Canadian Air Force Aerospace Warfare Centre, whose sponsorship made this thesis possible.

Abstract

Choosing the optimal set of parameters of an electronic countermeasures technique from the vast solution space provided by modern deception jamming systems is time consuming and non-trivial. Optimization algorithms can be used to find the optimal parameter set to a given problem in a fraction of the time required of direct-search methods. Both the genetic algorithm and particle swarm optimization have been shown to be effective when dealing with electromagnetic engineering problems. Previous attempts to improve electronic countermeasures techniques have used a genetic algorithm in a limited fashion to generate range gate pull-off and velocity gate pull-off techniques using a hardware-in-the-loop simulation. In the public domain, the particle swarm optimization has never been used for this specific problem.

This thesis compares the effectiveness and efficiency of the genetic algorithm and the particle swarm optimization when applied to the problem of electronic countermeasure technique parameter selection. To do so, the MATLAB® Global Optimization Toolbox and Tactical Engagement Simulation Software (TESS™) were integrated to provide a fitness evaluation of each candidate solution generated via the iterative process. Multiple optimizations were conducted for engagement scenarios between a ground-based radio-frequency command-guided surface-to-air missile system and an airborne target aircraft using a self-protection deception jammer. Simulation results show that effective electronic countermeasures deception jamming techniques can be generated using both optimization algorithms. However, the particle swarm optimization found effective techniques more often and in less time than the genetic algorithm.

Résumé

Les systèmes modernes de brouillage déceptif exigent plusieurs paramètres d'entrée. L'optimisation des paramètres d'une technique de contre-mesure électronique s'avère alors d'une tâche longue et difficile. Des algorithmes d'optimisation sont souvent plus rapides que la méthode de recherche directe pour estimer la solution optimale d'un problème à paramètres multiples. L'algorithme génétique et l'optimisation des essaims particulaires se sont révélés efficaces pour traiter certains problèmes de génie électromagnétique. Des tentatives précédentes visant à améliorer les techniques de contre-mesures électroniques ont utilisé l'algorithme génétique de manière limitée pour générer des techniques de déréglage des portes de distance et de vitesse dans une simulation de matériel incorporé. Cependant, les sources publiées ne mentionnent pas l'optimisation des essaims particulaires pour cette tâche.

Dans cette thèse, l'efficacité et l'efficience de l'algorithme génétique et de l'optimisation des essaims particulaires sont comparées pour l'estimation des paramètres optimaux des techniques de brouillage déceptif. Pour ce faire, la boîte à outils d'optimisation globale MATLAB® et le logiciel de simulation d'engagement tactique (TESS™) ont été intégrés pour évaluer les solutions candidates générées lors des itérations. De multiples optimisations ont été effectuées pour des engagements simulés entre un système de missile sol-air guidé par commande radiofréquences et un aéronef cible auto-protégé par un brouilleur de déception. Les résultats démontrent que les deux algorithmes d'optimisation sont capables de générer des techniques de brouillage déceptif efficaces. Cependant, l'optimisation des essaims particulaires a identifié des techniques efficaces plus souvent et plus rapidement que l'algorithme génétique.

Contents

List of Tables

List of Figures

List of Acronyms

CG	Command-Guided
CPU	Central Processing Unit
dB	Decibel
dBW	Decibel Watt
DECM	Defensive Electronic Countermeasures
ECCM	Electronic Counter-Countermeasures
ECM	Electronic Countermeasures
GA	Genetic Algorithm
GPU	Graphics Processing Unit
GUI	Graphical User Interface
JSR	Jamming-to-Signal Ratio
LORO	Lobe-On-Receive-Only
PRI	Pulse Repetition Interval
PSO	Particle Swarm Optimization
PW	Pulse Width
RAM	Random Access Memory
RCS	Radar Cross Section
RF	Radio Frequency
RGPI	Range Gate Pull-In
RGPO	Range Gate Pull-Off
RMC	Royal Military College of Canada
SAM	Surface-to-Air Missile
SAMCGAAA	Command Guided Surface-to-Air Missiles and Anti-Aircraft Artillery

SWC	Scan-With-Compensation
TELAR	Transporter Erector Launcher And Radar
TTI	Tactical Technologies Incorporated
TTR	Target Tracking Radar
TESS™	Tactical Engagement Simulation Software™
VGPO	Velocity Gate Pull-Off

1 Introduction

1.1 Background

The search for methods with which to deceive or defeat threat radar systems has been ongoing since World War II when chaff (referred to as window) was dropped to jam and spoof radar and navigation signals [1]. Technological advances have resulted in progress on both fronts, leading to a constant cycle of innovation. As radar systems have advanced, so too have the electronic countermeasures (ECM) designed to defeat them. Today, the most important aspects of the ECM development process are the acquisition of information about threat systems and the speed at which that information can be exploited. Although technology has provided greater flexibility in the production and use of highly effective ECM techniques, a vast solution space also exists from which to find suitable techniques.

The manual development of the jamming waveforms for ECM techniques can take considerable time and is often non-trivial in terms of parameter selection and technique validation. The setup, configuration, and evaluation of a particular system's circuit design and operation can require a large number of resources, including test sets, test facilities, and indoor/outdoor range space and time allocation. ECM waveforms are described by a large number of independent parameters that must be carefully selected and tuned to maximize the overall effectiveness against a threat radar system. Although it is possible to select these parameters based on experience regarding the applicable systems along with a trial and error approach, such a method is by no means optimal and is susceptible to

human error and the availability of suitable personnel. In theory, the use of signal-processing platforms designed to use evolutionary heuristic search methods and integrated with existing ECM test systems could offer an alternative approach to the development of optimized deception jamming waveforms and ECM techniques.

Evolutionary heuristics such as the genetic algorithm (GA) and the particle swarm optimization (PSO) have been proven effective across multiple disciplines when applied to complex, multi-objective optimization problems, and have been demonstrated to be more efficient than other more exhaustive search algorithms [2]–[4]. Each optimization technique is based on completely different philosophies, leading to strengths and weaknesses that are unique to each. Recent work has demonstrated that the GA and the PSO can be successfully applied to electromagnetic applications such as antenna design, radiation patterns, and waveform optimization [5]–[7]. Although there is no evidence (in the public domain) that the PSO has been applied to ECM technique development, the GA has been used recently to select optimal ECM parameters in a laboratory hardware-in-the-loop ECM simulation environment using MATLAB® and the Lab-Volt™ Radar Training System [8]. However, the GA was found to be time-consuming due to its serial implementation in MATLAB® [8]. Furthermore, the simulations used limited target motion profiles such as constant velocity, constant acceleration, and linear acceleration. The profiles did not address more realistic flight paths of a target such as changes from constant velocity to linear acceleration, nor did they use both range and velocity techniques concurrently for false target generation.

Other, more computationally simplistic stochastic global optimization methods, such as the PSO, which have not been used for ECM optimization, might be faster and more effective. The relative performance of the GA and the PSO should be compared based on their computational time and the effectiveness of the solutions they produce.

The computer-based simulation of threat and target engagements using a commercial off-the-shelf product such as the Tactical Engagement Simulation Software™ (TESS) [9] developed by Tactical Technologies Inc. (TTI) could prove to be useful for optimizing ECM waveforms in more realistic scenarios. The TESS™ product, being MATLAB®/Simulink® based, is an ideal platform to use as an evaluation tool.

1.2 Problem Statement

The sophisticated waveforms provided by modern jammer systems imply that a vast solution space exists from which to find optimal ECM techniques. Manual searches for effective techniques can be time consuming and non-trivial in nature. Although previous research has shown the feasibility of using the GA to develop ECM techniques, the work focussed on a hardware-in-the-loop simulation using a radar system that was scaled-down to a laboratory environment and which used non-manoeuvring target profiles. No publicly published work has documented the use of evolutionary heuristics, such as the GA or PSO, to optimize ECM techniques for more realistic threat and target platforms. The GA and PSO have the potential to reduce the workload associated with ECM technique development and may also lead to the discovery of new, previously unidentified techniques.

1.3 Thesis Statement

Effective electronic countermeasures deception jamming techniques can be generated by using evolutionary heuristics such as the genetic algorithm and the particle swarm optimization. A comparison of both methods is required to determine the best optimization method in terms of ECM effectiveness and efficiency.

1.4 Scope

Two evolutionary heuristics, specifically the GA and the PSO, will be used for global optimization. The study is limited to computer-based simulations using the software products MATLAB®/Simulink® and TESS™. Simulated engagements will be restricted to command-guided (CG) surface-to-air missile (SAM) threat systems with pulsed target tracking radars (TTRs) and either fighter or rotary-wing aircraft using a self-protection jammer. Only ECM deception jamming techniques that provide false range and velocity information will be generated.

1.5 Methodology

Building upon concepts previously explored using the GA [8], but using a purely software simulation, ECM technique candidate solutions will be generated through the global optimization algorithms GA and PSO and evaluated through simulated engagements between a representative threat system and a defensive target platform. Each optimization algorithm will be assessed based on effectiveness (i.e. how close the algorithm approaches the defined global optimum solution) and efficiency (i.e. the computational effort and time required for the algorithm to converge to the defined global optimum solution).

1.5.1 Software Integration

The first requirement is to integrate the MATLAB® Global Optimization Toolbox™, specifically, the GA and PSO functions, with the proprietary TESS™ product. This step includes configuring a two-way transfer of data; the candidate solution values generated by the optimization algorithm must be transferred to TESS™ as jammer parameters and the engagement simulation results must be returned to the optimization algorithm to compute a fitness score.

1.5.2 Fitness Function

The *fitness function* is a user-defined mathematical expression, or algorithm, that evaluates the effectiveness of a candidate solution (e.g. an ECM technique). The fitness function must be carefully chosen because it defines the metrics against which effectiveness is evaluated, and thereby influences the evolution of the optimization algorithms. TESS™ generates *effectiveness measures* in the form of scalar output variables for each engagement simulation including: missile miss distance, probability of kill, and probability of survival. In addition, a number of time-series output variables that characterize the engagement are generated, including: threat radar boresight angles, waveform power values (e.g. target and covering pulse amplitudes at the receiver), radar mode (i.e. search, acquisition, and track), and missile parameters. The fitness function may use some, or all of these variables as inputs to evaluate effectiveness.

1.5.3 Parallelization

Although a single TESS™ engagement simulation takes less than a minute to complete (depending on the scenario setup), the large number of candidate solutions imposes a significant time expense on the convergence rate of each algorithm. Parallelization using the MATLAB® Parallel Computing Toolbox™ is essential to reduce the total execution time of each optimization algorithm by conducting multiple engagement simulations concurrently rather than sequentially.

1.5.4 Simulation, Analysis, and Comparison

The TESS™ Air RF Master Interface provides a wide range of customization for engagement simulations, including threat and defensive platform selection, engagement flightpath characteristics, and jammer settings. The simulated threat system will be based on a realistic CG surface-to-air weapon system. A limited

number of engagement scenarios will be defined and simulated; variables such as target altitude, airspeed, and approach angle relative to the threat will be held constant. Simulations will also involve modifications to the optimization parameters of each algorithm to explore the suitability of each algorithm in dealing with the problem of ECM technique generation and to compare the two algorithms in their performance.

1.6 Risks and Mitigations

A number of risks have been identified that could influence the successful generation of ECM techniques through global optimization methods. First, the software integration of the MATLAB® Global Optimization Toolbox™ with the proprietary TESS™ product is not a trivial task, as TESS™ has not been used in this context before. Fortunately, as previously mentioned, the fact that TESS™ is based on MATLAB®/Simulink® suggests the integration is feasible.

The process of parallelization, which could significantly decrease the execution time for the optimization process, poses additional risks. TESS™ may require licensing for multiple instances of the program to be run simultaneously. Furthermore, the integration of TESS™ with the Parallel Computing Toolbox™ has not been done before. TESS™ does support multicore computing, which indicates that parallel computing should be possible.

The limited public availability of threat and jammer system specifications for definition in TESS™ poses another risk, since the TESS™ interface requires a number of system parameters to define a system model. These parameters include transmitter and receiver specifications, waveform operating characteristics, and physical features. Fortunately, TESS™ comes pre-loaded with a limited number of default threat, target platform, and jammer systems that should be sufficient for use in engagement scenarios.

Finally, the large number of optimization parameters and simulation variables that can be modified for each algorithm threaten to increase the scope, and consequently, the time required to collect sufficient data for a comparison of each algorithm. To assess the effect of a single algorithm option change or simulation variable may require multiple optimization runs, which imposes a time constraint risk for this thesis. Only a limited number of simulation variables should be modified between engagement scenarios; the majority of optimization parameters should be held at their default values with only a limited number modified to improve the optimization process, if required.

1.7 Thesis Outline

Chapter 2 provides a high-level overview of ECM deception jamming techniques, focussing on the general concepts of jamming and more specifically on the deception techniques used in this work. Additionally, a brief overview of the ECM waveform parameters is provided. The global optimization algorithms, GA and PSO, are introduced, along with the MATLAB® Global Optimization Toolbox™ [10] in which these algorithms are implemented. Finally, the TESS™ software product is introduced.

Chapter 3 describes the initial validation conducted on the GA and PSO algorithms using a series of test functions to analyze algorithm setup, convergence rates, and their suitability for application to the ECM technique generation problem. The GA and PSO implementations in MATLAB® are also reviewed and compared.

Chapter 4 presents the design process used in the development of the simulation system, including the software architecture chosen for the implementation. The integration of TESS™ with the MATLAB® Global Optimization Toolbox™ is explained, and each of the program blocks required for system operation are described. The requirements for and integration of the

MATLAB® Parallel Computing Toolbox™ are discussed. Finally, the design of the deception jamming techniques, in the context of the TESS™ jammer implementation, is covered.

Chapter 5 presents the results of the software simulations. First, the selection of optimization options and solution space bounds used during the simulation process is discussed. Engagement scenario design and simulation conditions are also examined. The optimization results are then presented and analyzed, with a focus on the effects of each deception technique parameter, or optimization variable, on the performance of the generated ECM techniques against the threat system and their resulting fitness. The performance of the two optimization algorithms is compared, concentrating on the complexity of the algorithm, its convergence speed, and the generation of suitable ECM techniques by comparing the fitness scores.

Chapter 6 concludes the thesis and discusses areas of future research for the use of global optimization algorithms in the development of ECM techniques.

2 Literature Review

2.1 Electronic Countermeasures

ECM, in the context of the radio frequency (RF) spectrum, can be defined as the systems, tactics, and techniques used to interfere with the operation of radars through the denial of information sought by the radar or to surround the desired radar returns with so many false targets that the true information is unresolvable [11], [12]. ECM can be classified as either active or passive and includes both noise and deception jamming [12].

2.1.1 Jamming

The intentional transmission or re-transmission of amplitude, frequency, phase, or otherwise modulated intermittent or continuous RF signals to interfere with, exploit, deceive, mask, or degrade the reception of radar returns is called jamming [12]. A jammer can interfere with a victim radar either by injecting artificial noise or deceptive signals into the receiver. The strength of the jamming signal is normally quantified as the ratio of the effective jammer power, (the jammer signal power at the radar receiver) to the signal power, (the original target echoes the victim radar wants to receive), otherwise known as the jamming-to-signal ratio, JSR [11]. For a monostatic radar tracking a target with a self-protection jammer, the power received at the victim radar from the jammer, J, may be represented by the one-way radar range equation [13]:

$$J = \frac{P_j G_j G_r \lambda^2}{(4\pi)^2 R^2} \tag{2.1}$$

where

P_j is the peak transmitted power from the jammer in watts;

G_j is the gain of the transmit antenna of the jammer;

G_r is the gain of the receive antenna of the victim radar;

λ is the wavelength of the transmitted signal in meters; and

R is the range from the target platform/jammer to the victim radar in meters.

The power of the target echo received at the victim radar, S, may be represented by the two-way radar range equation [13]:

$$S = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R^4} \tag{2.2}$$

where

P_t is the peak transmitted power from the radar in watts;

G_t is the gain of the transmit antenna of the radar; and

σ is the radar cross section (RCS) of the target in square meters.

The JSR can then be derived by combining (2.4) and (2.2) to give:

$$JSR = \frac{P_j G_j 4\pi R^2}{P_t G_t \sigma}. \tag{2.3}$$

The JSR is usually specified in decibels (dB), which is obtained as follows:

$$JSR_{dB} = 10log(JSR). \tag{2.4}$$

The range at which the jammer power equals the signal power is called the crossover range, $R_{J=S}$, given by [13]:

$$R_{J=S} = \sqrt{\frac{P_t G_t \sigma}{P_j G_j 4\pi}}. \tag{2.5}$$

The distance from the tracking radar to the target at which adequate desired signal strength is available to track the target, despite the jammer signal, is called the burn-through range, R_{BT} [13]. Although burn-through may occur at the crossover range, effective jamming usually requires a JSR_{dB} that is greater than 0 dB, called JSR_{min} (in linear form). The burn-through range, R_{BT}, is therefore given by [13]:

$$R_{BT} = \sqrt{\frac{P_t G_t \sigma JSR_{min}}{P_j G_j 4\pi}}. \tag{2.6}$$

In spite of jamming, a target is detectable by the radar when the target's range is less than R_{BT}.

Equation (2.6) can be used in the design of ECM techniques when the victim radar characteristics are known. Often, radar systems may be designed with higher than necessary power to increase the range at which burn-through occurs against jamming targets [13]. Accurate threat system intelligence can enable selection of the jammer power, P_j, in order to decrease the burn-through range.

Deception jamming is a more advanced type of jamming. Whereas traditional jamming involves the active transmission of noise, deception jamming uses sophisticated waveforms designed to provide false range or velocity information to a victim radar system [11]. Deception jammers are more sophisticated than noise jammers since the waveform parameters are directly related to the performance parameters and modes of operation of the victim system. As a result, deception jammers consist of more complex hardware and software to generate the desired signals. When successful, the victim radar improperly accepts the jamming signal as a target with a false range, velocity, or angle. Prime examples of deception jamming against tracking radars include the range gate pull-off (RGPO), range gate pull-in (RGPI) and the velocity gate pull-off (VGPO).

2.1.2 Range Gate Pull-Off

A tracking radar maintains and updates estimates of a target's range, velocity, and/or angle. These estimates are often bounded by a narrow window of values known for historical reasons as a *gate*. Typically, a radar tracks a target in range through the use of early and late gates. The gates are moved in unison to constantly equalize the energy of a return pulse between the early and late gate, which centers the target between the gates. During an RGPO attack, the jammer transmits a false return pulse similar to the reflected pulse that the target would produce (i.e. its *skin return*), but at a higher power, which captures the radar's range gate. The transmission of the jammer pulse is then delayed by a gradually increasing amount, as depicted in Figure 2.1 [14]. By capturing the range gates of the victim radar with a stronger delayed pulse, the gates are pulled away from the true radar return, as shown in Figure 2.2 [14]. The increase in power in the late gate causes the victim radar, which determines range to the target based on the arrival time of reflected pulses, to incorrectly calculate a range to target that is greater than the true target range [11], [13], [14]. Increasing the delay time either parabolically or exponentially will make it appear to the victim radar that the target is turning away from the radar [14].

Careful consideration is required when determining the speed at which the range gates will be pulled away from the target. If the rate of pull-off exceeds the tracking rate of the victim radar, the jamming will be ineffective. Selection of the pull-off rate requires either technical knowledge of the victim radar or consideration of the task the radar was designed to carry out [11]. The use of an improper pull-off rate may also lead some radar systems to detect the presence of jamming leading to the use of electronic counter-countermeasures (ECCM).

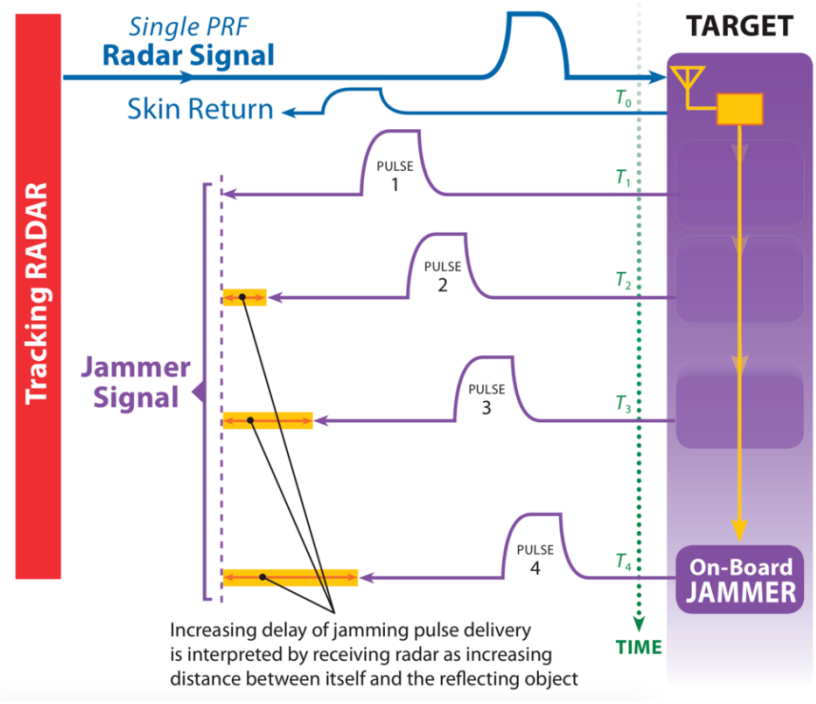


Figure 2.1: RGPO jammer operation (reproduced from [14])

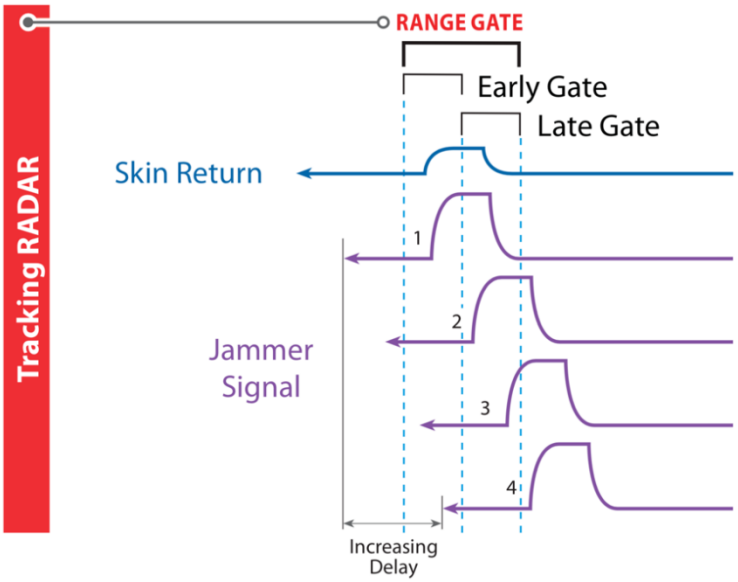


Figure 2.2: Delayed, amplified pulses of RGPO (reproduced from [14])

2.1.3 Range Gate Pull-In

A similar form of range deception involves tracking the pulse repetition frequency of the victim radar to anticipate the arrival time of each pulse and transmitting subsequent false return pulses at a higher power and in advance of the true skin return, as shown in Figure 2.3 [14]. By leading the true skin returns by an increasing amount with each jamming pulse, the RGPI makes it appear as though the target is turning toward the radar. The increase in power in the early gate of the victim radar causes the victim radar to pull its range estimate away from the target.

Since the timing of future pulses must be calculated, RGPI is effective only against tracking radars with fixed or staggered pulse repetition intervals (PRI) [14]. Radars that use randomly jittered PRI are not susceptible to RGPI jamming since the PRI cannot be predicted.

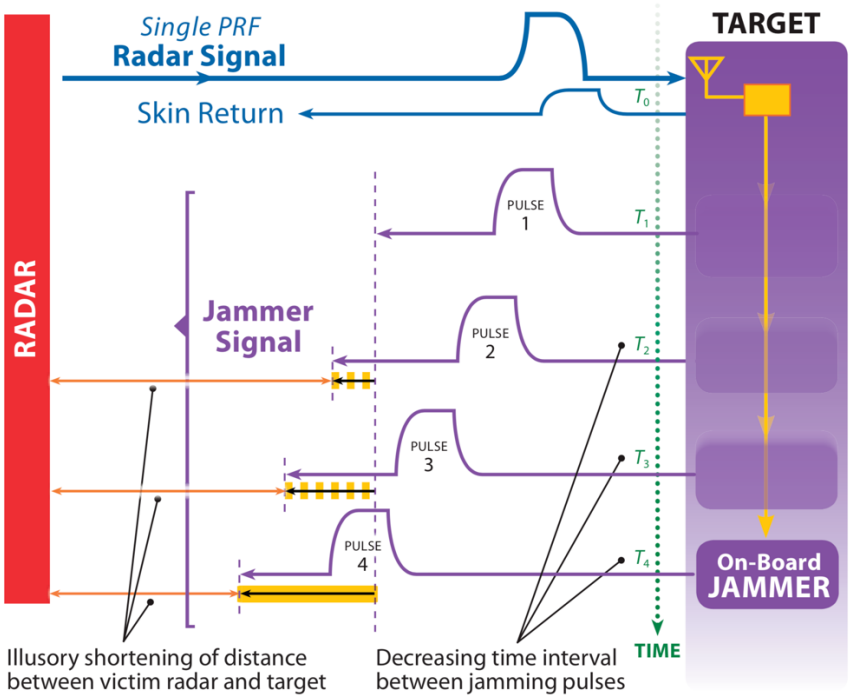


Figure 2.3: RGPI jammer operation (reproduced from [14])

2.1.4 Velocity Gate Pull-Off

Radar targets reflect the transmitted energy back at a shifted frequency proportional to the relative radial speed between the radar and the reflecting object. The frequency offset, known as the *Doppler frequency*, f_d, is given by [14]:

$$f_d = \frac{2 f_{Tx} v_t}{c} \tag{2.7}$$

where f_{Tx} is the transmitter frequency, v_t is the radial target velocity relative to the receiver, and c is the speed of light. A negative frequency corresponds to a target receding from the receiver (i.e. a negative velocity) whereas a positive frequency corresponds to a target closing on the receiver (i.e. a positive velocity). The Doppler frequency is the basis for the target radial velocity estimation that is performed by both continuous wave and pulse-Doppler (coherent) tracking radars.

A frequency filter, or velocity gate, isolates the desired target return based on its frequency shift, corresponding to a relative velocity. The VGPO jamming technique generates a false radar return with the same frequency offset as that of the target, but at a higher power to capture the gate [11], [13]. The false return is then swept away from the frequency of the true target return, breaking the victim radar's velocity track, as shown in Figure 2.4. As with RGPO, the rate of pull-off is an important consideration because the radar tracking circuitry will be designed to track only up to the maximum rate of change in velocity of a known class of targets.

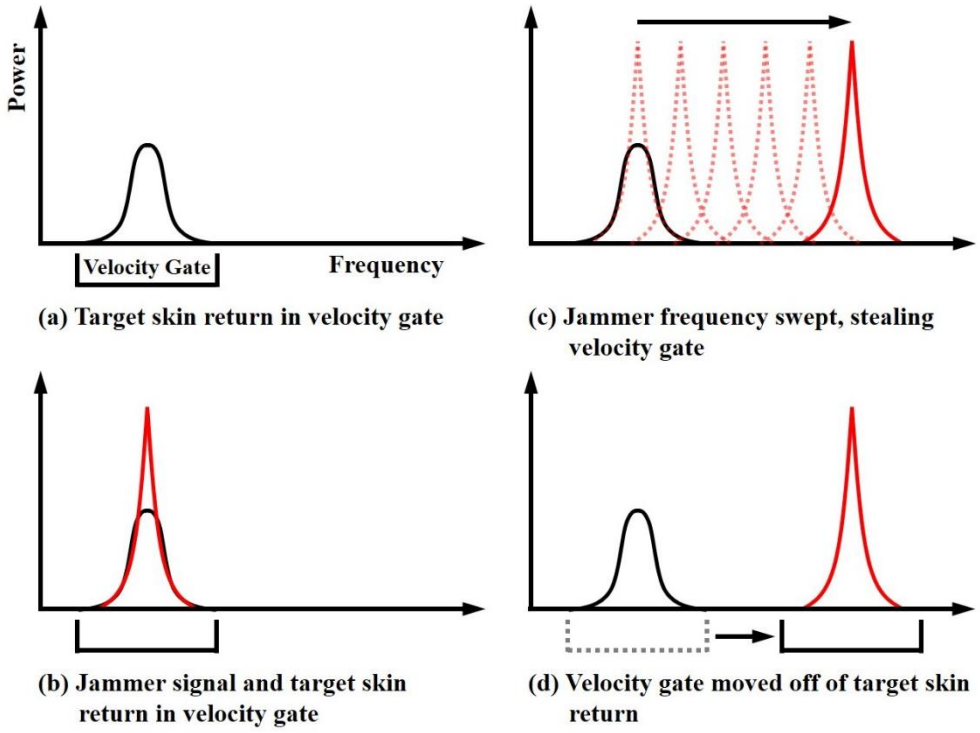


Figure 2.4: VGPO jammer operation [11]

Use of a single range or frequency deception jamming technique is not sufficient to deceive a radar system that tracks in both range and velocity, as shown in Figure 2.5. Therefore, deception jamming techniques must attack both range and velocity gates to be effective against coherent radar systems. A jamming technique that targets both range and velocity (frequency) is said to be a *coordinated attack*.

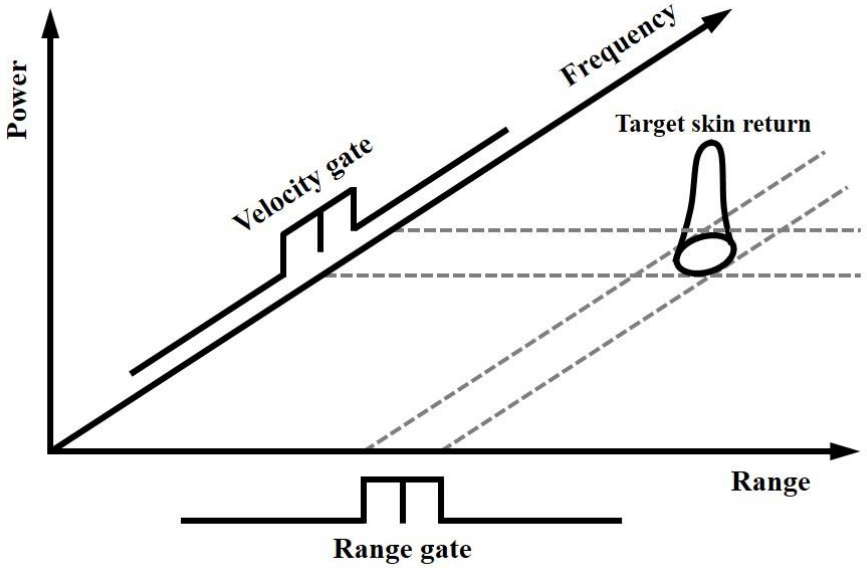


Figure 2.5: Range and velocity tracking gates of coherent pulse-Doppler radars

2.1.5 Technique Modelling and Parameter Selection

A deception jamming technique that creates a false target (i.e. RGPO, VGPO, or a combination of the two) has several parameters. The duration of the attack, during which the gate is walked-off of the true target, is referred to as the false target walk-off time. The initial distance between the true target and false target, referred to as initial delay, may be some value greater than or equal to zero, given in units of distance or time. Similarly, the maximum distance between the true and false targets is referred to as maximum delay. The rate at which the false target moves away from the true target may be defined as constant velocity, constant acceleration, or linear acceleration. Finally, the JSR must be defined for the walk-off, including the rate of amplitude increase and the maximum JSR value. Techniques may also include a dwell time at the beginning and/or end of the walk-off. Although many more parameters may be defined for a given technique, these seven parameters are the focus of this research.

2.1.6 Technique Scoring

Determining the effectiveness of a given ECM technique in simulation normally involves pitting a model of the target platform and its self-protection jammer against a modelled threat radar and weapon system. The effectiveness measures of such simulations are often given as scalar outputs in terms of: the likelihood that the weapon system would disable or destroy the target (i.e. probability of kill, P_{kill}), the likelihood that the target would survive the engagement (i.e. probability of survival, P_{surv}), and the shortest distance measured between the weapon system and target platform during an engagement (i.e. miss distance, D_{miss}). In addition to the programmed parameters of the ECM technique, a number of external factors can also affect the scoring, including target/threat relative orientation and distance, target manoeuvring, and relative velocity. The effectiveness or *fitness* of a given ECM technique can be calculated using the simulation output data as input to a function. This fitness function generates a scalar value between 0 and 1 representing the score for the ECM technique, where 0 is defined as the ideal solution and 1 is a completely ineffective technique.

2.2 The Genetic Algorithm

Relying on random variation and selection, evolutionary algorithms mimic nature's tendencies towards competition and innovation to solve optimization problems [2]. GAs, first proposed in the 1960's, are a class of evolutionary algorithms inspired by natural selection whereby a system learns and adapts to the surrounding environment [2]. GAs are set apart from other evolutionary algorithms by three distinguishing features [5]:

1. the representation of data is typically via bit-strings;
2. the probability of selection is proportional to the relative fitness of an individual; and,
3. the creation of new individuals is primarily performed through crossover between population members.

Exploitation of the above distinguishing properties can achieve parameter optimization; however, a clear understanding of the search environment is required. When the fitness function is properly defined such that suitable population members can be identified, the fitness function performs in much the same way as nature when selecting the fittest of a species.

2.2.1 Terminology

As the name would imply, the GA draws many parallels with that of biological genetics. The fundamental building blocks in biology, known as genes, are represented in the GA as data bit-strings, or a binary encoding of a parameter. An array of parameter values then forms a chromosome. For an *N*-dimensional optimization of N parameters (given by $p_1, p_2, p_3, \dots, p_N$), the chromosome would be defined as [5]:

$$chromosome = [p_1 p_2 p_3 \dots p_N]\,. \tag{2.8}$$

The parameters can be defined as discrete or continuous. Continuous parameters necessitate the application of limits (representing physical properties or other bounds) or the restriction of the parameters to a subset of possible values. The population takes the form of a matrix in which each row is represented by a chromosome [6]:

$$population = \begin{bmatrix} chrom_1 \\ chrom_2 \\ \vdots \\ chrom_N \end{bmatrix}. \tag{2.9}$$

A fitness function f takes each chromosome as an input and calculates the fitness associated with each one [6]:

$$f\left\{\begin{bmatrix} chrom_1 \\ chrom_2 \\ \vdots \\ chrom_N \end{bmatrix}\right\} = \begin{bmatrix} fitness_1 \\ fitness_2 \\ \vdots \\ fitness_N \end{bmatrix}. \tag{2.10}$$

Through natural selection, only the fittest members of a population may survive, which is accomplished in the GA via one of two main methods. One involves sorting the population by fitness and then discarding all but a certain number of members. Another sets a threshold fitness and discards all those members that fail to meet the threshold fitness value. After selection for survival, only some members of the population will be selected for mating. The selection of mates can be carried out via either a roulette wheel or tournament. In the roulette wheel, also known as a proportionate selection, each chromosome is assigned a probability of selection based on its fitness. Conversely, tournament selection randomly divides the population into subsets and then selects the chromosome with the best fitness in each group to breed [6].

The generation of offspring, or new potential solution sets, is normally carried out by some form of crossover. Crossover operates on two of the selected parents, randomly selecting portions of each parent chromosome and splicing the

data together to form one or more offspring [2]. Chromosome mutation involves the replacement at random of a parameter, or portion of the chromosome with some other value during breeding. Through mutation, the algorithm continues to explore diverse areas of the solution-space [6].

2.2.2 Algorithm Description

The generalized flow chart of a GA is shown in Figure 2.6. The initial population, or starting matrix of chromosomes, is normally generated via a random guess at the optimal solution [6]. During algorithm design, the initial population size must be carefully considered, taking into account the desired computational complexity and the tendency towards premature convergence. A large population size will thoroughly explore the solution-space, but convergence to the desired end state will take longer. Smaller population sizes will perform a coarse search of the solution space and will tend to converge on local maxima/minima rather than the global maxima/minima. Selection then begins via the previously discussed means. Application of the fitness function will determine what intermediate population will move on to selection for the mating pool. Both the number of crossover points in the mating process along with the mating pairs are chosen via probabilistic processes. Careful definition of the crossover probability will help to prevent exact replication between population generations, which is normally undesirable. Random mutation within the offspring helps to prevent premature convergence; however, it should be used to alter only a small portion of the total population. For electromagnetic applications the mutation rate is normally accepted to be on the order of 0.1 to 1 % of all genes [5].

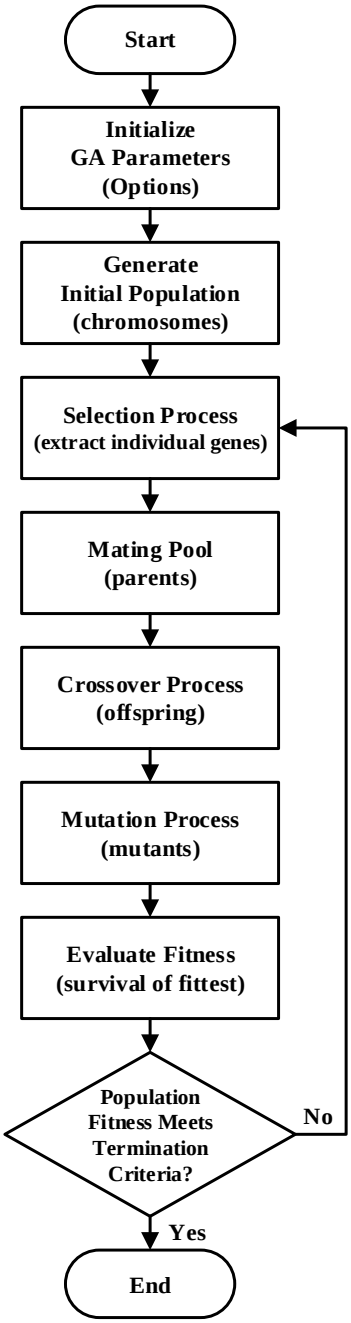


Figure 2.6: GA flowchart [15]

Once the offspring have been created, the fitness function is applied to determine how well each offspring satisfies the conditions for optimization. The assignment of fitness values follows, after which the offspring and parents are grouped as the current generation and a check for solution conditions completes the cycle. The exit criteria for the algorithm may be based on any number of different tests. Examples include a minimum average fitness, a best performance achieved, or some other tolerance level in which the majority of the chromosomes settle within a given error of one another. When the exit criteria are met, the optimized chromosome represents the desired solution.

2.3 The Particle Swarm Optimization

The PSO was first proposed in 1995 in an attempt to simulate the social behaviour and movement of flocking birds, schools of fish, or swarming bees [4], [16]. The initial swarm consists of a set of randomly generated candidate solutions which then propagate in the pre-defined solution space towards the optimal solution over a number of iterations [17]. The members of the swarm assimilate and share information about the solution space amongst each other through consecutive iterations. The inspiration for the PSO is “the ability of flocks of birds, schools of fish, and herds of animals to adapt to their environment, find rich sources of food, and avoid predators by implementing an “information sharing” approach, hence, developing an evolutionary advantage.” [17]

2.3.1 Algorithm Description

The principle of the algorithm is that each candidate solution may be represented by a particle in a swarm [4]. Each particle has a position and velocity vector, where the position coordinate represents a parameter value. For an *N*-dimensional optimization, each particle will have a position in *N*-dimensional space

representing a candidate solution [4], [16]. The PSO algorithm consists of three steps: generation of the particles' positions and velocities, velocity update, and position update. The generalized flow chart of the PSO algorithm is shown in Figure 2.7.

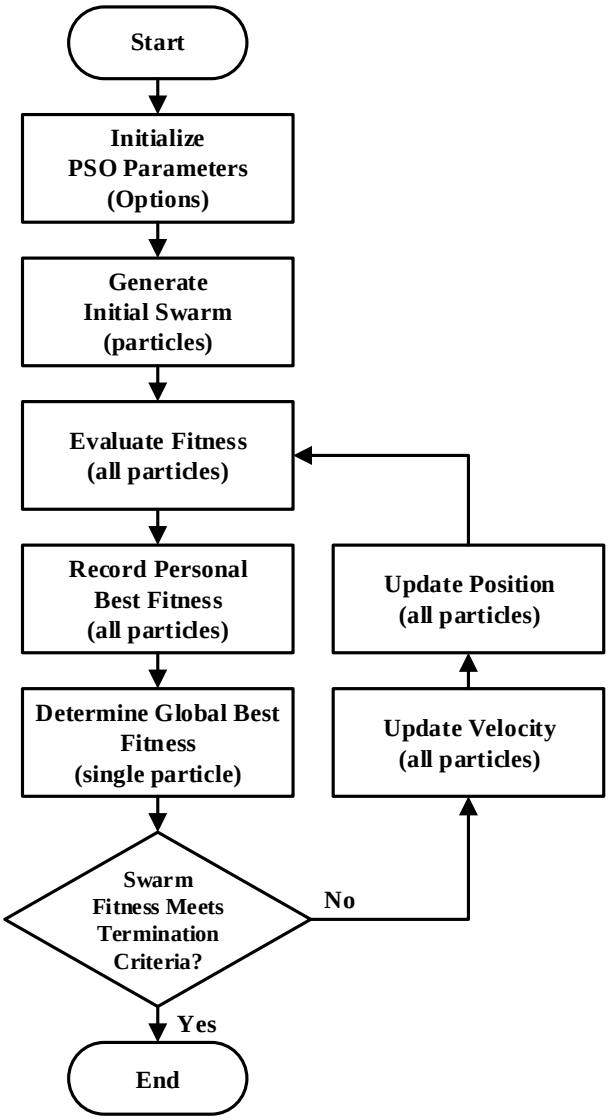


Figure 2.7: PSO algorithm flowchart [4]

The positions and velocities of the initial swarm of particles are usually randomly generated using upper and lower bounds $\mathbf{x}_{max}$ and $\mathbf{x}_{min}$ on the parameter values. For the i^{th} particle at time $k = 0$, the position $\mathbf{x}_k^i$ and velocity $\mathbf{v}_k^i$ are given by [17]:

$$\mathbf{x}_0^i = \mathbf{x}_{min} + rand_1(\mathbf{x}_{max} - \mathbf{x}_{min}) \tag{2.11}$$

and

$$\mathbf{v}_0^i = \frac{\mathbf{x}_{min} + rand_2(\mathbf{x}_{max} - \mathbf{x}_{min})}{\Delta t} = \frac{position}{time}. \tag{2.12}$$

The terms $rand_1$ and $rand_2$ are uniformly distributed random variables having any value between 0 and 1. The initialization process therefore ensures that the swarm is randomly distributed across the solution space.

The second step is to update the velocities of all particles at time $k + 1$ using the particles' fitness values, which are functions of the particles' current positions in the solution space at time k. The fitness value of a particle determines which particle has the best global value in the current swarm, $\mathbf{p}_k^g$, and also determines the best position of each particle over time, $\mathbf{p}^i$. The velocity update formula uses these two pieces of information for each particle in the swarm along with the effect of current motion, $\mathbf{v}_k^i$, to provide a search direction, $\mathbf{v}_{k+1}^i$, for the next iteration. The velocity update formula is given by [16], [17]:

$$\mathbf{v}_{k+1}^i = w\mathbf{v}_k^i + c_1 rand_3 \frac{(\mathbf{p}^i - \mathbf{x}_k^i)}{\Delta t} + c_2 rand_4 \frac{(\mathbf{p}_k^g - \mathbf{x}_k^i)}{\Delta t} \tag{2.13}$$

which includes two random parameters, represented by the uniformly distributed variables, $rand_3$ and $rand_4$, to ensure appropriate coverage of the solution space and avoid entrapment in local optima. The first term represents the current motion, the second term represents the particle memory influence, and the third term represents the swarm influence, all of which affect the new search direction. Three weight factors, namely, inertia factor, w, self-confidence factor, c_1, and swarm

confidence factor, c_2, control the rate of convergence of the algorithm. Although the original PSO algorithm used the values of 1, 2, and 2 for w, c_1, and c_2, respectively [16], [17], it has been suggested that setting them to 0.5, 1.5, and 1.5, respectively, provides the best convergence rate for a range of test problems considered [17].

Finally, each iteration requires a position update for each particle in the swarm, as depicted in Figure 2.8. The position update formula is given by [16], [17]:

$$\mathbf{x}_{k+1}^{i} = \mathbf{x}_{k}^{i} + \mathbf{v}_{k+1}^{i}\Delta t\,. \tag{2.14}$$

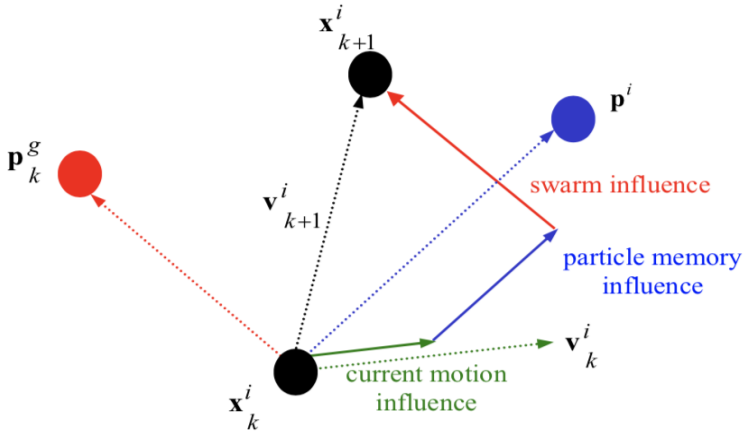


Figure 2.8: PSO velocity and position updates (reproduced from [17])

2.4 MATLAB® Global Optimization Toolbox™

The Global Optimization Toolbox™ comprises a series of functions that search for global solutions to single and multi-objective problems [10]. The toolbox provides options for algorithm behaviour, tolerances, and stopping criteria. In addition, intermediate results of an optimization can be accessed using output functions, including plotting functions. The toolbox allows the user to define the fitness function, which can use data from an external simulation. In this case, simulations within TESS™ can generate the engagement results used by the fitness function to generate a fitness score for each candidate solution.

Both the GA and PSO toolbox functions accept lower and upper bounds; these bounds limit the components of the solution **x** and can be used to obtain faster and more reliable solutions. For example, bounds can be used to restrict the minimum and maximum delay, velocity, and acceleration rate of a range deception technique pulse.

Detailed descriptions of the MATLAB® implementations of the GA and PSO may be found at [18] and [19], respectively, with summaries included in Appendix A.

2.5 Tactical Engagement Simulation Software™

TESS™ is a commercial physics-based software simulator for modelling guided missile engagements [20]. The Air RF Master Interface is specifically designed for RF guided systems and includes tools for simulating combinations of defensive countermeasures such as chaff and decoy deployment, active jamming, and platform manoeuvres in a realistic electromagnetic and physical environment. The interface provides for full customization of both the threat system and target platform, including waveform parameters of threat radars and target jammers. With a capability to perform either single engagements or sequential batch runs, the

interface captures and stores the miss distance, probability of kill, probability of survival, and radar mode (as a percentage of engagement time) for each engagement simulation. TESS™ generates a series of output plots during simulation execution, including: signal power levels at the threat radar, radar boresight, target, and jammer pulse positions (azimuth, range, elevation, Doppler), radar mode, and missile lateral acceleration. In addition, a display of the engagement simulation space is provided, an example of which is shown in Figure 2.9. Since TESS™ is MATLAB®/Simulink® based, integration with the MATLAB® toolboxes is feasible. Detailed descriptions of the TESS™ software package may be found in [20] and [21].

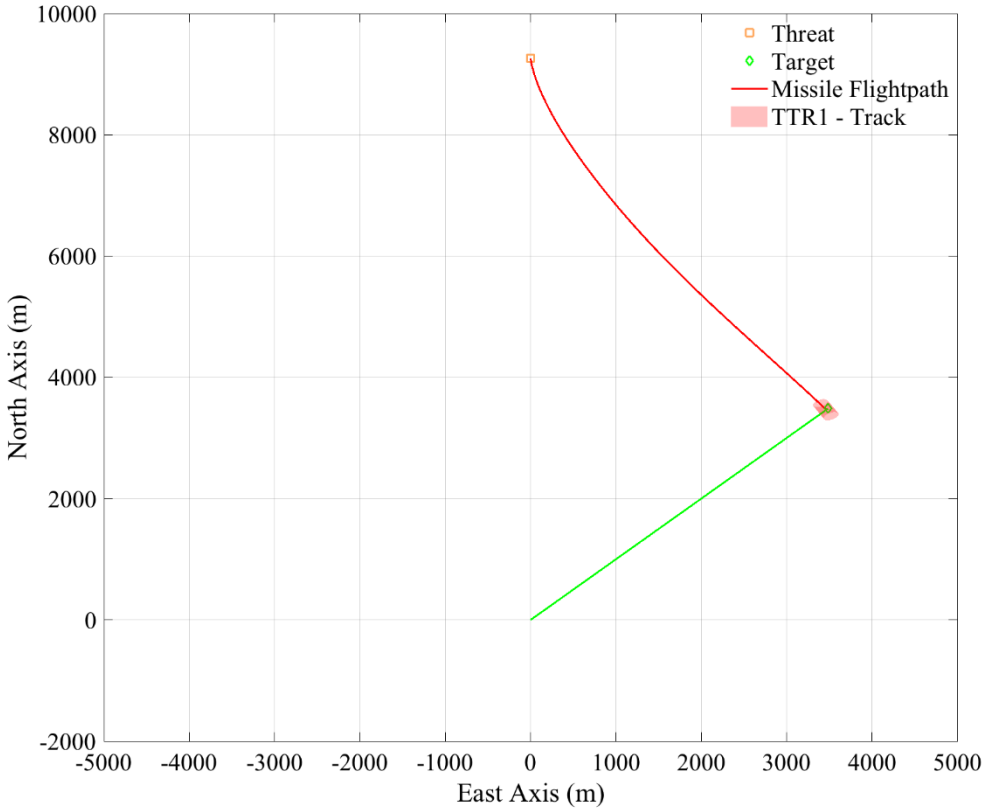


Figure 2.9: TESS™ engagement simulation display

3 Optimization Algorithm Validation and Comparison

This chapter describes the validation process of the two chosen stochastic global optimization algorithms. Before using the optimization algorithms for the problem of ECM technique generation, a validation was conducted to analyze algorithm setup, convergence rates, and suitability in dealing with single-objective optimization problems. An initial comparison of the two algorithms was performed to aid in optimization parameter selection and setting solution space bounds for the intended problem of ECM technique generation.

The GA and PSO algorithms were applied to a series of test functions to validate their overall utility and suitability for their application to the ECM technique generation problem. In addition, the MATLAB® implementations of the optimization algorithms were compared by addressing the capabilities of each function, the flexibility of user customization through options and input parameters, and the complexity associated with parameter selection to tailor the optimization process to a specific problem.

3.1 Test Functions

A number of test functions, referred to as artificial landscapes or benchmark functions, can be used to evaluate the performance of optimization algorithms [22]. The convergence rate, accuracy, and overall performance of a given optimization algorithm can be characterized and compared to other optimization methods. Test functions provide different situations that optimization algorithms may have to deal with when applied to optimization problems, such as a single global minimum among multiple local minima spread throughout a wide search space, a single global minimum within a steep, narrow valley, or multiple global minima interspersed among local minima. The use of test functions with known solutions also permits the tuning of optimization parameters to increase the convergence rate and accuracy of each optimization algorithm, although there is no guarantee that the settings will apply to other solution spaces.

3.1.1 Rosenbrock Function

The Rosenbrock function [23], [24], also known as Rosenbrock's valley or Rosenbrock's banana function, is a non-convex (i.e. neither convex nor concave: it curves up and down), multimodal (i.e. having multiple local minima) function with a single global minimum within a long, narrow, parabolic-shaped valley. Although finding the valley is trivial, convergence to the global minimum is difficult. In 2-dimensional space, the function is defined by

$$f(x, y) = (a - x)^2 + b(y - x^2)^2. \tag{3.1}$$

The global minimum is located at $(x, y) = (a, a^2)$, where $f(x, y) = 0$. Normally the parameters are set as $a = 1$ and $b = 100$ such that the global minimum is at $(1, 1)$. A plot of the Rosenbrock function of two variables is shown in Figure 3.1.

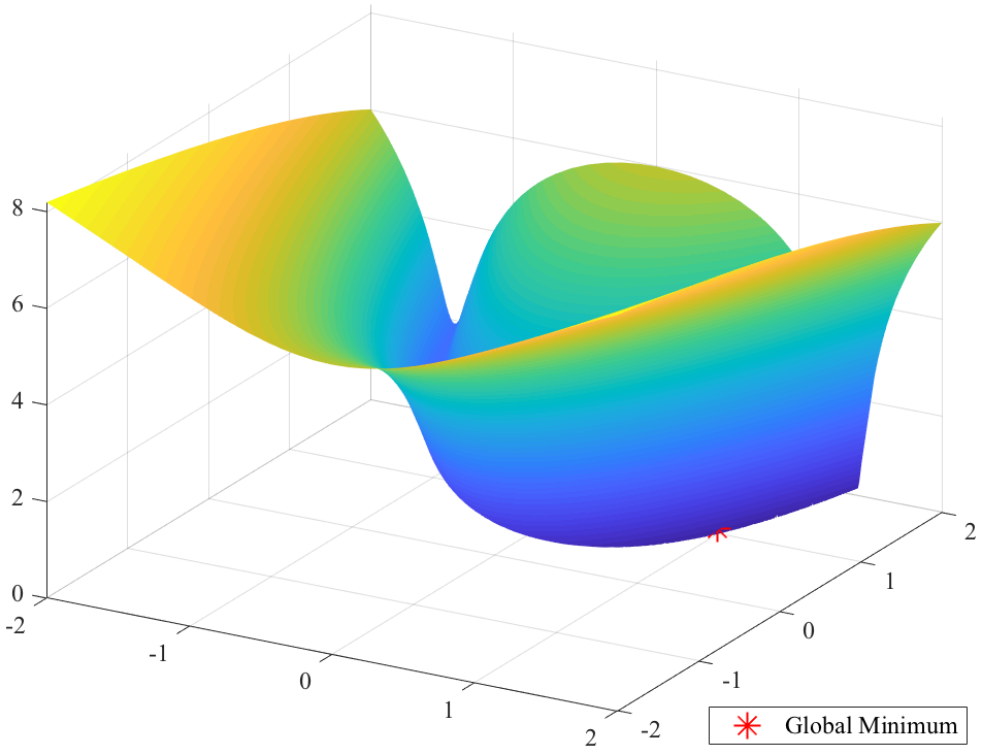


Figure 3.1: Plot of the Rosenbrock function of two variables

The Rosenbrock function may be defined in *n*-dimensional space. The multidimensional generalization of the Rosenbrock function is

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[(a - x_i)^2 + b\left(x_{i+1} - x_i^2\right)^2 \right] \tag{3.2}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$. When the parameters are set to $a = 1$ and $b = 100$, the global minimum $f(\mathbf{x}) = 0$ is at $(x_1, x_2, \ldots, x_n) = (1, 1, \ldots, 1)$; for $4 \leq n \leq 7$ a local minimum also occurs near $(x_1, x_2, \ldots, x_n) = (-1, 1, \ldots, 1)$.

3.1.2 Rastrigin Function

The Rastrigin function [25]–[27] is a non-convex multimodal function defined in *n*-dimensional space. Since the Rastrigin function covers a large search space and has a large number of local minima, finding the global minimum is difficult. For an *n*-dimensional domain, the function is defined by

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right] \tag{3.3}$$

where $A = 10$ and $x_i \in [-5.12, 5.12]$. The global minimum is at $\mathbf{x} = 0$ where $f(\mathbf{x}) = 0$. A plot of the Rastrigin function of two variables is shown in Figure 3.2.

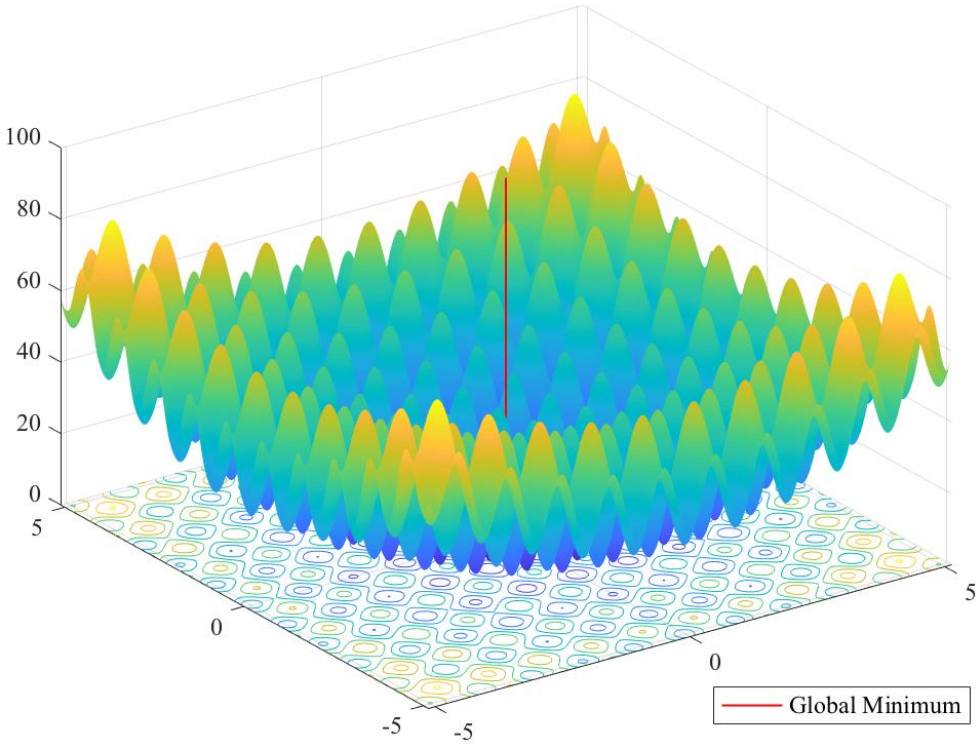


Figure 3.2: Plot of the Rastrigin function of two variables

3.1.3 Hölder Table Function

The Hölder table function [22] is another example of a non-convex multimodal function. This function, defined on 2-dimensional space, has many local minima and four global minima. The function is defined by

$$f(x,y) = -\left|\sin(x)\cos(y)\,exp\left(\left|1-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right|. \tag{3.4}$$

The function can be defined on any input domain but is usually evaluated on $x \in [-10,10]$ and $y \in [-10,10]$. The four global minima are at $\mathbf{x} = (\pm 8.05502, \pm 9.66459)$ where $f(\mathbf{x}) = -19.2085$. A plot of the Hölder table function is shown in Figure 3.3.

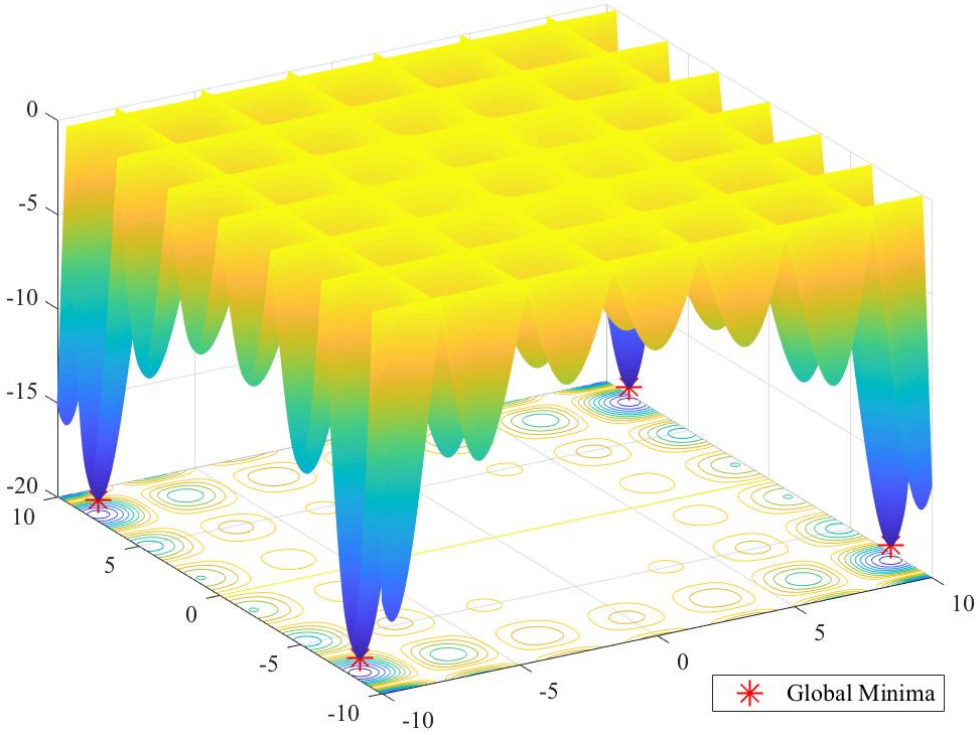


Figure 3.3: Plot of the Hölder table function

3.2 Validation Test Results

Both the GA and PSO were initially executed using the default optimization options [18], [19] for each of the three test functions with two input variables. The Rosenbrock and Rastrigin functions were not bounded, whereas the Hölder table function was bounded to $x \in [-10,10]$ and $y \in [-10,10]$. The default population size for the GA was 50; the PSO had a default swarm size of 20. As each algorithm used randomly generated values when initializing the population/swarm, the random number generation was controlled using the function `rng` and the input `default` (the default settings were the Mersenne Twister [28] with seed 0).

A round was one complete optimization from generation of the initial population/swarm to termination of the algorithm at convergence. Each algorithm was called within a `for` loop and 101 rounds were performed for each. The random number generator was reset prior to each `for` loop such that each algorithm started with the same random seed but results varied between each round. The first round of each algorithm always took longer by an order of magnitude or more than subsequent rounds. The results from the first round of each algorithm were documented separately, the results of rounds 2 through 101 were averaged, and the best result was noted. The results of the optimization for each of the three test functions are shown in Table 3.1, Table 3.2, and Table 3.3, respectively.

Neither algorithm performed particularly well for the Rosenbrock function. The best solution achieved by the PSO was better than the GA, but the average solution of the GA was better. In fact, the average of the solutions found by the PSO was so poor that it was not even located within the valley of the Rosenbrock function. The PSO achieved the exact solution of the Rastrigin function on three rounds and multiple rounds came within 1×10^{-10} or better of the solution; however, the average was not sufficient to be considered solved. Although the GA

converged to within 2.1×10^{-3} of the solution, this was not considered close enough to be a solution to the unbounded Rastrigin function. Both algorithms found one of the four minima of the bounded Hölder table function the majority of the time. In most cases the optimization terminated because the change in the objective value was less than the function tolerance (exit flag 1), where the default value was 1×10^{-6} for both algorithms. Some optimizations terminated when the maximum number of generations/iterations was reached (exit flag 0). Following the first round, in which the PSO was consistently slower than the GA, the PSO consistently terminated the optimization faster than the GA, having performed fewer function evaluations; however, the number of iterations of the PSO was usually greater than the number of generations of the GA. Although a direct comparison of speed is possible, the results are skewed by the population/swarm sizes.

Table 3.1: Rosenbrock function optimization results

Algorithm	**Rosenbrock (Unbounded)**						
	Round	**Execution Time (s)**	**Solution**	**Value**	**Generations / Iterations**	**Function Evaluations**	**Exit Flag**
GA	1	0.3622	[0.345440, 0.144399]	0.491302	119	6000	1
	Best (69)	0.04623	[0.980075, 0.962232]	6.808183e−04	91	4600	1
	Average	0.04567	[1.075226, 2.361244]	2.161397	86.7	4386	1 (95) 0 (5)
PSO	1	1.0308	[−7.398292, 54.749688]	70.553699	124	2500	1
	Best (99)	0.03734	[1.000178, 1.000358]	3.191513e−08	170	3420	1
	Average	0.03029	[−3.038250, 448.406533]	495.185175	132.8	2676	1 (97) 0 (3)

Table 3.2: Rastrigin function optimization results

Algorithm	Rastrigin (Unbounded)						
	Round	**Execution Time (s)**	**Solution**	**Value**	**Generations / Iterations**	**Function Evaluations**	**Exit Flag**
GA	1	0.3379	[0.978493, 0.944311]	2.546291	56	2850	1
	Best (67)	0.1034	[0.000823973, −0.00313809]	0.00208832	200	10050	0
	Average	0.05886	[0.00166144, 0.0799611]	1.214550	104.5	5274	1 (90) 0 (10)
PSO	1	1.0316	[1.919633e−06, −6.562464e−07]	8.165131e−10	127	2560	1
	Best (32, 45, 64)[1]	0.02485	[5.471364e−10, 1.387164e−09]	0	107	2160	1
	Average	0.02193	[0.0199385, 0.0197586]	0.139830	90.6	1831	1 (100)

Note: 1. Where multiple rounds tied for best, the first round to reach the best minimum is included.

Table 3.3: Hölder Table function optimization results

Algorithm	Hölder Table [−10,10]						
	Round	**Execution Time (s)**	**Solution**	**Value**	**Generations / Iterations**	**Function Evaluations**	**Exit Flag**
GA	1	0.4422	[8.0550341, –9.664599]	–19.208503	71	3600	1
	Best	[1]					
	Average	0.08721	[±8.055446, ±9.633054]	–19.112425	67.5	3425	1 (100)
PSO	1	1.0240	[8.0550253, –9.664583]	–19.208503	54	1100	1
	Best	[2]					
	Average	0.01463	[±8.093156, ±9.704444]	–19.010340	49.8	1017	1 (100)

Notes: 1. Only one round did not converge to the solution; the remaining 100 rounds tied for best.
2. Only 12 rounds did not converge to the solution.

Next, the search spaces for the Rosenbrock and Rastrigin functions were bounded to $x_i \in [-2, 2]$ and $x_i \in [-5.12, 5.12]$, respectively, and the optimizations were rerun with default options. The results for the bounded Rosenbrock and Rastrigin functions are shown in Table 3.4 and Table 3.5, respectively.

Bounding the Rosenbrock and Rastrigin functions improved the results for both algorithms, which were able to converge to the known solutions multiple times. The solutions generated by the PSO, when evaluated to at least four significant figures, were more accurate than those generated by the GA; the PSO also took less time (by an order of magnitude in the case of the Rosenbrock) to find the solutions. Clearly, bounding the search space had an influence on the optimization process; however, the data also indicated that a single execution of either optimization algorithm was not sufficient to draw the conclusion that the optimal solution had been achieved. This suggested that multiple rounds of each optimization algorithm with different random seeds for each round would be required to achieve a reliable solution.

The default optimization options for each algorithm resulted in convergence to known solutions when applied to the selected test functions; however, the optimization parameters might need to be modified or 'tuned' in order to better address the unique problem of ECM technique generation.

Table 3.4: Bounded Rosenbrock function optimization results

Test Function	**Rosenbrock [–2,2]**						
	Round	**Execution Time (s)**	**Solution**	**Value**	**Generations / Iterations**	**Function Evaluations**	**Exit Flag**
GA	1	0.6059	[1.272954, 1.620862]	0.0745241	200	10050	0
	Best (97)	0.07984	[0.999496, 0.998977]	2.772922e–07	68	3450	1
	Average	0.2064	[1.011465, 1.030856]	0.00796079	180.1	9055	1 (20) 0 (80)
PSO	1	1.0399	[1.022435, 1.045581]	5.076459e–04	58	1180	1
	Best (75)	0.02798	[0.999982, 0.999964]	3.315381e–10	128	2580	1
	Average	0.02303	[0.998285, 0.996755]	3.052633e–04	96	1940	1 (100)

Table 3.5: Bounded Rastrigin function optimization results

Test Function	Rastrigin [–5.12,5.12]						
	Round	Execution Time (s)	Solution	Value	Generations / Iterations	Function Evaluations	Exit Flag
GA	1	0.4520	[0.994964, 9.627379e–06]	0.994959	83	4200	1
	Best (3)	0.1034	[6.164289e–07, –5.736482e–07]	1.406697e–10	84	4250	1
	Average	0.09810	[0.0492594, 0.0394104]	0.325086	79.2	4008	1 (100)
PSO	1	1.0485	[0.978493, 0.944311]	2.546291	48	980	1
	Best (75, 77)	0.01712	[–1.476723e–09, –2.689717e–09]	0	71	1440	1
	Average	0.01882	[–0.0295536, –3.020851e–07]	0.108362	72.2	1463	1 (100)

3.3 MATLAB® Implementation Comparison

Review of the available documentation immediately identified key differences between the MATLAB® implementations of the GA and PSO. In terms of capability, the most significant difference is that only the GA can accept constraints on the optimization problem. To accomplish a direct comparison of the two algorithms the candidate solutions they generate should be restricted by the same bounds and constraints. Since constraints are not possible for the PSO, control of the candidate solutions, when applied to real-world scenarios, will be limited to the application of lower and upper bounds on each variable of the candidate solution.

A number of MATLAB® GA and PSO optimization options influence the execution time of each function. The default values for the population/swarm size, maximum number of generations/iterations, and maximum number of stall generations/iterations are different for each, making a direct comparison difficult without changes to the default settings.

The GA function has 28 unique options that can be user-modified. The three options that most significantly control GA operation are *selection*, *crossover*, and *mutation*. The options provide user-selectable selection, crossover, and mutation functions, along with modifiable parameters for each function; however, there are a number of available built-in functions for each option, plus the ability to write custom functions. There are also two reproduction options: *elite count* and *crossover fraction*. Elite count specifies the number of individuals that are guaranteed to survive to the next generation, and crossover fraction specifies the fraction of the next generation, other than elite children, that are produced by crossover [18]. Depending on the functions chosen for selection, crossover, and mutation, as many as nine options and parameter settings are available to modify. Modifications to the default GA options would be time-consuming and require a number of simulations to determine the best combination of settings.

The PSO function has 20 unique options that can be user-modified. There are four options that directly control PSO operation: *inertia range*, *minimum adaptive neighborhood size*, *self-adjustment weight*, and *social-adjustment weight*. Since these options are either a two-element vector or a scalar value, the total number of parameter settings to control the algorithm is much lower for the PSO when compared to the GA.

The optimization options for each algorithm, along with their default values and those values used in this thesis, are described in detail in Appendix A.

3.4 Summary

The GA and PSO optimization algorithms were validated using three test functions: the Rosenbrock, the Rastrigin, and Hölder table. Both algorithms were shown to converge to the known solutions of each test function; however, bounding the search space was required to achieve an acceptable result. In addition, a single optimization run was not sufficient to converge to the known solution. Instead, multiple optimization runs, each beginning with a different random seed, will provide an indication of the tendency for the algorithm to approach the known solution.

For a bound solution space, the PSO consistently achieved solutions closer to the known solution, when compared to the GA. The PSO also took less time to converge; however, this was due to the default swarm size being significantly smaller than the GA population size (20 vs. 50, for the PSO and GA, respectively). This indicates that the PSO is capable of converging to a better solution while testing fewer candidate solutions at each iteration. The PSO would appear to be more efficient when solving the test functions used; however, the problem of ECM technique generation may prove different.

Selection of options that control algorithm operation needs to follow an iterative trial and error process to determine their effect on the optimization outcome. The wide degree of customization afforded by the MATLAB® implementations of the GA and PSO, although flexible, imposes a significant time cost in 'tuning' the algorithms to a specific problem. The problem of ECM technique generation may not require significant deviation from the option default values to achieve the generation of effective ECM techniques.

4 ECM Technique Generation Methodology

This chapter presents an overview of the design of the software architecture that bridges the existing TESS™ proprietary software with the task-specific MATLAB® toolboxes. The combined system is designed to generate ECM techniques using global optimization, which can be faster than direct-search methods. The following sections detail the design process, challenges, and considerations, including development of the fitness function, parallelization of the process for reduced computation time, and the definition of deception jamming techniques compatible with the TESS™ product.

4.1 Integration of TESS™ with the Global Optimization Toolbox™

The TESS™ product is based in MATLAB®/Simulink® and is normally accessed via a graphical user interface (GUI). TESS™ permits the modification of a number of threat, target platform, and environmental parameters for either single engagements or batch runs (in which multiple engagements with different parameter values are run either sequentially or in parallel). Engagement parameters are normally modified in dialog boxes of the GUI by the user. Such manual parameter updates are sufficient for simulating engagements with pre-planned conditions and techniques; however, global optimization randomly generates initial candidate solutions, updating the candidate solution parameters through multiple

iterations in which subsequent populations are based on the results of previous populations.

Global optimization algorithms have not previously been used to provide input parameters to TESS™, which presents a number of challenges. The TESS™ GUI does not provide direct access to parameters, which are stored either as variables in the MATLAB® base workspace or as Simulink® mask parameters. Fortunately, the core component of TESS™, the Simulink® model, can be run directly from the MATLAB® environment without the requirement to access TESS™ via its GUI. This feature is essential since the optimization algorithms in the MATLAB® Global Optimization Toolbox™, which are functions called from the MATLAB® command line, require that a candidate solution be passed as an input argument to a MATLAB® function and the computed fitness be passed back to the optimization algorithm as an output argument.

Candidate solution parameters generated by the optimization algorithm must be mapped to the TESS™ Simulink® model variables stored in the MATLAB® base workspace and engagement simulation output variables must be accessed to permit fitness function computation. The hierarchical structure of the MATLAB® workspaces (i.e. a separate workspace is created for each function, with a base workspace for the command line), combined with the MATLAB® implementations of the GA and PSO (in which all subsequent processes are executed from a function instantiated by the GA or PSO function), require that the process flow include the following steps:

1. Initialize the program with user inputs (defined below);
2. Call the optimization function (GA or PSO);
3. Transfer the input parameters to the TESS™ Simulink® model base workspace variables;
4. Simulate the engagement using the TESS™ Simulink® model;
5. Compute a fitness score based on the TESS™ Simulink® model output variables and a user-defined fitness function;

6. Iteratively continue the process until convergence to a solution is achieved; and
7. Output the parameters of the generated ECM technique.

The resulting high-level software architecture is shown in Figure 4.1. Each block is described in the following subsections.

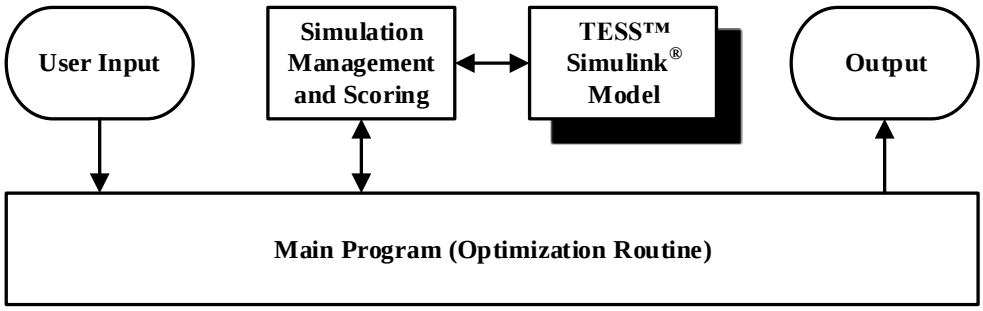


Figure 4.1: ECM technique generation software architecture

4.1.1 User Input

The user input includes a number of user-selectable options for the program: the optimization algorithm to be used, optimization algorithm options (e.g. population/swarm size, maximum number of iterations, search space upper and lower bounds, number of optimization variables), airborne target type and initial conditions (e.g. altitude, velocity, approach angle relative to the threat), and target manoeuvre profiles. User inputs were hard-coded variables in the main MATLAB® script that could be modified prior to program execution.

4.1.2 Main Program

The main program serves as the user interface of the ECM technique generation program. The program includes settings for the simulation conditions and optimization options (user input), initialization routines for loading variables and

opening the TESS™ Simulink® model, the function call for the optimization algorithm, and functions for viewing and saving the results. The optimization function call (GA or PSO) enters an iterative loop in which all simulation and scoring functions are executed until the optimization ends, due to one of several exit criteria (as detailed in Appendix A).

4.1.3 TESS™ Simulink® Model

The threat system, RF channel, transmit and receive environments, and target platform, are simulated entirely within TESS™. The core TESS™ component includes a Simulink® model designed to simulate engagement scenarios for a specific class of weapon system. The Command Guided Surface-to-Air Missiles and Anti-Aircraft Artillery (SAMCGAAA) model was chosen because it provides a threat system capability conducive to the evaluation of range and frequency deception jamming techniques. The top-level block diagram of the TESS™ Simulink® model is shown in Figure 4.2 [29].

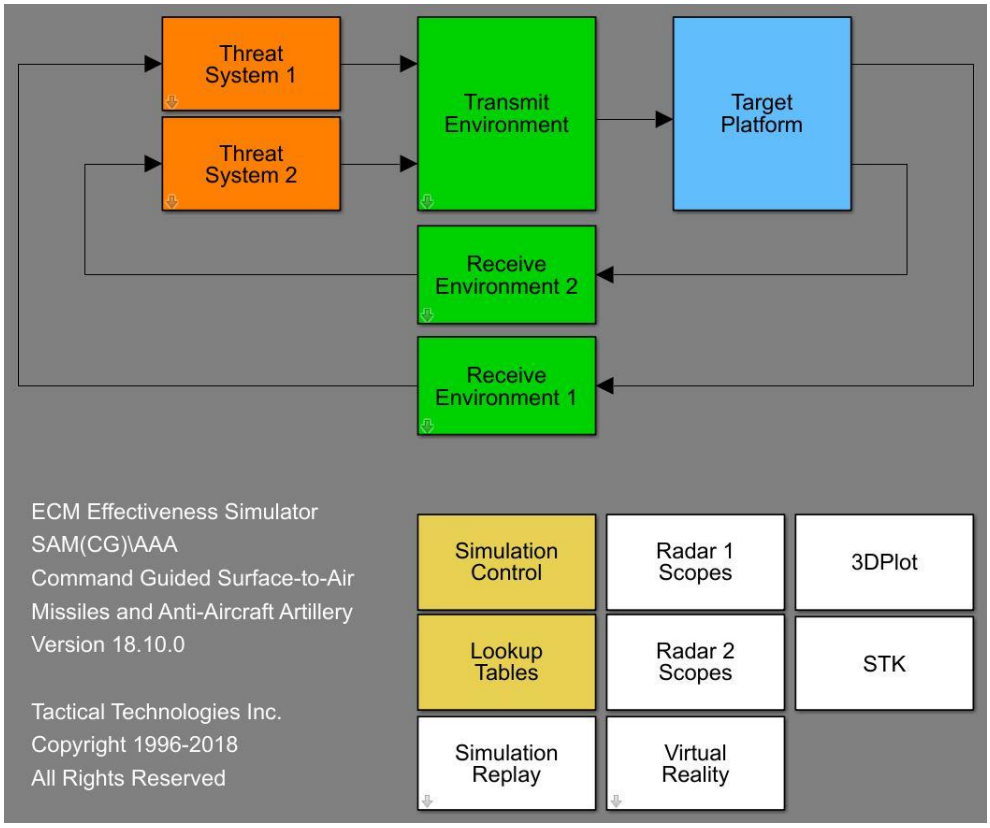


Figure 4.2: TESS™ SAMCGAAA Simulink® model [29]

Each block has a series of system parameters that are modifiable either through dialog boxes within Simulink® or through the MATLAB® command line. Within the *Threat System* blocks are the transmit and receive systems as well as the missile system. The *Target Platform* block includes the target aircraft and its self-protection systems, including the jammer and other ECM systems such as chaff, towed decoys, and expendable active decoys.

The TESS™ SAMCGAAA Simulink® model was delivered with generic pre-programmed systems, including: one threat system, an airborne target represented by a fighter or rotary-wing aircraft, and one self-protection jammer system. Although unclassified publicly-available specifications were used in the programming of these systems, it is understood that TTI made a number of

educated assumptions and extrapolations of specific systems to populate the system block parameters. As a result, the TESS™ product is considered Controlled Goods under the Defence Production Act [30] and the system parameters of the particular threat and jammer systems that were simulated cannot be disclosed herein. The threat, airborne target, and self-protection jammer system parameters were verified, where possible, against unclassified data widely available on the internet.

Although generic pre-programmed systems were used in this thesis, the generic system blocks are customizable and can be altered to better represent specific military systems, should this intelligence be made available to the user.

4.1.3.1 Ground-Based Threat System

The threat system is based on unclassified system specifications for the 9K33 Osa (NATO reporting name SA-8 *Gecko*) [31]. The SA-8 is an anti-aircraft SAM system that provides missile tracking via an RF command guidance system (i.e. missile guidance is provided solely through the ground-based radar system). Although early versions of the SA-8 had limited range and altitude performance, the SA-8 was shown to be effective against low-flying fixed and rotary-wing aircraft [32]. Upgrades have increased system performance to a maximum engagement altitude of 12 km and maximum range of 15 km [33]. Use of a pulsed TTR make this system appropriate for evaluating the effectiveness of range and frequency deception techniques in protecting fixed and rotary-wing aircraft. The generic threat used both non-coherent and coherent receiver modes for evaluating range-only (i.e. RGPO/RGPI) and range and frequency coordinated (i.e. RGPO/RGPI and VGPO) deception jamming techniques. A photograph of the SA-8B transporter erector launcher and radar (TELAR) vehicle is shown in Figure 4.3. The rotating parabolic dish antenna mounted on top of the vehicle is for surveillance and acquisition whereas the flat-panel arrays (tan coloured) are those used for target tracking, fire control, and command guidance of the missile.

Figure 4.3: 9K33AKM Osa AKM / SA-8B Gecko TELAR vehicle [31]

4.1.3.2 Airborne Target Platform

The airborne target platform can be selected as either a fixed or rotary-wing aircraft, modelled as a cylinder. The aircraft dimensions are used by TESS™ in the calculation of miss distance [20]. The model also includes a parameter called vulnerable area, given in square meters, which is used in the probability of kill and probability of survival calculations [20]. The default values for radius, length, and vulnerable area are: 5 m, 14 m, and 50 m^2. The default dimensions were retained since they are representative of either a fighter aircraft (e.g. the F-16 is approximately 15 m long and 4.9 m high [34]) or an escort/attack helicopter (e.g. the Bell CH-146 is approximately 17.1 m long and 4.6 m high [35]), which are examples of airborne targets likely to be engaged by the SA-8 threat system.

4.1.3.3 Self-Protection Jammer

The self-protection jammer is based on unclassified system specifications for the AN/ALQ-126B Airborne Defensive Electronic Countermeasures (DECM) set [36]. This legacy system, still carried on the CF-18 fighter aircraft, is primarily designed to jam TTRs [37] and is capable of a number of deception jamming techniques, including both RGPO/RGPI and VGPO. The jammer provides coverage up to the I/J RF bands and is capable of emitting in excess of 1 kW power per band. Fore and aft high-band antennas provide 60-degree beam width with a 15-degree lookdown angle for ground threat coverage.

4.1.4 Simulation Management and Scoring

The optimization algorithms included in the Global Optimization Toolbox™ require that all simulation and scoring functions take place within a function (referred to in MATLAB® documentation as a fitness or objective function) called by the optimization algorithm function. Once the optimization function is called from the MATLAB® command line, all subsequent function calls take place within this objective function or a custom output function, typically used to save or plot intermediate optimization states and results at the completion of each iteration of the optimization. Simulation management and scoring includes the following tasks:

1. Transfer the ECM technique candidate solution variable set generated by the optimization algorithm to the base workspace jammer parameters of the TESS™ Simulink® model;
2. Set the target platform initial flight path approach angle in the base workspace target platform parameters of the TESS™ Simulink® model;
3. Simulate the engagement via the TESS™ Simulink® model; and
4. Evaluate the engagement results and generate a fitness score that is passed back to the optimization algorithm.

4.1.4.1 Fitness Function

No publicly available guidance, either theoretical or experimental, exists for an optimal ECM fitness function. Therefore, the fitness function was designed based on the available performance output parameters from the Simulink® model and the desired engagement outcomes resulting from an effective ECM technique. Effective ECM outcomes include:

1. Prevention of a missile launch;
2. Maximizing the missile miss distance; and
3. Minimizing the engagement time in which the threat radar is tracking the real target.

The fitness function contains a series of conditional statements that evaluate the airborne target's ability to survive the missile engagement. The fitness score is computed using a series of equations that are solved sequentially based upon the values of select output parameters. Five distinct conditions are evaluated to arrive at the final fitness score of the ECM technique candidate solution.

TESS™ computes the probability of kill, P_{kill}, and probability of survival, P_{surv}, such that:

$$P_{kill} + P_{surv} = 100\,. \tag{4.1}$$

Although a number of engagement simulation parameters (such as target vulnerable area) are used in computing P_{kill}, it is largely based on the missile miss distance, D_{miss}, between the target aircraft and the missile.

When P_{kill} is greater than or equal to 90 percent, the ECM candidate solution is deemed ineffective and the fitness score is set to 1. If multiple target platform flight path approach angles are undergoing evaluation for the current candidate solution, the optimization is terminated, the candidate solution is dismissed, and the remaining approach angles are not evaluated.

If a missile launch does not occur, the candidate solution is deemed 100 percent effective and the fitness score is set to 0. If multiple target platform flight path approach angles are undergoing evaluation for the current candidate solution, the remaining approach angles are evaluated.

If D_{miss} is greater than 100 m but less than 500 m, the fitness is computed as a linear function of D_{miss} from 0.3 to 0.1:

$$fitness = 0.35 - 0.0005 \times D_{miss} . \tag{4.2}$$

If D_{miss} is less than or equal to 100 m, the fitness is computed as a weighted sum of the probability of survival, P_{surv}, D_{miss}, and percent of engagement time that the threat radar is not in track mode, where T_{track} is the percent of time that the radar is in track mode:

$$fitness = 1 - \frac{1}{100}\left(W_{surv}P_{surv} + W_{miss}D_{miss} + W_{track}(100 - T_{track})\right) \tag{4.3}$$

The weights were chosen as follows:

Probability of survival weight, $W_{surv} = 0.1$;

Miss distance weight, $W_{miss} = 0.5$; and

Percent of time not in track mode weight, $W_{track} = 0.4$.

These weight values were chosen for two reasons: since P_{kill} and P_{surv} are based on D_{miss}, they provide limited value in determining the fitness for small miss distances; and, using the above weights, for a P_{surv} of 100 percent, a D_{miss} of 100 m, and the threat radar in track mode 75 percent of the engagement time, the computed fitness using (4.3) is 0.3. This value aligns with the linear function from (4.2) and provides continuity, rather than a step change, between (4.2) and (4.3).

Where multiple target platform flight path approach angles are evaluated for a single ECM candidate solution, the overall fitness score is computed as the maximum fitness (worst case) value for all approach angles evaluated.

Simulation management and scoring, which includes the fitness function, is displayed as a flowchart in Figure 4.4.

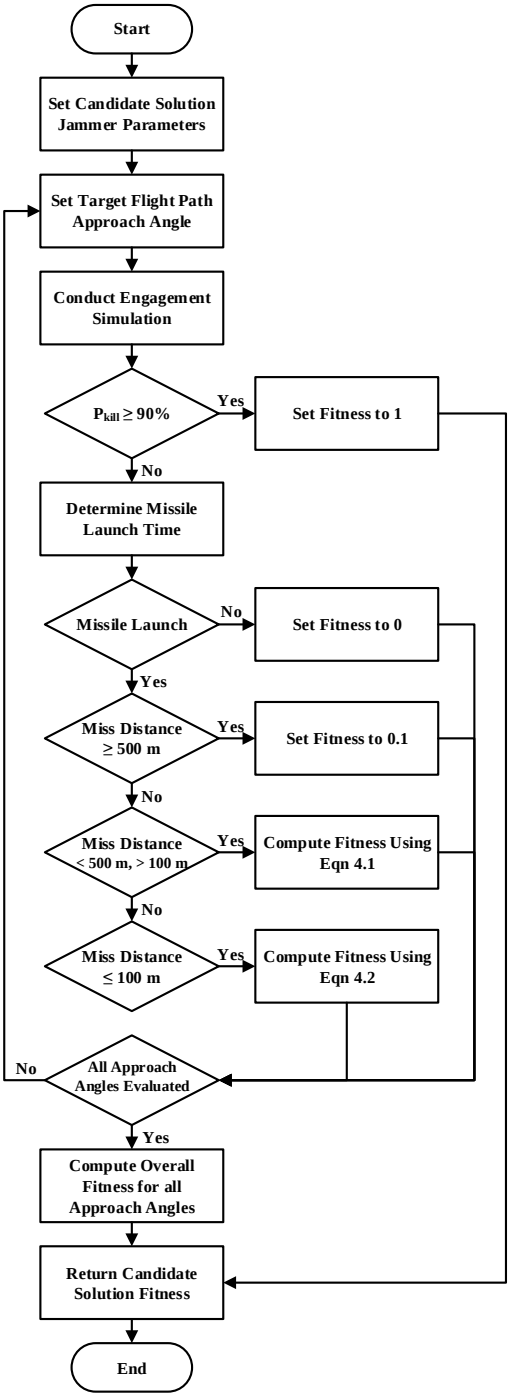


Figure 4.4: Simulation management and scoring flowchart

4.1.5 Program Output

Each optimization algorithm provides as output a data set after each iteration/generation. Included in the data set are the values of each candidate solution in the swarm/population along with the fitness score of each member. From this information the best and average fitness scores can be computed for each iteration/generation. A history of the entire optimization process is recorded by the main program for later analysis. At the completion of the optimization process the candidate solution with the best fitness score along with various metrics (such as execution times and exit flags) are also recorded.

4.2 Parallelization

Parallelization refers to the execution of similar or iterative processes simultaneously rather than in a sequential manner [38]. The MATLAB® Parallel Computing Toolbox™ permits the use of multicore central processing units (CPUs), graphics processing units (GPUs), and computer clusters to solve computationally and data-intensive problems [39].

The Global Optimization Toolbox™ allowed for parallelization of the optimization via the Parallel Computing Toolbox™. However, the addition of the TESS™ Simulink® model and its associated parameters and hidden internal variables, when combined with the customized fitness scoring process, made parallelization a non-trivial endeavour. Modifications to the initialization of the optimization routine were required to enable a parallelized fitness function.

Taking advantage of the processing power of multicore computing, MATLAB® computational engines, or *workers* can be run locally with one worker per core available. The function that carries out simulation management and scoring is instantiated once per worker with each one using its own workspace and executing its own instance of the TESS™ Simulink® model. Since a directory of

files in support of the TESS™ Simulink® model is dynamically updated during each simulation, it is necessary to copy the entire directory, along with the model file and MATLAB® script and functions, to a temporary directory for each worker in the parallel pool. The parallelization, where n is the total number of CPU cores available, is depicted in Figure 4.5.

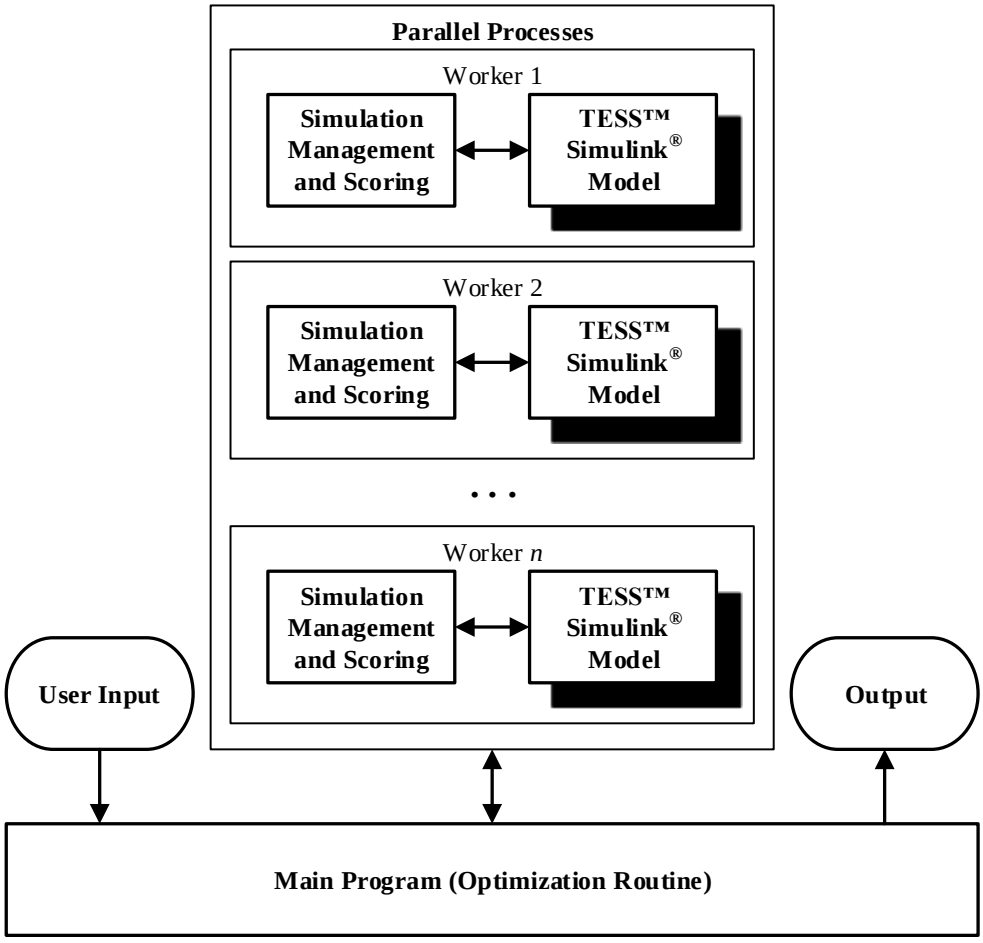


Figure 4.5: Parallelization of the simulation and scoring functions

4.2.1 Serial versus Parallel Benchmark Comparison

The serial and parallel implementations of the simulation and scoring functions were compared using an engagement simulation to determine the speed increase obtained via parallelization. The simulated engagement was conducted between the TESS™ generic fighter aircraft and threat system (with a non-coherent receiver). Optimizations of a six-variable range-only technique (i.e. initial position, final position, initial dwell time, final dwell time, velocity, and acceleration) were conducted using both the GA and PSO, each using their default options and a population/swarm size of 20. All optimization variables and bounds were held constant for the serial and parallel executions and the same initial random seed was used.

Simulations were run on a computer with the Windows 7 Enterprise (with Service Pack 1) 64-bit operating system. Two Intel® Xeon® CPU E5-2650 v4 processors, rated at 2.20 GHz, were installed. The computer had 32 GB of random access memory (RAM) installed. Due to computer memory constraints at the time of benchmark testing, a maximum of 16 parallel workers were used.

The results of the benchmark comparison are shown in Table 4.1. Execution overhead is the time required by the program to conduct the setup and initialization of the simulation, including opening the TESS™ Simulink® model, loading initialization variables, and closing the model upon completion of the optimization. The parallel implementation includes the additional tasks of starting the parallel pool, creating temporary directories, copying the model files, and loading the Simulink® model for each worker, as well as closing the model, removing model files and temporary directories, and shutting down the parallel pool. Although this additional execution overhead was greater for the parallel implementation, it did not have a significant impact on the overall execution time given the time required to run a single optimization round.

Table 4.1: Serial vs. parallel benchmark comparison results

	Overhead	**GA Execution Time**	**PSO Execution Time**	**Total Execution Time**
Serial	0.75	847.48	465.59	1313.82
Parallel	3.51	145.04	81.21	229.76
Overall Speedup				5.72x

Note: All times are in minutes.

The results (i.e. the ECM techniques and associated fitness scores) for the serial and parallel implementations were identical. For both serial and parallel implementations, the GA required 68 generations and 1380 function evaluations to converge to a solution, whereas the PSO required 37 iterations and 760 function evaluations to converge. Although the parallel implementation demonstrated increased overhead associated with initialization, for the specific variable set, simulation parameters, and population/swarm size, it was 5.72 times faster than the serial implementation. The parallelization of the simulation and scoring functions allowed ECM technique generation to be performed in hours instead of days. The increase in speed can be used to intensify the optimizations, either by increasing the population/swarm size and the number of individual rounds for each optimization algorithm (each with a unique random seed), or by using multiple engagement geometries.

Each worker uses approximately 2 to 2.5 GB of RAM when running the parallelized optimization routine. As a result, to maximize the use of all available CPU cores, the available RAM should be no less than 2.5 times the number of cores. Subsequent simulations were performed on the same computer as described above, but with a total of 64 GB of RAM installed. This permitted the use of all 24 CPU cores when running the parallelized optimization routine.

4.3 Deception Jamming Technique Design

The deception jamming techniques introduced in Chapter 2, namely the RGPO/RGPI and VGPO, are used for the engagement simulations. TESS™ provides options for programming single techniques or combinations of multiple techniques for the generation of false targets. The design of range and frequency deception jamming techniques was limited to the parameters available within the TESS™ jammer system.

4.3.1 Range Deception

In TESS™, the jammer range deception program is defined via ten parameters, defined in Table 4.2. For a given single pulse range technique, the pulse is assumed to be On for the duration of the engagement. Thus, the parameters *Pulse On* and *Pulse Off* are excluded from the variable set. Therefore, any single pulse range technique is defined by the eight remaining variables.

Table 4.2: TESS™ jammer range program parameters

Parameter	Units	Description
Pulse On	s	Elapsed time from jammer turn-on time until the pulse sequence commences
Pulse Off	s	Elapsed time from Pulse On until the pulse turns off
Initial Position, R_0	μs (1 μs = 150 m)	Initial position of the pulse false range target, relative to the aircraft position
Final Position, R_{max}	μs (1 μs = 150 m)	Final position of the pulse false range target, relative to the aircraft position
Initial Dwell Time, T_i	s	Time that the pulse false range target dwells over the Pulse Initial Position
Final Dwell Time, T_f	s	Time that the pulse false range target dwells over the Pulse Final Position
Velocity, v	m/s	Velocity with which the pulse false range target moves from the Pulse Initial Position to the Pulse Final Position
Acceleration, a	m/s^2	Acceleration of the pulse false range target
Cover Pulse Reduction	dB	Attenuation relative to the jammer's full power
Pulse Width, PW	μs	Pulse width used by the pulse false range target

The TESS™ implementation models the most common range deception profile: constant acceleration. The resulting walk-off profile for a single pulse of the range deception technique is shown in Figure 4.6.

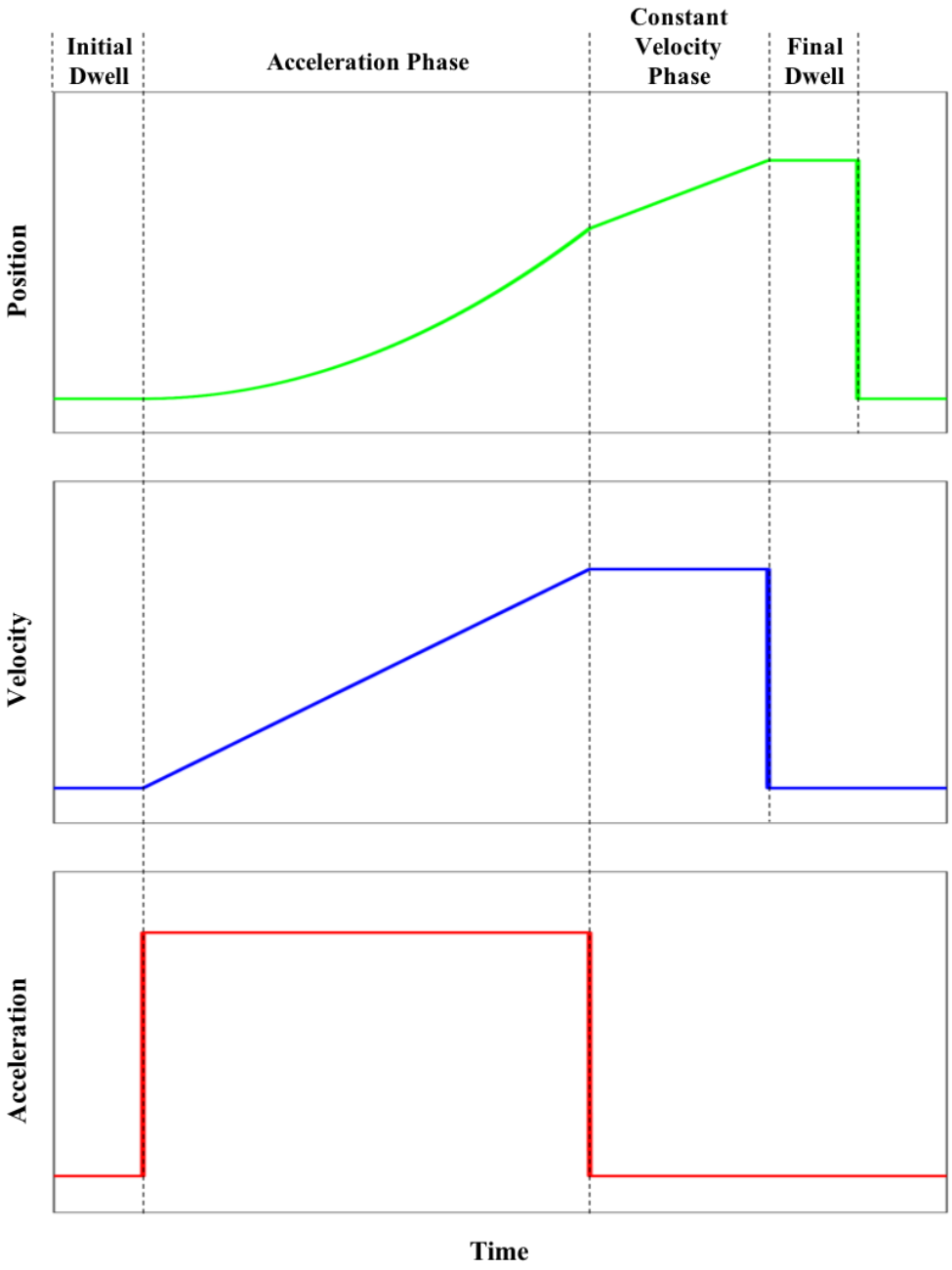


Figure 4.6: Range deception walk-off profile (single pulse)

The jammer pulse starts at or near the real target, at R_0, dwells (stays with the target aircraft) for some period of time, T_i, and then accelerates at a constant rate, a, away from the target until the maximum velocity, v, is reached (or the maximum range, R_{max}, is reached; whichever occurs first). The pulse then continues to move away from the target at constant velocity until reaching the maximum range, R_{max} (if R_{max} was not reached prior to achieving the maximum velocity). The pulse dwells in position for some period of time, T_f, before transmission ceases and the cycle begins again. The constant acceleration and combined linear/constant velocity profile result in a combined parabolic/linear position profile.

4.3.2 Frequency Deception

In TESS™, frequency deception programs are defined by waveform type (programmable, sinusoid, and noise) and then by a corresponding set of parameters. To achieve a frequency deception technique (i.e. VGPO), coordinated with a given range technique, the programmable type is used. The programmable frequency deception program accepts two vectors, each containing up to 50 values. A vector of time values, in seconds, corresponds to a vector of frequency values, in kHz. Together, the time and frequency vectors form a single period of a piece-wise linear periodic sequence. The frequency profile corresponding to a false target turning away from the threat radar (i.e. negative Doppler frequency shift, RGPO) is shown in Figure 4.7.

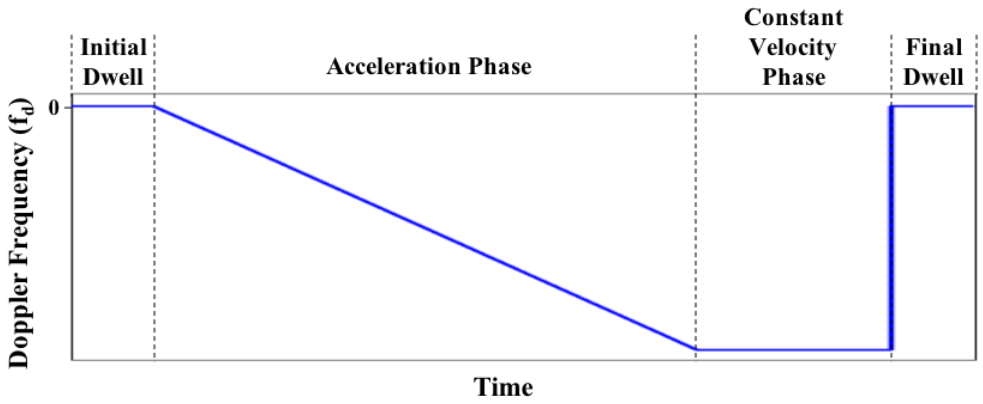


Figure 4.7: Frequency deception walk-off profile (single pulse)

The frequency technique can be coordinated with a given single-pulse range technique using the initial and final positions, R_0 and R_{max}, initial and final dwell times, T_i and T_f, velocity, v, and acceleration, a, of the range technique pulse. The basic equations of motion [40] can then be used to compute the instantaneous velocity corresponding to the programmed range technique. Recalling that the frequency program can only use up to 50 discrete frequency values, the Doppler frequency, f_d, required for a defined time step is computed using (2.7). Since the frequency technique uses the range technique parameters to generate the time and frequency values, only one additional parameter is required for a coordinated range/frequency technique: a binary number where a 0 indicates frequency coordination is not used and a 1 indicates frequency coordination is active. Therefore, a frequency coordinated range technique is defined by nine variables.

When both range and frequency deception programs are executed concurrently, the false target pulse will inject coordinated false range and velocity information into the receiver of a coherent pulse-Doppler radar system. The technique, if effective, will either force the victim radar to lose target track (*break-lock*) so frequently that a missile launch will be prevented, or the false range and velocity information will lead a launched missile off course to miss the target aircraft.

4.4 Summary

This chapter presented the design of the software architecture that bridges the proprietary TESS™ Simulink® model with the MATLAB® Global Optimization Toolbox™ and Parallel Computing Toolbox™. The design process, challenges, and considerations, including development of the fitness function and parallelization of the process for reduced computation time, were covered. Details of the TESS™ Simulink® model, such as the default parameter settings for the threat system, airborne target, and self-protection jammer, were introduced. The definition of deception jamming techniques that meet the programming requirements of TESS™ was also explored.

The final design of the software architecture was dictated by the TESS™ Simulink® model, the MATLAB® optimization functions, and the resulting storage and transfer of data between each software block. Despite these constraints, integration of the proprietary TESS™ Simulink® model with the MATLAB® Global Optimization Toolbox™ was achieved for the first time. Parallelization of the simulation and scoring processes was also accomplished, and was demonstrated to significantly decrease the execution time for the optimization process when compared to the serial implementation in benchmark testing.

Successful integration of the required software elements was a significant milestone in the development of a system capable of generating ECM techniques using global optimization, which can be faster than direct-search methods. However, each optimization algorithm has a series of parameters and options that must be carefully selected to achieve convergence in minimal execution time. Finally, a series of engagement simulations with specific conditions such as threat-target geometry, target type, altitude, and airspeed, and both non-manoeuvring and manoeuvring profiles, must be defined.

5 Simulation Setup and Results

This chapter begins with a discussion of the optimization setup, which includes optimization options and solution space bounds, focussing on the reasoning behind selecting specific values and their effect on the optimization process. Engagement scenario design, including the simulation conditions and their role in scoping the overall execution, is also examined.

The results from each engagement scenario are presented and analyzed, with a focus on the effects of each deception technique parameter, or optimization variable, on the performance of the generated ECM techniques against the threat system and their resulting fitness. The performance of each optimization algorithm is compared, concentrating on the convergence speed and the generation of suitable ECM techniques by comparing the fitness scores.

5.1 Optimization Setup

The optimization setup involves defining all parameters required to carry out the optimization with each algorithm function. Each algorithm is run with the same optimization options, where possible and appropriate (e.g. options that affect total execution time or termination criteria). Other options unique to each algorithm must be set accordingly if the default value is not to be used. Setup also involves selecting upper and lower bounds on each optimization variable in the problem. Finally, each optimization algorithm must be initialized with a random seed, which is a number used to generate a pseudo-random sequence of numbers that are in turn

used by the algorithm at each generation/iteration. Each algorithm is run ten times per simulation, where each run is referred to as a *round*. Each round is seeded with the number sequence of the round (e.g. round one is seeded with '1', round two is seeded with '2', etc.); this is done for reproducibility. At the end of each simulation, ten rounds using GA and ten rounds using PSO are completed, resulting in a total of 20 ECM techniques and their associated fitness score.

5.1.1 Optimization Options

Only a limited number of available optimization options were modified for each algorithm. All simulations were run using the parallelized simulation and scoring functions, using 24 CPU cores. This permitted 24 candidate solutions to be evaluated simultaneously. The population/swarm size was limited to 48 for all simulations. Choosing a multiple of 24 meant that all CPU cores were used to evaluate the population/swarm at each generation/iteration, maximizing the efficient use of the available computing capability. Keeping this number at 48 also provided the best trade-off in terms of overall execution time and search of the bounded solution space.

The fitness limit (referred to as objective limit by the PSO algorithm), was set to 0, since this was the minimum value that the fitness function was designed to find. Once the scoring system found a candidate solution with a fitness of 0 the simulation would end. The function tolerance was used to determine whether the average relative change in the best fitness score was changing between generations/iterations. Initially, the function tolerance was kept at its default value of 1×10^{-6}; however, this was later changed to 1×10^{-3} since the fitness function only provided fitness scores measured to more than one significant figure if the miss distance was less than 500 m. The maximum number of generations/iterations was limited to 100; however, no simulation was observed to reach this maximum. The maximum stall generations/iterations were options that

were found to significantly influence the execution time of each algorithm. The default value for the GA is 50; however, the default value for the PSO is only 20. If a simulation did not converge to the minimum fitness score of 0 it would usually stall at a fitness score of 0.1 and continue until reaching the maximum stall generations/iterations. Initially, the GA option was kept at the default of 50, which resulted in the GA always taking much longer to complete a simulation when compared to the PSO. In these cases, observing the scores at each generation indicated that function evaluations beyond 20 generations, when the fitness score had stalled, had no effect on improving the fitness score. Thus, for subsequent simulations, the value was set to 20 generations for the GA.

Optimization options that influence crossover and mutation rates for the GA were modified from the default values once during testing. The results of those changes are discussed in the results section of this chapter.

5.1.2 Optimization Bounds

Since constraints could not be defined for the PSO algorithm implemented in MATLAB®, only optimization bounds were used to provide an equal basis of comparison between the two algorithms. The upper and lower bounds for each optimization variable are chosen to limit the solution space. Limiting the solution space reduces the number of candidate solutions, decreasing the execution time required to converge to an optimized solution. However, the chosen bounds may also exclude potential global solutions. The bounds imposed on each optimization variable must be logical when considering the problem and should consider the capabilities and limitations of the jammer system being simulated. The optimization variable bounds chosen for ECM technique generation are shown in Table 5.1.

Table 5.1: Optimization variable bounds

Optimization Variable	Units	Lower Bound	Upper Bound
Initial Position, R_0	μs	–5	5
Final Position, R_{max}	μs	–15	15
Initial Dwell Time, T_i	s	0	5
Final Dwell Time, T_f	s	0	5
Velocity, v	m/s	200 (Fighter) 5 (Rotary)	600 (Fighter) 90 (Rotary)
Acceleration, a	m/s^2	10 (Fighter) 10 (Rotary)	60 (Fighter) 30 (Rotary)
Frequency Coordination	-	0	1
Cover Pulse Reduction	–dB	0	3
Pulse Width, PW	μs	0.5	2

The pulse positions are defined in units of μs, where 1 μs equals 150 m. For a RGPO/RGPI technique, the pulse initial position is normally expected to be directly over or very close to the actual target position. The chosen bounds permit the initial pulse position to be within 750 m of the target, which is greater than the range resolution of the threat system (the SA-8 has a published acquisition radar accuracy of 300 m and an engagement radar range resolution accuracy of 55 m). The pulse final position is expected to be some distance from the target, either closer to the threat (a pull-in), or further from the target (a pull-out). The chosen bounds permit the final pulse position to be within 2250 m of the target.

The dwell times are limited to 5 s. The earliest launch time of a missile is 4 s after simulation start (this is due to threat system settings described in the next section). An initial dwell time of 5 s permits a scenario in which a missile launch occurs before the pulse accelerates away from its initial position.

The maximum fighter velocity of 600 m/s is equivalent to 1,166 knots or approximately Mach 1.75 (at sea level), which is typical of fourth generation fighter aircraft. The minimum velocity of 200 m/s or 389 knots is near the lower end of cruise speeds for fighter aircraft. The maximum rotary-wing velocity of 90 m/s is equivalent to 175 knots, which is typical of military helicopters. Since helicopters can hover (zero velocity) the minimum velocity is 5 m/s or approximately 10 knots (the pulse must be able to move during the pull-off technique).

Although fighter aircraft are capable of higher lateral accelerations (up to 9 g), the pulse acceleration is limited to approximately 6 g since the acceleration of the pulse should simulate the turn-in or turn-out of the target with respect to the threat; evasive manoeuvres above 6 g are uncommon due to the rapid energy loss associated with high load factors. Pulse acceleration for the rotary-wing aircraft was initially limited to approximately 3 g, since the design standard load factor limit for helicopters is 3.5 g, and helicopters rarely reach load factors above 2 g. The velocity and acceleration limits for rotary-wing simulations were later changed to match those of the fighter aircraft since the simulated threat system does not have the ability to distinguish between aircraft types when tracking the target or jamming pulse.

Since frequency coordination is a binary selection (it is either on or off) the variable is limited as such. Generated values less than 0.5 are rounded to 0 and values 0.5 or greater are rounded to 1.

Pulse attenuation is limited to a maximum of 3 dB, or half the maximum transmit power of the jammer.

The pulse width (PW) of the jamming pulse is limited to between 0.5 and 2 μs. The PW of the radar transmitter is 0.5 μs, and a reflected signal would be expected at the same PW. As discussed further in the results, PWs approaching 0.1 μs resulted in unexpected threat radar system behaviour.

5.2 Engagement Scenario Design

A series of engagement scenarios were developed to test the ability of the integrated system to generate suitable ECM techniques. The design involved consideration of realistic flight profiles for the target aircraft, including altitude, airspeed, flight orientation relative to the threat, and range to the threat.

The time required to conduct a single optimization round dictated that a limited number of variables could be considered when developing the engagement scenarios. As a result, the altitude and airspeed of the fighter and rotary-wing targets were held constant and only the direction of flight, or approach angle relative to the threat, was varied between optimizations. Due to time constraints, only two scenarios used a target manoeuvring at constant altitude and airspeed.

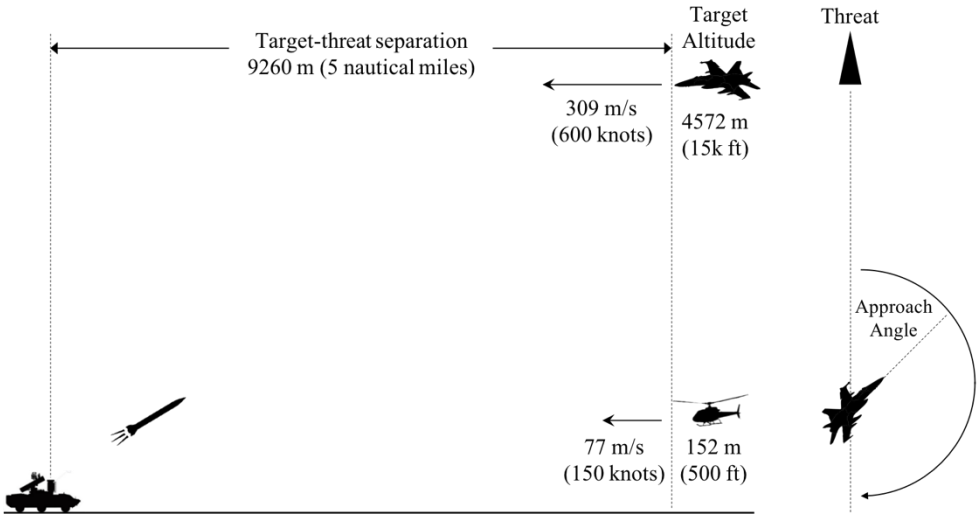
5.2.1 Scenario Setup

The threat system TTR consisted of a radar transmitter in the K_u band with a fixed PRI and a receiver type called *scan-with-compensation* (SWC), a form of monopulse tracking. A SWC system uses two Lobe-On-Receive-Only (LORO) signals, 180 degrees out of phase with each other, which are fed into separate receiver channels. Target tracking techniques such as SWC and LORO are described further in [21]. Except where noted otherwise, the radar was a coherent system (i.e. pulse-Doppler). TESS™ offers ECCM options for the threat system; by default, the track on jam option was enabled.

For all engagements, the threat platform was located due North from the target platform at a range of 9260 m (5 nautical miles) and an altitude of 5 m above sea level. The default setting for missile launch time was 0.5 s (i.e. the time required for the missile to launch). TESS™ provides an option to require the threat system to have a tracking lock for a minimum period of time before missile launch. Although the SA-8 has a published reaction time (from target detection to launch)

of 26 s [32], the simulated threat system would immediately launch upon simulation start since the target was within the engagement envelope of the threat system. To prevent immediate launch and provide for the generation of ECM techniques that may generate a repetitive break-lock condition, the post lock-on delay before launch was set to 3 s. This value was chosen since it provided an appropriate delay for the threat system. Longer delays would be unrealistic and unfairly inhibit the threat system; shorter delays resulted in a missile launch for every engagement, regardless of the ECM technique in use.

At the start of each engagement, the target platform was located due South from the threat system. The fighter target platform was at an altitude of 4572 m (15,000 feet), at a velocity of 309 m/s (600 knots). The rotary-wing target platform was at an altitude of 152 m (500 feet), at a velocity of 77 m/s (150 knots). The target platform direction of flight was defined as an approach angle relative to the threat system at the start of the simulation (i.e. an approach angle of 0 degrees indicated that the target platform was travelling due North directly toward the threat system; an approach angle of 90 degrees indicated that the target platform was travelling due East, perpendicular to the threat system). The engagement scenario geometry, showing both the side view and horizontal plan view, is depicted in Figure 5.1. Engagements ended when the missile reached its point of closest approach to the target. When no missile launch occurred, simulations were limited to a maximum of 40 s; this provided sufficient time for the threat to engage the target and resulted in the target flying directly over the threat if its approach angle was set to 0 degrees.



Note: Figure not to scale.

Figure 5.1: Engagement geometry (left: side view, right: horizontal view)

5.2.2 Scenario Variations

Some engagement scenarios were run with one or more simulation variables changed from the standard scenario setup. Engagements were run with the jammer turned off to collect baseline data between the threat system and target at each approach angle. Engagements against coherent and non-coherent TTRs were conducted to analyze the effect of frequency coordination on the generation of effective ECM techniques.

5.3 Simulation Results

For each engagement scenario, ten optimization rounds with unique random seeds were run for each algorithm. Individual ECM techniques generated during the optimization process were also analyzed through single engagements to confirm the results and to compare the algorithm performance. The simulation results are summarized in the following sections and included in tabular form in Appendix B.

5.3.1 Non-Jamming Targets

The engagement scenario was first run with each target flying the defined flight profile but with the jammer turned off. The engagements were run individually, thus no optimization was performed. The results of each engagement are given in Table 5.2 and Table 5.3, for the fighter and rotary-wing aircraft, respectively.

Table 5.2: Non-jamming fighter engagement results

Approach Angle (deg)	Miss Distance (m)	P_{kill}	P_{surv}	% Radar Track	% Radar Search
0	5.52	95.57	4.43	95.55	0.21
45	15.96	31.13	68.87	96.08	0.16
90	175.13	0	100	75.31	21.86
135	2447.10	0	100	25.93	71.77
180	920.52	0	100	98.43	0.075

Table 5.3: Non-jamming rotary-wing engagement results

Approach Angle (deg)	Miss Distance (m)	P_{kill}	P_{surv}	% Radar Track	% Radar Search
0	3.92	100	0	96.02	0.16
45	4.11	100	0	96.13	0.15
90	5.31	100	0	96.39	0.14
135	3.39	100	0	96.64	0.13
180	1.70	100	0	96.74	0.13

In the case of the fighter engagement, the probability of kill quickly dropped off when the target was not directly in-bound to the threat. At an approach angle of 90 degrees the threat radar lost its target track at 16.59 s and entered search mode, unable to re-acquire the target for the remainder of the engagement (which ended at 40 s). The target track loss occurred when the target reached a horizontal range of 10,584 m (a slant range of 11,530 m) from the threat. At 135 degrees the threat radar lost track at 7.39 s, entered search mode, and was unable to re-acquire the target. The target track loss occurred at a horizontal range of 11,538 m (a slant range of 12,411 m). At 180 degrees the target track was maintained for the duration of the engagement but the missile was unable to catch up to the target before 40 s had elapsed. The published acquisition and tracking ranges for the SA-8 are 30 km and 20 km, respectively, although the engagement range and altitude of the missile is much lower (estimated to be 10 to 12 km in altitude and up to 15 km in range). The low probability of kill was likely associated with the target moving away from the threat at high speed and at the upper altitude range of the threat's engagement range.

The threat system achieved a 100 percent probability of kill against the rotary-wing aircraft for all approach angles simulated. This is hypothesized to be due to the fact that the target was travelling at relatively low speed at a low altitude, well within the engagement range of the threat system.

Although the simulated threat system is not particularly effective against a fast-moving, medium to high altitude target, the ability of the ECM technique generation system can still be evaluated. Techniques that either prevent missile launch or increase the miss distance may exist which improve upon the benefit of altitude and airspeed against the threat system. The initial engagement results indicate that only approach angles from 0 through 90 degrees require further study when using the fighter target platform; whereas approach angles from 0 through 180 degrees should be explored with the rotary-wing target platform.

5.3.2 Generated ECM Techniques

Jammer range techniques generated by both algorithms converged to one of two forms: short-duration pulses, or long-duration pulses. Pulses with a total duration of less than 8 s (i.e. the total time inclusive of dwell times and the time required to move from the initial position to the final position using the technique acceleration rate) captured the TTR range and velocity gates (for a coherent threat) and forced the threat radar to continually cycle between its search, acquisition, and track modes, as shown in Figure 5.2. For approach angles between 0 and 45 degrees and 135 and 180 degrees, pulses of this type prevented a track lock of sufficient time to permit a missile launch (i.e. no-launch condition) and received fitness scores of 0. However, the threshold of 8 s for pulse duration was not observed to be fixed; for ten unique engagement scenarios there were six individual techniques that had a total duration longer than 8 s (up to 14.01 s) and yet prevented a missile launch.

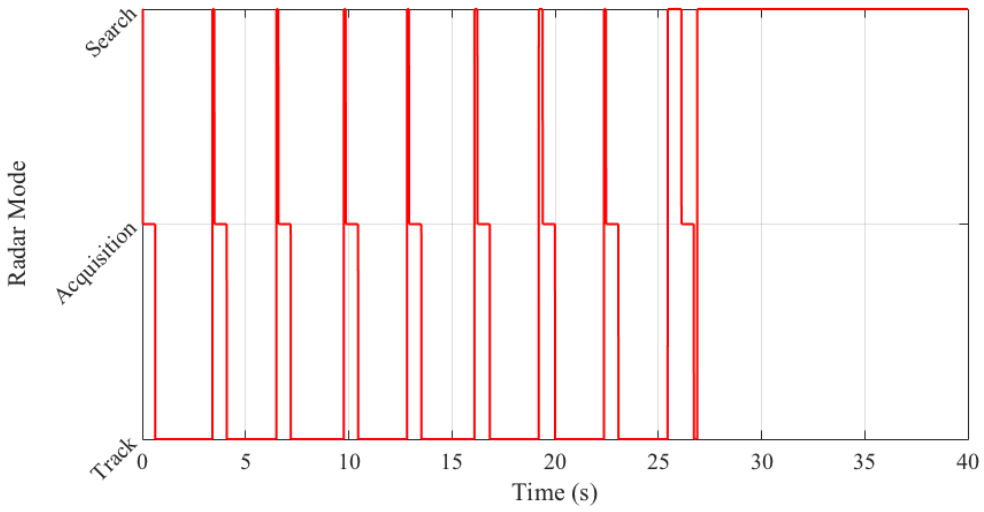


Figure 5.2: Radar mode for no missile launch (typical)

Technique pulses with a total duration greater than 8 s were still able to capture the TTR range and velocity gates; however, since the threat radar

maintained a track (on the false target pulse), missile launch occurred. Such techniques resulted in the missile reaching its point of closest approach to the target while the TTR was in search mode, as shown in Figure 5.3. This implies that the command guidance control to the missile was providing incorrect tracking, which is a desirable result. In this case, techniques that were able to maximize the missile miss distance received fitness scores of 0.1 (for miss distances greater than 500 m). However, as with the short duration pulses above, the 8 s threshold was not fixed, since for the same ten unique engagement scenarios there were three individual techniques that had a total duration shorter than 8 s (as low as 4.1 s) that did not prevent a missile launch.

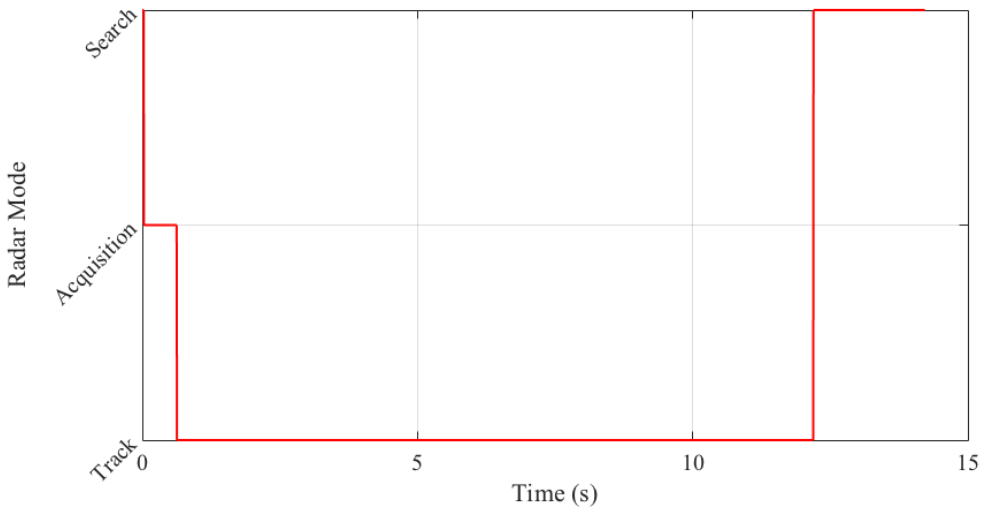


Figure 5.3: Radar mode for large missile miss distance (typical)

The short-duration and long-duration pulse types described above consist of very different pulse parameters. In terms of the defined solution space and fitness function, techniques capable of preventing a missile launch exist near a global minimum (within the bounded solution space); whereas, techniques that maximize miss distance exist near a local minimum. These global and local minima are far apart within the solution space (e.g. short duration pulses vs. long duration pulses).

5.3.3 Target Approach Angle

The approach angle between the target aircraft direction of flight and the threat system was varied from 0 to 180 degrees at 45 degree increments between optimization runs. Since the probability of kill for the fighter at 135 and 180 degree approach angles was zero, with associated large miss distances, engagements at those angles were not run for the fighter.

ECM techniques generated for both the fighter and rotary-wing aircraft indicate that prevention of a missile launch is possible at approach angles of 0 and 45 degrees. In addition, rotary-wing techniques indicate that prevention of a missile launch is also possible at approach angles of 135 and 180 degrees. Neither algorithm found ECM techniques that prevent a missile launch at approach angles of 90 degrees. This is likely due to the geometry of the jammer antenna patterns, which are forward and aft facing with 60 degree beamwidths. At a 90 degree approach angle, very little jammer transmitted energy would be received at the threat radar.

5.3.4 Algorithm Performance

For the engagement scenarios and system specifications tested, the PSO was consistently better at converging to solutions with fitness scores of 0 (i.e. converging to 0), requiring fewer iterations, on average, than the GA for the same scenarios. Of twelve unique engagement scenarios, ten yielded a no-launch result (i.e. a fitness score of 0) from either algorithm. The GA converged to 0 for seven of the scenarios, whereas the PSO converged to 0 for nine scenarios. Given that there are ten rounds per scenario, per algorithm, out of 100 optimization rounds, the PSO converged to 0 a total of 65 times; the GA converged to 0 only 36 times.

The PSO algorithm methodology likely lends itself better to this problem type, when compared to the GA. Using the default PSO options, the PSO performed well when searching the solution space. The fact that every particle has its velocity and position updated based on both its own performance and the performance of the swarm means that every particle will improve over time; however, with particles moving about the solution space there is also a good chance that the global best position can change if a single particle nears a global minimum.

The GA is limited to random changes in a portion of the population based on the previous performance of selected population members. If the population begins to converge to a local minimum there is less chance that convergence to the global minimum will be achieved. The mutation function is a GA option intended to prevent premature convergence to a local minimum and allow for a wider search of the solution space. The default function in MATLAB® is called *mutationgaussian*, which as the name implies, generates mutation from a Gaussian distribution with a standard deviation that is scaled by a recursive formula between generations [41]. The level of mutation provided by the default option is likely insufficient for this problem.

One scenario in which the PSO converged to 0 in five of ten optimization rounds, but where the GA never converged to 0, was rerun for the GA with different mutation options. Five optimization rounds were run using the GA mutation function *mutationuniform* with a mutation rate of 0.05 (i.e. 5 percent). The GA converged to 0 in only one of the five rounds (requiring 4 generations). A higher mutation rate may improve the performance of the GA; however, the requirement to optimize the mutation rate is one of the weaknesses of the GA. Performance may improve as a function of mutation rate, but assessing such an improvement consumes time; this process is unnecessary with the PSO.

5.3.5 Algorithm Speed and Execution Time

In terms of algorithm speed in completing each generation/iteration, the GA and PSO were equivalent. The mean time required per generation for the GA was only slightly shorter (around 1 to 2 min) than the mean time required per iteration for the PSO, as shown in Table 5.4. This is likely due to the standard time required to conduct a single engagement simulation via the TESS™ Simulink® model, which is significantly longer than any other individual process carried out by each optimization algorithm (such as initialization, population/swarm creation, and candidate solution updates at each generation/iteration).

Table 5.4: Optimization algorithm mean execution time

Scenario Type	**GA – Mean Time Per Generation (min)**	**PSO – Mean Time Per Iteration (min)**
Fighter vs Coherent Threat (0, 45, 90 deg)	3.40[1]	3.79[1]
Fighter vs Non-Coherent Threat (0, 45 deg)	12.33	13.48
Manoeuvring Fighter vs Coherent Threat (0, 45 deg)	13.68	15.55
Rotary-Wing vs Coherent Threat (0, 45, 90, 135, 180 deg)	14.11	15.20

Note: 1. Following a software license extension for TESS™, issued by TTI, execution times inexplicably increased by a factor of about 4 to 5 times for both algorithms.

The total execution time required to converge to a solution is a better measurement to perform a comparison. As stated in the previous section, the PSO converged to 0 more often and required fewer iterations to do so, on average, than the GA. Thus, the PSO consistently took less time to converge to 0, despite the fact that the mean time per iteration for the PSO was 1 to 2 min longer than the mean time per generation for the GA. For example, for the ten unique scenarios where a no-launch result was found (by either algorithm), the mean number of iterations required for the PSO to converge to 0 (in the 65 optimization rounds that did so),

was 3.9 iterations; whereas, the GA required a mean of 4.5 generations to converge to 0 (in the 36 optimization rounds that did so). The PSO reveals itself to be the faster of the two algorithms. Examples of the algorithms converging to 0 are shown in Figure 5.4 and Figure 5.5, for the GA and PSO, respectively.

Where the algorithm stalled and converged to 0.1, and the number of stall generations/iterations (a user-defined optimization option) was equal, both algorithms took, on average, the same amount of time. Examples of the algorithms stalling at fitness scores of 0.1 are shown in Figure 5.6 and Figure 5.7, for the GA and PSO, respectively. The best fitness score in the current population/swarm is shown for each generation/iteration, along with the mean fitness of all the candidate solutions in the current population/swarm for each generation/iteration.

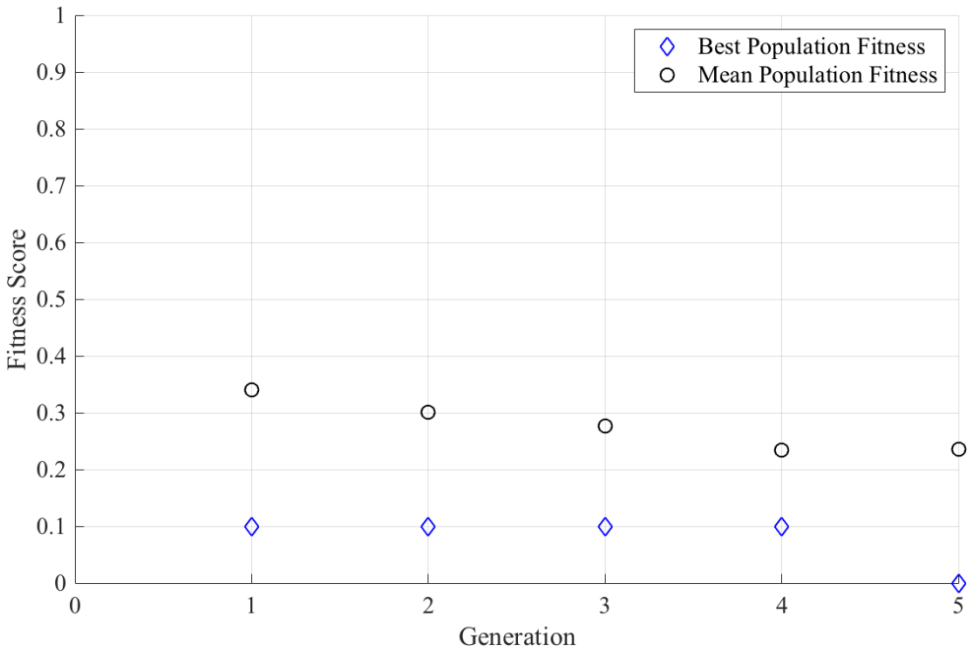


Figure 5.4: GA convergence to 0 (typical)

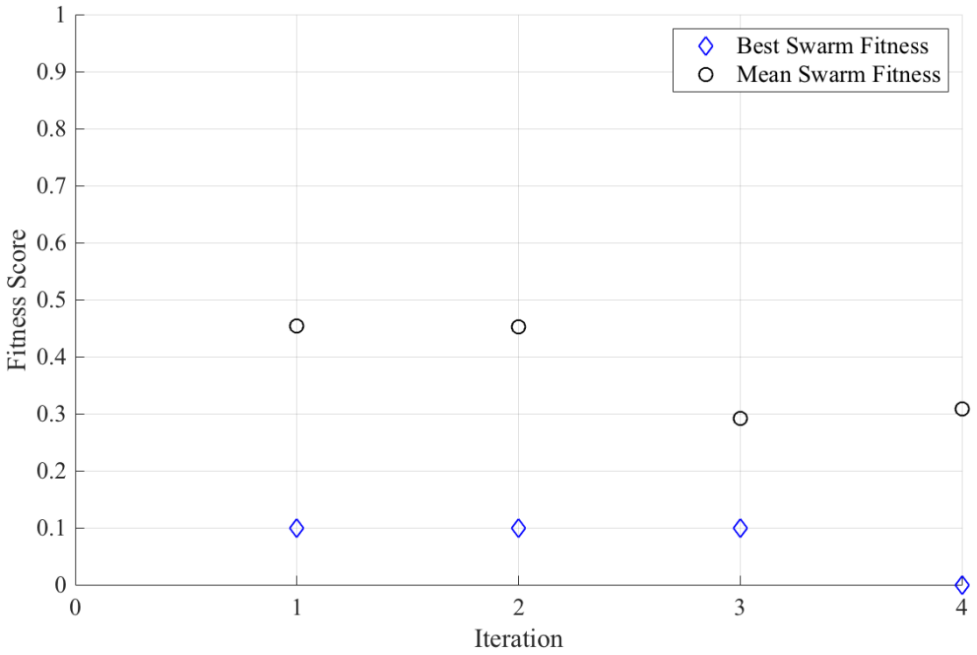


Figure 5.5: PS convergence to 0 (typical)

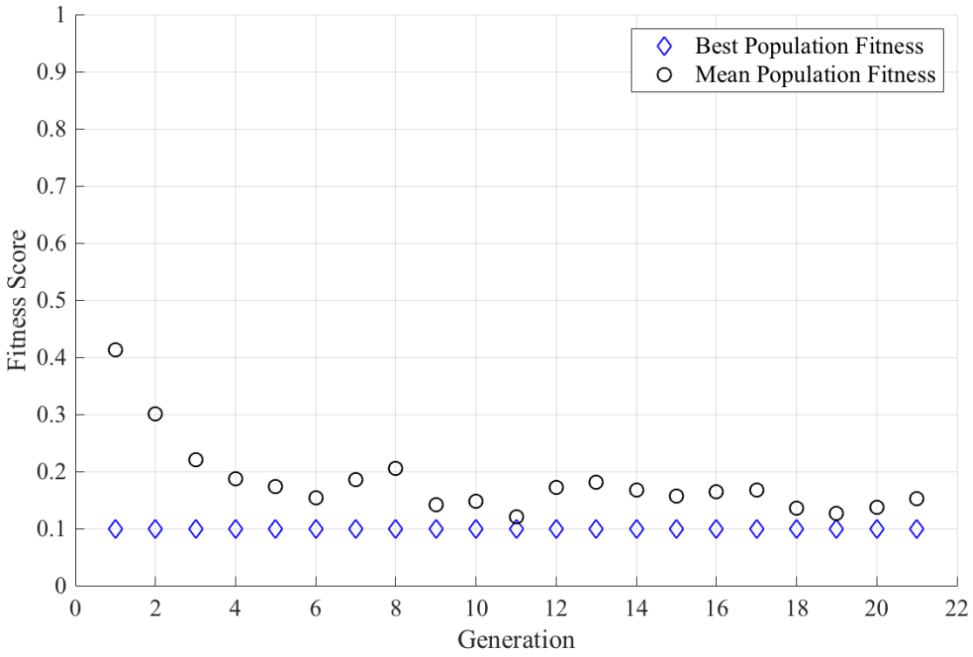


Figure 5.6: GA stall at 0.1 (typical)

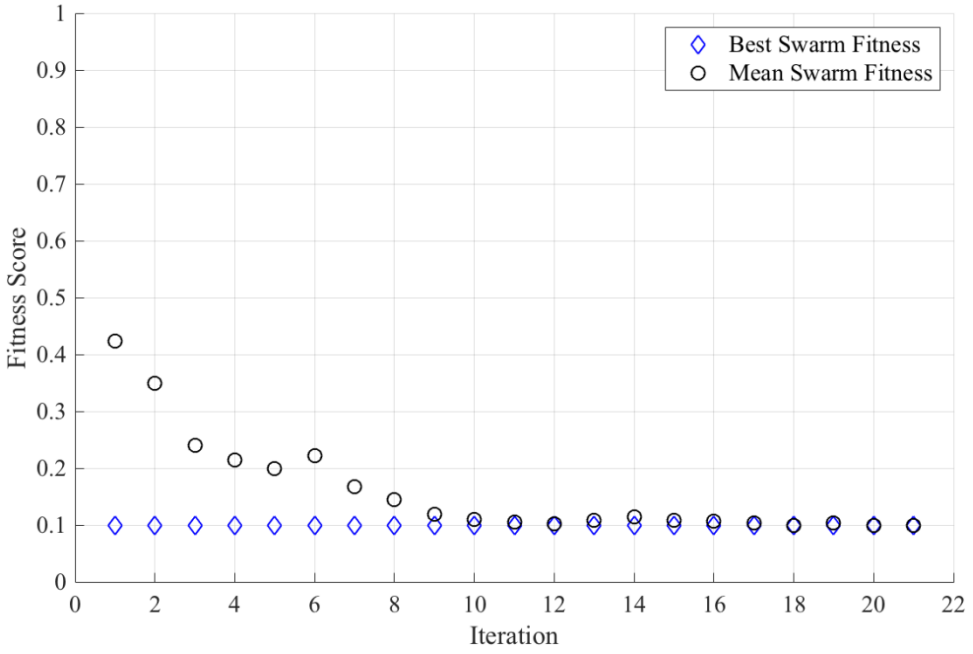


Figure 5.7: PS stall at 0.1 (typical)

5.3.6 Algorithm Convergence

As previously stated, the PSO performed better in terms of the number of approach angles for which a no-launch condition was achieved (i.e. 9 for the PSO vs. 7 for the GA), the number of individual ECM techniques generated that achieved a no-launch condition (i.e. 65 for the PSO vs. 36 for the GA), and the number of iterations required to converge (thus the total execution time). However, both algorithms demonstrate interesting behaviour when the convergence process is analyzed more closely.

In general, when conducting an optimization with the PSO, the mean fitness score of the swarm would decrease quickly (within five to ten iterations) to a value near 0.1 (mean fitness scores between 0.11 and 0.15). The mean fitness score would then remain stable between 0.1 and 0.11 for the remaining iterations until either the best fitness score decreased further to 0 or the maximum number of stall iterations was reached. Figure 5.7 provides an example of this typical behaviour of the PSO algorithm.

Conversely, when conducting an optimization with the GA, the mean fitness score of the population would initially decrease (normally at a lower rate than the PSO) from its starting value but then continue to randomly change by as much as ±0.11 between generations. This suggests that the process of mutation between generations continues the random search of the solution space while the algorithm is stalled near a local minimum. This is an example of how the GA might be better at exploring the solution space while at a local minimum; however, despite this quality, it performed worse than the PSO. It is hypothesized that a larger stall limit may provide a better chance for the GA to escape from the local minimum than the PSO has, but at the expense of a longer simulation time. Figure 5.6 provides an example of this typical behaviour of the GA; Figure 5.8 is an example of an extreme case of the mean fitness changing between generations.

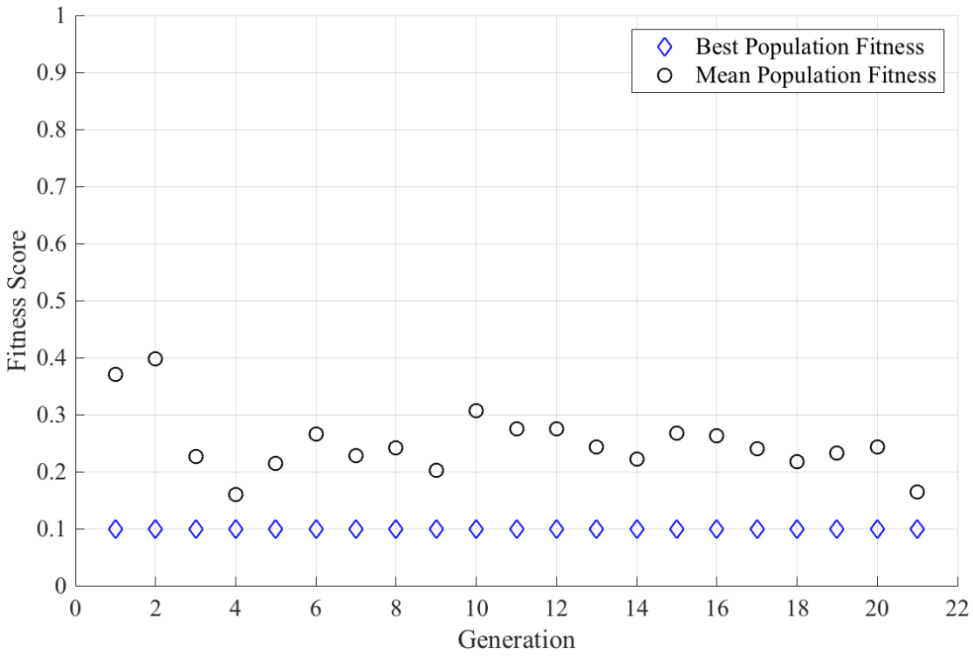


Figure 5.8: GA mean fitness changes between generations

As presented previously, five optimization rounds were run using the GA mutation function *mutationuniform* with a mutation rate of 0.05 (i.e. 5 percent). Use of this mutation function for the optimization process generated convergence to 0 for one round, where the default mutation function was unable to do so for the same scenario in ten rounds. Both optimizations had a maximum of 50 stall generations. The four rounds that stalled at a best fitness score of 0.1 showed less random oscillation in the mean fitness score between generations; however, these results are not sufficient to draw a conclusion regarding the change in mutation rate, since only one round converged to 0, which could be considered *luck* for the GA. An example of the GA optimization using the mutation function *mutationuniform* with a mutation rate of 0.05 is shown in Figure 5.9.

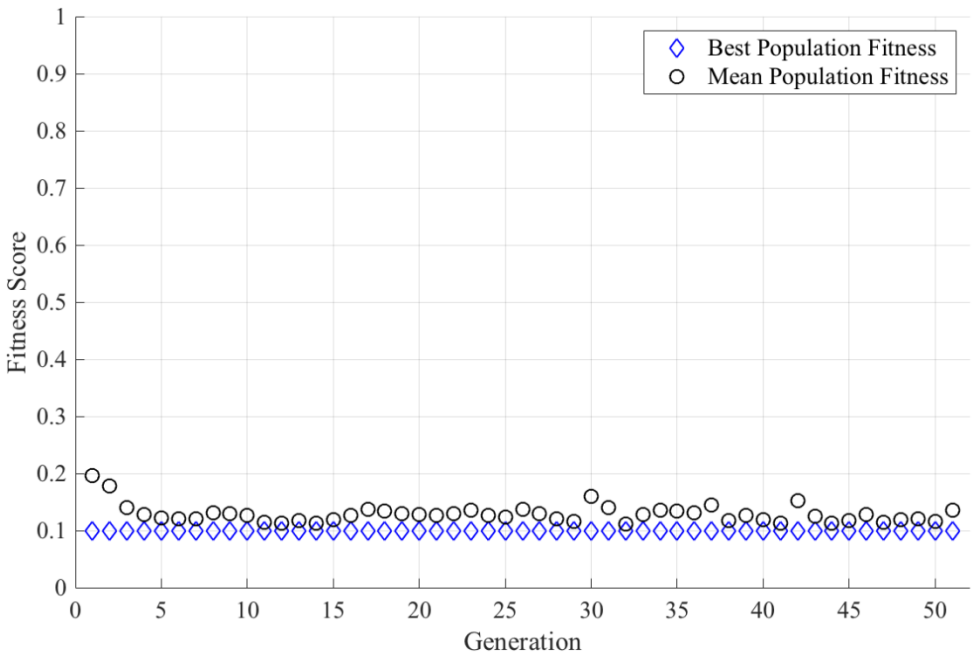


Figure 5.9: GA mean fitness changes between generations with uniform mutation rate 0.05

The convergence behaviour of both algorithms was markedly different for one engagement scenario when compared to all other scenarios. An engagement between the rotary-wing aircraft and the coherent threat radar system was simulated at an approach angle of 90 degrees, using the fighter velocity and acceleration rate upper and lower bounds. In ten optimization rounds for each algorithm, neither algorithm was able to converge to fitness scores of either 0.1 or 0.

The lowest fitness score achieved by the GA was 0.303, which corresponded to a missile miss distance of 93.35 m. That same round also took the greatest number of generations (25) to converge for this scenario. The nine other rounds took between 21 and 24 generations to converge to fitness scores between 0.321 and 0.340. An example of the GA convergence behaviour is shown in Figure 5.10.

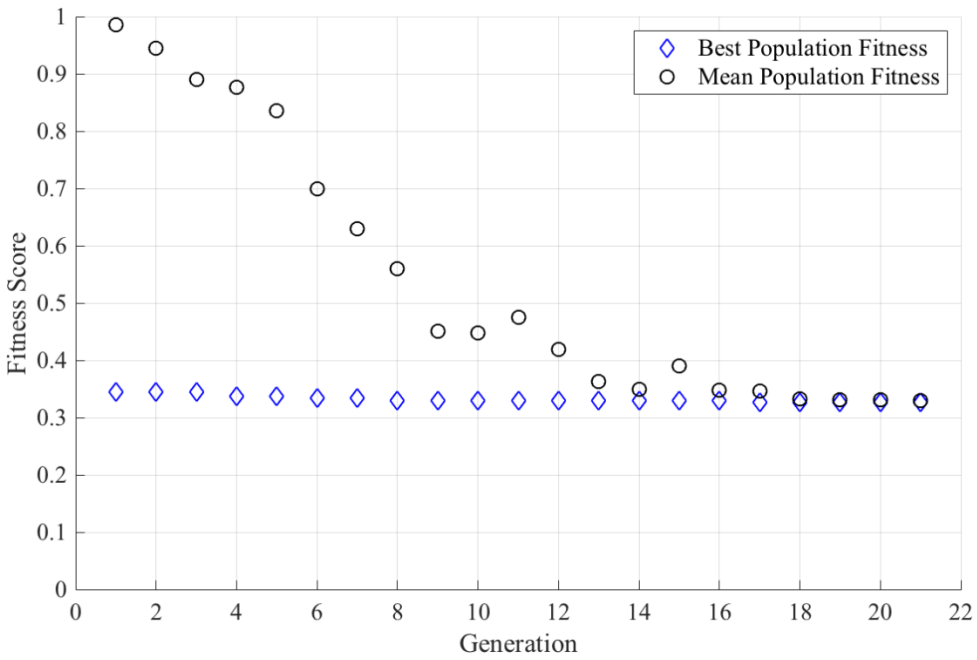


Figure 5.10: GA convergence for rotary-wing at 90 degrees (typical)

The lowest fitness score achieved by the PSO was 0.279, which corresponded to a missile miss distance of 142.44 m. Surprisingly, the PSO required between 26 and 81 iterations to converge during the ten optimization rounds; the worst fitness score achieved was 0.334 in 26 iterations. For most of the optimization rounds, the mean fitness score remained relatively high (between 0.8 and 1.0) for as many as 50 iterations and never came any closer than 0.25 above the best fitness score. Therefore, the majority of candidate solutions generated by the PSO were ineffective. An example of the PSO convergence behaviour is shown in Figure 5.11.

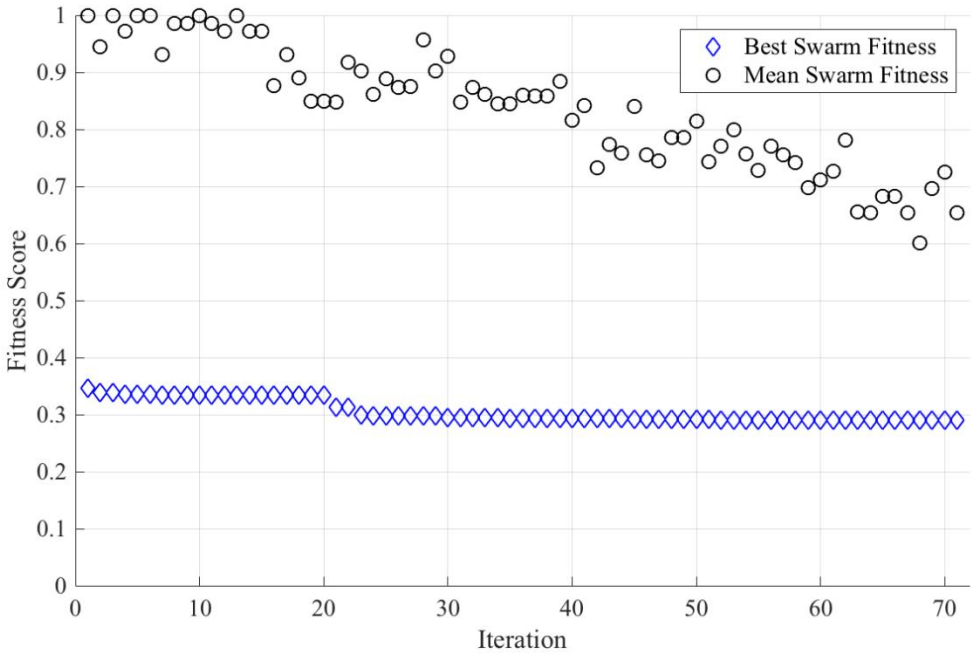


Figure 5.11: PSO convergence for rotary-wing at 90 degrees (typical)

Although the GA performed better than the PSO for this scenario, this is likely an example of a scenario for which the jammer offers very limited protection. In this case, one algorithm is not favoured over the other. However, for the vast majority of engagement scenarios investigated, the PSO provided better convergence rates and more optimal results.

5.3.7 Frequency Coordinated Techniques

Against the threat system with a coherent TTR, ECM techniques, with two exceptions, were generated with frequency coordination included. Where techniques did not include frequency coordination, the techniques were normally unable to prevent a missile launch (i.e. fitness scores greater than or equal to 0.1). Two techniques were generated without frequency coordination that were able to prevent a missile launch (i.e. fitness scores of 0) (rotary-wing at 45 and

180 degrees approach angle with fighter optimization bounds; PSO algorithm). Techniques that included frequency coordination and were able to prevent missile launch were subsequently unable to achieve the same result if frequency coordination was disabled. This indicates that frequency coordination is an important parameter in the optimization process against this particular threat.

Against the non-coherent threat, frequency coordination had no influence over the technique fitness scores (i.e. techniques that were generated with frequency coordination could have that parameter disabled and still achieve the same fitness score).

5.3.8 Jammer Pulse Width Effects

By not bounding the search space, or using bounds larger than one would normally assign to a given problem, the GA and PSO are capable of finding novel solutions that may not be assumed to exist. An example that highlights this benefit follows.

Initially, the lower bound on PW was set to 0.1 μs. During initial optimization rounds, using both the GA and PSO, generated techniques with PWs approaching the lower bound (0.1 μs) did not require frequency coordination to prevent missile launch (i.e. fitness score of 0). Such a narrow PW would not usually be considered as a potential solution, as the radar is tuned to expect echoes having the same PW as that which are emitted. Visual analysis of the radar range, azimuth, elevation, and Doppler traces for one such engagement, shown in Figure 5.12, Figure 5.13, Figure 5.14, and Figure 5.15, respectively, show that although the threat radar attempted to track the false target pulse, the tracking position oscillated around the pulse position in range, azimuth, and Doppler. In elevation, the radar position oscillated around a gradually decreasing elevation, tracking neither the target nor the false target pulse. This resulted in the threat radar continually cycling between its search, acquisition, and track modes, as shown in Figure 5.16, preventing a track lock of sufficient time to permit a missile launch.

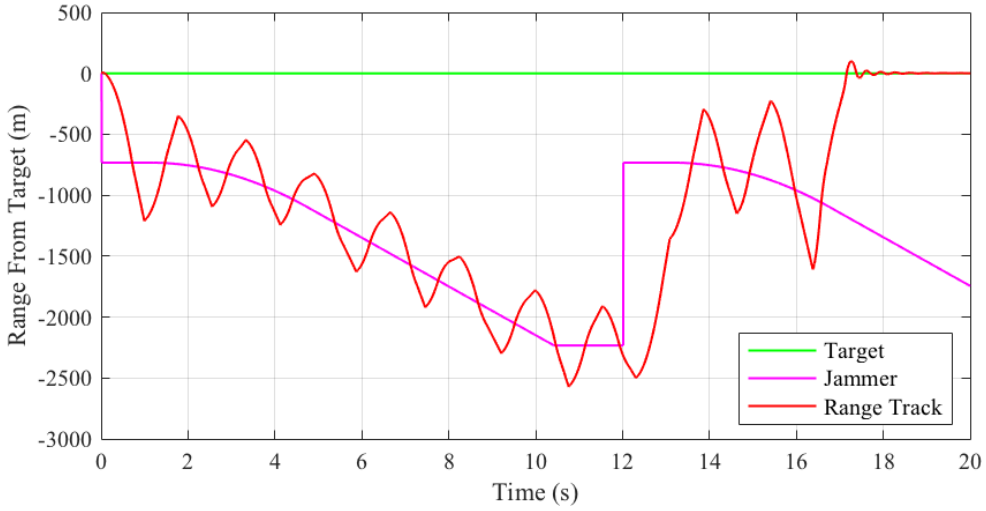


Figure 5.12: Radar range tracking for 0.1 μs jammer PW

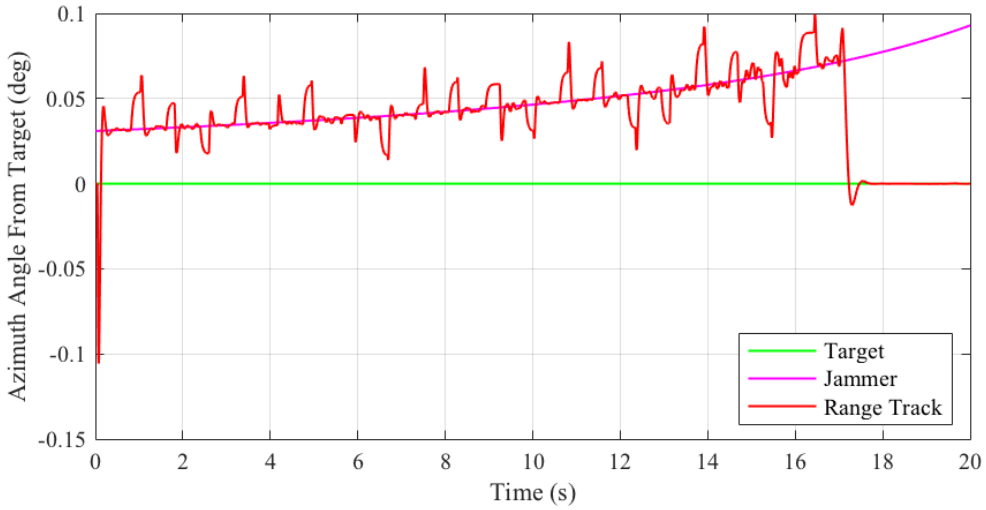


Figure 5.13: Radar azimuth tracking for 0.1 μs jammer PW

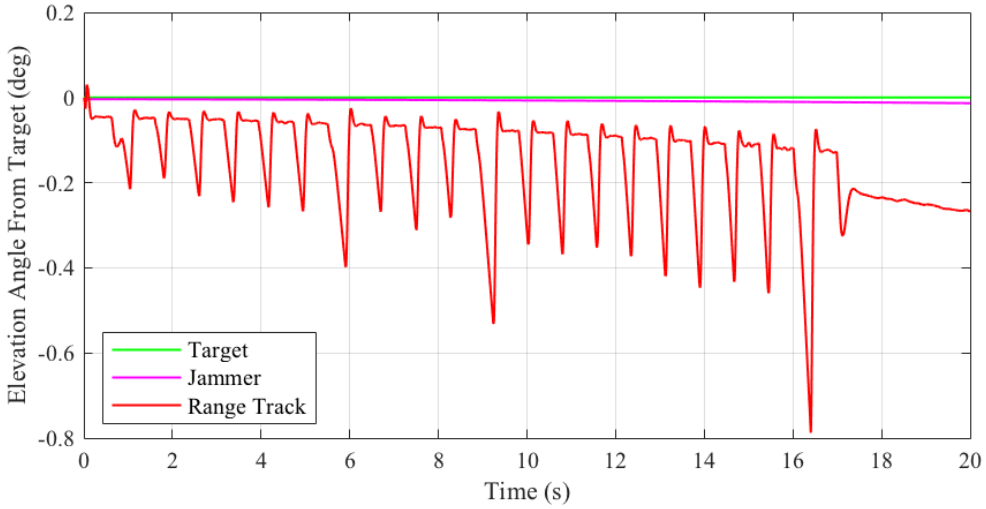


Figure 5.14: Radar elevation tracking for 0.1 μs jammer PW

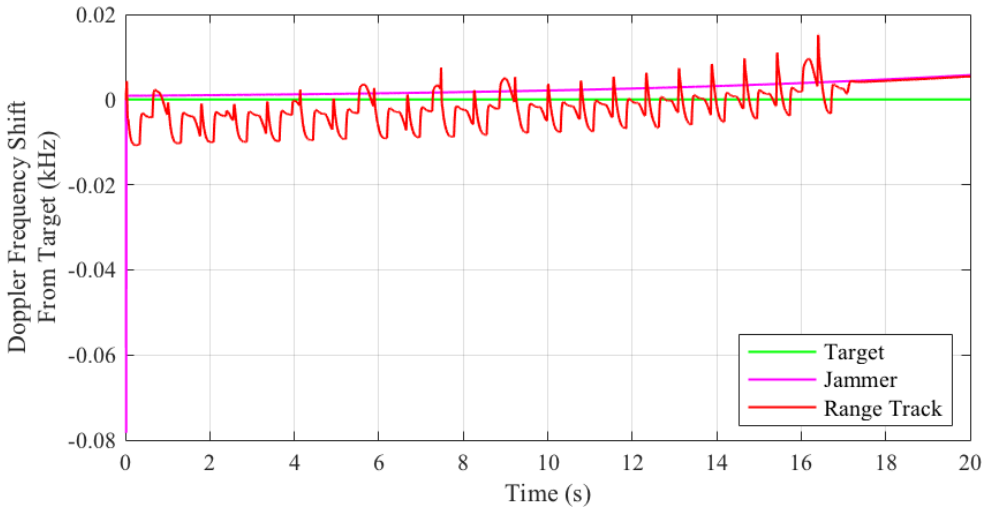


Figure 5.15: Radar Doppler frequency tracking for 0.1 μs jammer PW

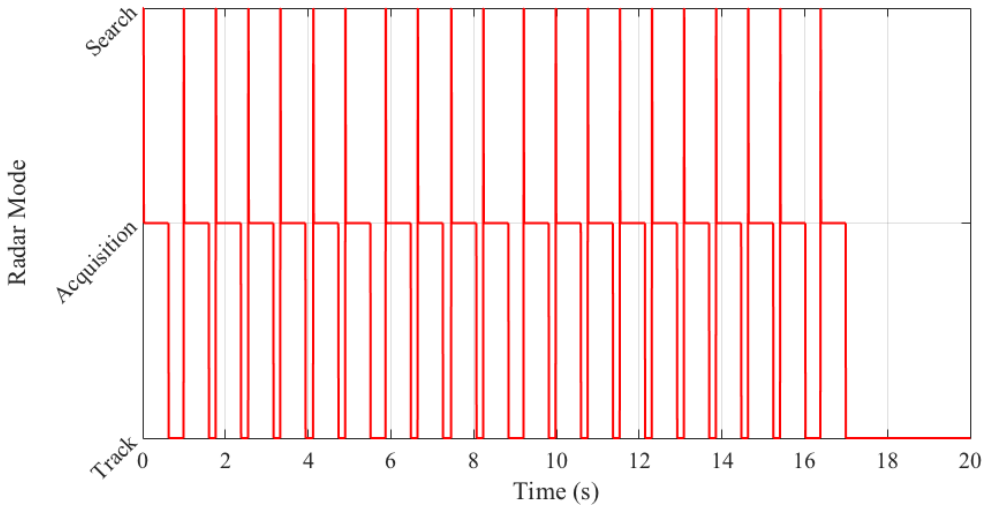


Figure 5.16: Radar mode for 0.1 μs jammer PW

This characteristic may be related to the design of the TESS™ Simulink® model threat system and the resonance frequency of one of its tracking loops. At such a small PW, the resulting bandwidth may have overwhelmed the modelled receiver tracking circuit, causing the oscillation. This result is an example of stochastic optimization techniques finding unexpected solutions, since such effective jamming would not be anticipated from use of such a short PW. Since this was likely a limitation of the model, the PW lower bound was set to 0.5 μs for all subsequent simulations.

5.3.9 Jammer Pulse Power Reduction

Within the defined bounds, where the pulse power reduction ranged from 0 dB (i.e. full jammer transmit power) to 3 dB (i.e. half the jammer transmit power), there was no correlation between the generated value and effectiveness of the associated technique. In a number of cases, techniques with similar values for all other parameters had pulse power reduction values at the extremes of the defined range, with equal fitness scores. Techniques capable of preventing missile launch were re-simulated under the same conditions with only the pulse power reduction value varied; technique performance was unchanged.

With the target beginning at 5 nautical miles and flying directly toward the threat, the unreduced jammer power in the radar tracking cell remained in the range of –70 to –80 dBW; the power of the target return ranged from –120 to –100 dBW, as shown in Figure 5.17. Burn-through occurred between 25.8 and 26.2 s into the engagement, when the target was between 1.16 and 1.29 km (horizontal range) from the threat. Burn-through occurred at such a close range to the threat that the TTR was unable to reacquire and track the target.

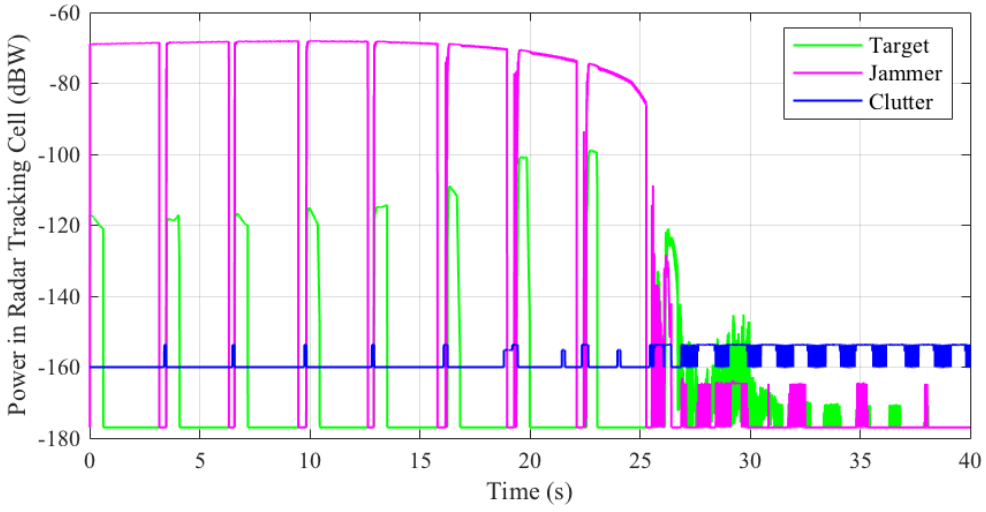


Figure 5.17: Radar tracking cell power levels for an unreduced jammer pulse

5.3.10 Target Manoeuvring

Due to time limitations, only two engagement scenarios with a manoeuvring target aircraft were simulated against a threat system with a coherent TTR. The fighter aircraft conducted a 3 g level left turn for 10 s, starting at 1 s after the engagement start. The target aircraft flight profiles are shown in Figure 5.18 and Figure 5.19 for initial approach angles of 0 and 45 degrees, respectively.

For both scenarios, both algorithms generated ECM techniques capable of preventing a missile launch. At 0 degrees, the PSO converged to 0 for all 10 optimization rounds, whereas the GA only converged to 0 for two rounds. At 45 degrees, the PSO converged to 0 for all but one round, whereas the GA converged to 0 in seven out of ten rounds. As with previous scenarios, the PSO converged faster, requiring fewer iterations, on average, than the GA. Both algorithms are therefore capable of generating ECM techniques for manoeuvring target profiles. Further research into various manoeuvring profiles combined with jamming techniques is recommended.

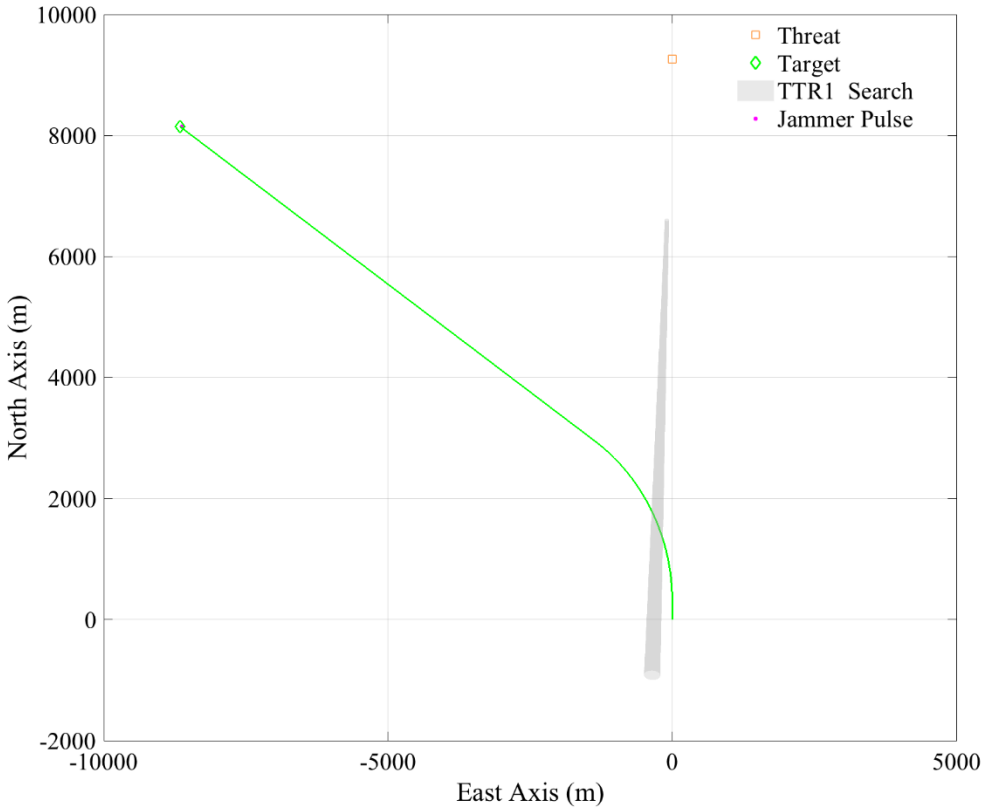


Figure 5.18: Fighter manoeuvre profile, 0 degree initial approach, 3 g left turn

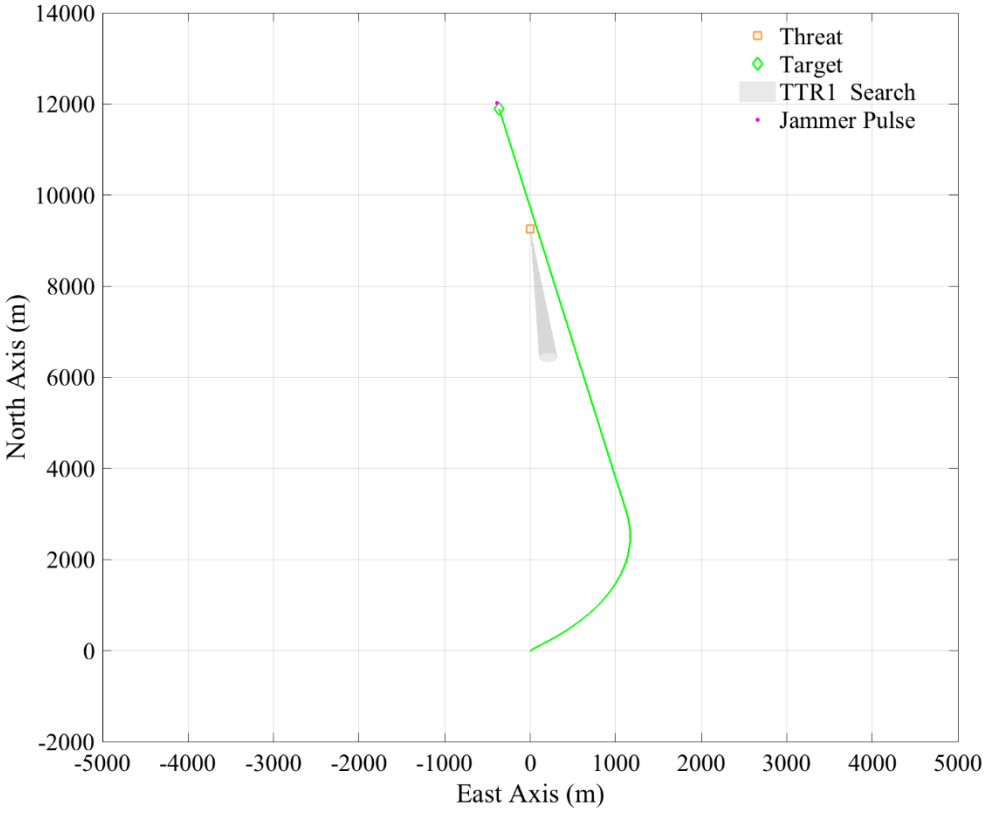


Figure 5.19: Fighter manoeuvre profile, 45 degree initial approach, 3 g left turn

5.4 Summary

This chapter discussed the optimization setup, including the optimization options and solution space bounds. The selection of specific values and their effect on the optimization process was highlighted. Commonalities between each unique engagement scenario, including the simulation conditions and their role in scoping the overall execution, was also examined.

The simulation results were presented, beginning with the engagement scenario outcomes for a non-jamming target aircraft. Those simulations indicated that the threat system was ineffective against a fighter aircraft at typical speeds and

altitudes, when the approach angle between the target and threat was between 90 and 180 degrees. The results of early engagement scenarios helped to limit the number of scenarios required for the comparison and contributed to the selection of solution space bounds and optimization options.

It was shown that jammer range techniques generated by both algorithms took on one of two forms: those that prevented a missile launch or those that maximized the missile miss distance. The way in which the fitness function was defined for this thesis meant that the optimization algorithms were judged based on their ability to find a no-launch solution (i.e. a single objective). The distance between the two solution types, within the solution space, tested each algorithm's ability to explore a wide range of the solution space in a minimal amount of time.

The effect of each ECM technique parameter on the overall performance of the technique was also analyzed. The nine parameters that defined a combined range and frequency deception jammer program in TESS™ were shown to have varying levels of importance when implemented against the defined threat system. Initial and final positions were of less importance as long as the initial and final dwell times, velocity, and acceleration were of appropriate values to generate a short, fast pulse. Frequency coordination was crucial against coherent threat systems, as expected. Cover pulse attenuation was not a significant factor in technique performance unless it was of sufficient magnitude to permit burn-through while the TTR was still tracking the target aircraft or false target. The jammer PW, if set relatively close to the TTR PW, played no discernable roll in technique performance; however, as the PW approached 0.1 μs, an oscillation in the target tracking system of the TTR appeared, resulting in the threat radar continually cycling between its search, acquisition, and track modes. Although this method was effective in jamming the threat radar, it remains to be determined whether this was a characteristic of the TESS™ Simulink® model or whether an actual system would be similarly affected.

Finally, the flight dynamics of the target aircraft were found to have a significant effect on the ECM techniques generated and the time required to carry out the optimization. The distance from the threat system, as well as the approach angle relative to the threat, determined whether sufficient jammer power was received at the TTR so as to have an effect on the threat radar receiver and target tracking system. Level flight manoeuvring (i.e. level, constant-speed turns) did not reduce the ability of either algorithm to converge to a solution; ECM techniques that prevented missile launch were found for engagement scenarios with manoeuvring aircraft. Further investigation into more dynamic target manoeuvring is recommended.

6 Conclusion

6.1 Summary

As radar systems continue to evolve and adapt, so too must ECM techniques to remain effective against modern threats. Stochastic global optimization has provided an alternative to direct-search methods when searching for solutions to complex problems. However, evolutionary heuristics such as the GA and PSO have not previously been used to generate ECM range and frequency deception jamming techniques.

Through the use of test functions, the GA and PSO algorithms were compared for overall speed, accuracy (i.e. convergence to known solutions), and simplicity of use. The PSO was shown to be more efficient, in that it converged to solutions closer to the known function values, and in less time than the GA. The PSO also had fewer options to control the optimization process, making it more simplistic and easier to use.

The software integration of the MATLAB® Global Optimization Toolbox™ with the proprietary TESS™ product was not a trivial task, as TESS™ had not been used in this context before. Fortunately, the fact that TESS™ is based on MATLAB®/Simulink® allowed for a successful integration, which had not previously been accomplished. Without the availability of guidance or experience from previous work, a fitness function capable of determining the effectiveness of a candidate solution ECM technique was developed. The fitness function, when combined with the simulation and scoring system and the TESS™ Simulink®

model, generated a fitness score based on the ability of the ECM technique to either prevent a missile launch or maximize the missile miss distance. Successful parallelization of the simulation and scoring functions allowed ECM technique generation to be performed in hours instead of days. The increase in speed permitted an increase in the population/swarm size and the number of individual rounds for each optimization algorithm, as well as the use of multiple engagement geometries.

Optimization options and solution space bounds were chosen to maximize convergence speed while maintaining a large solution space from which to draw effective ECM techniques. The engagement scenarios were designed to simulate a CG SAM threat system with a pulsed TTR and either fighter or rotary-wing aircraft using a self-protection jammer. Only ECM deception jamming techniques that provide false range and velocity information were generated.

6.2 Conclusions

Both the GA and PSO were successful in finding effective ECM techniques for the simulated jammer to use against the threat radar system. The ECM range and frequency deception jamming techniques generated by both the GA and PSO algorithms took on one of two forms: those that prevented a missile launch or those that maximized the missile miss distance. For this thesis, the fitness function was defined such that the optimization algorithms were judged based on their ability to find a no-launch solution, with less-favourable fitness scores based on the missile miss distance produced by the ECM technique. The distance between the two solution types, within the solution space, required each algorithm to explore a wide range of the solution space in a minimal amount of time. Although both algorithms were successful in finding ECM techniques capable of preventing a missile launch, the tendency for both algorithms to stall in local minima resulted in the algorithms

failing to find the (bounded) global minimum on a number of optimization rounds where the global minimum existed in the bounded solution space.

The PSO demonstrated superior performance both in terms of the number of ECM techniques that reached the defined global optimum (65 for the PSO vs. 36 for the GA), and the total time to converge to those solutions. In most cases, the PSO converged faster to the defined optimal solution, doing so in an average of 3.88 iterations, as compared to 4.5 generations for the GA, or stabilized much more quickly to a solution with a large missile miss distance. However, the GA was still capable of generating effective ECM techniques. The GA continues to explore the solution space in a stalled state depending on the mutation options selected; however, no discernable improvement in performance during a stalled state was shown in this work. The ECM technique generation system developed for this thesis was most effective when using the PSO algorithm.

6.3 Contributions

The most important contributions of this thesis are:

1. The design and implementation of an automated system, bridging the proprietary TESS™ Simulink® model with the MATLAB® Global Optimization Toolbox™, capable of generating effective ECM range and frequency deception jamming techniques through stochastic global optimization.
2. Successful parallelization of the above process via the MATLAB® Parallel Computing Toolbox™, permitting a reduction in computation time by 5.72 times for this specific problem.
3. Definition of a single-objective fitness function and implementation of a scoring system for determining ECM technique effectiveness against CG SAM threat radar systems.

4. Scenario-based comparison of the GA and PSO global optimization algorithms when generating ECM range and frequency deception jamming techniques.
5. Validation of the GA and PSO algorithms for the generation of effective ECM techniques, and identification of suitable optimization parameters for algorithm performance.
6. Identification of the most important ECM technique parameters when generating techniques through stochastic global optimization.

The successful integration of the proprietary TESS™ Simulink® model with the MATLAB® Global Optimization Toolbox™ and Parallel Computing Toolbox™ was shown to be effective in automatically generating ECM techniques through stochastic global optimization. The resulting system may be useful in discovering new or previously unrealized ECM techniques, in less time than conventional means.

6.4 Future Work

Through the completion of this thesis, three areas of study requiring future work were identified. First, the definition of the fitness function used to score engagement scenario results and to determine the effectiveness of a given technique should be further explored. Other simulation outputs such as the angular, range, and Doppler error of the TTR may provide further valuable information for the scoring process.

Second, the performance of the GA may be significantly improved through better selection of optimization options. For example, changes to the mutation function and mutation rate may increase the probability of the GA converging to the optimal solution faster, if the defined global optimum exists. Although the PSO was shown to be faster and more simplistic in its use, the GA provides additional optimization options not available from the PSO.

Finally, use of the ECM technique generation system developed for this thesis should be demonstrated using the specifications and system parameters of actual threat systems, target aircraft, and self-protection jammers. Such simulations would be of a classified nature, but may demonstrate the suitability of the developed system for application to the processes of technique generation, validation, and DECM system programming.

Bibliography

[1] N. Polmar, "The U. S. Navy: Electronic Warfare (Part 2)," *U.S. Naval Institute | Proceedings Magazine*, vol. 105, no. 921, Nov-1979.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Evolutionary Computation 1: Basic Algorithms and Operators*. Philadelphia, PA: Institute of Physics Publishing, Ltd., 2000.

[3] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed. New York, NY: Springer Science+Business Media, LLC, 2007.

[4] Y. Rahmat-Samii and D. Gies, "Genetic algorithm (GA) and particle swarm optimization (PSO): powerful paradigms for unconventional designs," in *Proceedings of 2004 URSI – EMTS, International Symposium on Electromagnetic Theory*, Pisa, Italy, 2004, pp. 957–959.

[5] R. L. Haupt, "An Introduction to Genetic Algorithms for Electromagnetics," *IEEE Antennas and Propagation Magazine*, vol. 37, no. 2, pp. 7–15, Apr. 1995.

[6] R. L. Haupt and D. H. Werner, *Genetic Algorithms in Electromagnetics*. Hoboken, NJ: John Wiley & Sons, Inc., 2007.

[7] Y. Rahmat-Samii, "Genetic algorithm (GA) and particle swarm optimization (PSO) in engineering electromagnetics," in *17th International Conference on Applied Electromagnetics and Communications, 2003. ICECom 2003.*, Dubrovnik, Croatia, 2003, pp. 1–5.

[8] J. D. Townsend, "Improvement of ECM Techniques through Implementation of a Genetic Algorithm," M.S. thesis, Dept. Elect. and Comp. Eng., AFIT, Wright-Patterson AFB, OH, 2008.

[9] *Tactical Engagement Simulation Software (TESS™), Air RF v18.10.0, Multifunction Surveillance Radar and Surface-to-Air/Air-to-Air RF Guided Missile*. Ottawa, ON: Tactical Technologies Inc. - a Leonardo Company, 2018.

[10] *MATLAB Global Optimization Toolbox, Release 2018b*. Natick, MA: The MathWorks Inc., 2018.

[11] D. Adamy, *EW 101: A First Course in Electronic Warfare*. Norwood, MA: Artech House, Inc., 2001.

[12] A. Farina, "Electronic Counter-Countermeasures," in *Radar Handbook*, 3rd ed., M. I. Skolnik, Ed. New York, NY: McGraw-Hill, 2008, pp. 24.1-24.67.

[13] Avionics Department, *Electronic Warfare and Radar Systems Engineering Handbook*, 4th ed. Point Mugu, CA: Naval Air Warfare Center Weapons Division, 2013.

[14] G. W. Stimson, H. D. Griffiths, C. J. Baker, and D. Adamy, *Introduction to Airborne Radar*, 3rd ed. Edison, NJ: SciTech Publishing, 2014.

[15] M. McGrath and P. Marshall, "ECM techniques generator," in *48th Midwest Symposium on Circuits and Systems, 2005.*, Covington, KY, USA, 2005, pp. 1749-1752 Vol. 2.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, Australia, 1995, vol. 4, pp. 1942–1948 vol.4.

[17] R. Hassan, B. Cohanim, O. de Weck, and G. Venter, "A Comparison of Particle Swarm Optimization and the Genetic Algorithm," in *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Austin, TX, 2005, pp. 1–13.

[18] "Find minimum of function using genetic algorithm - MATLAB ga." [Online]. Available: https://www.mathworks.com/help/gads/ga.html. [Accessed: 29-Jan-2019].

[19] "Particle swarm optimization - MATLAB particleswarm." [Online]. Available: https://www.mathworks.com/help/gads/particleswarm.html. [Accessed: 29-Jan-2019].

[20] "Air RF v18.10.0, Multifunction Surveillance Radar and Surface-to-Air/Air-to-Air RF Guided Missile, Master Interface User's Guide." Tactical Technologies Inc. - a Leonardo Company, 19-Oct-2018.

[21] "SAM(CG)/AAA v18.10.0, Command Guided Surface-to-Air Missiles and Anti-Aircraft Artillery, User's Guide." Tactical Technologies Inc. - a Leonardo Company, 19-Oct-2018.

[22] S. K. Mishra, "Some new test functions for global optimization and performance of repulsive particle swarm method," *SSRN Electronic Journal*, Aug. 2006.

[23] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, Jan. 1960.

[24] "Rosenbrock function," *Wikipedia*. 18-Jun-2018.

[25] R. A. Rastrigin, *Systems of extremal control*. Nauka, Moscow, 1974.

[26] "Rastrigin function," *Wikipedia*. 05-Nov-2018.

[27] H. Mühlenbein, M. Schomisch, and J. Born, "The parallel genetic algorithm as function optimizer," *Parallel Computing*, vol. 17, no. 6, pp. 619–632, Sep. 1991.

[28] "Mersenne Twister," *Wikipedia*. 30-Jan-2019.

[29] *ECM Effectiveness Simulator, SAM(CG)AAA Command Guided Surface-to-Air Missiles and Anti-Aircraft Artillery*. Ottawa, ON: Tactical Technologies Inc. - a Leonardo Company, 2018.

[30] Legislative Services Branch, "Consolidated Federal Laws of Canada, Controlled Goods Regulations, SOR/2001-32." Minister of Justice, 22-Jun-2016.

[31] C. Kopp, "9K33 Osa/Romb Self Propelled Air Defence System / SA-8 Gecko," Air Power Australia, APA-TR-2009-0704, Jul. 2009.

[32] "9K33 Osa," *Wikipedia*. 05-Jun-2019.

[33] "SA-8 Gecko 9K33 OSA Ground-to-air missile system." [Online]. Available: https://www.armyrecognition.com/russia_russian_missile_system_vehicle_uk/sa-8_gecko_9k33_osa_ground-to-air_missile_system_technical_data_sheet_specifications_information_uk.html. [Accessed: 13-Jun-2019].

[34] "General Dynamics F-16 Fighting Falcon," *Wikipedia*. 22-Aug-2019.

[35] "Bell CH-146 Griffon," *Wikipedia*. 30-May-2019.

[36] "Electronic Warfare Forecast, ALQ-126B - Archived 04/2002," Forecast International, Jul. 2001.

[37] J. Loo and S. Labeaume, "CF-18 AN/ALQ-126B-MG 13 IIP Interface to the DREO Electronic Warfare Engagement Simulation Facility," Defence Research Establishment Ottawa, Ottawa, ON, Technical Note 93-4, Nov. 1992.

[38] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*. Redwood City, CA: Benjamin-Cummings Publishing Co., Inc., 1989.

[39] "Parallel Computing Toolbox Documentation." [Online]. Available: https://www.mathworks.com/help/parallel-computing/index.html. [Accessed: 14-May-2019].

[40] P. M. Fishbane, S. Gasiorowicz, and S. T. Thornton, *Physics For Scientists and Engineers*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1996.

[41] "Genetic Algorithm Options - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/gads/genetic-algorithm-options.html#f6633. [Accessed: 23-Aug-2019].

A MATLAB® Global Optimization Toolbox™ Algorithm Implementations

This appendix provides a summary of the GA and PSO algorithm implementations in MATLAB®, with a focus on the input variables, optimization options, and output arguments as they are used in this thesis.

A.1 Genetic Algorithm

The MATLAB® Global Optimization Toolbox™ [10] implements the GA as `ga.m`, which finds the local constrained or unconstrained minimum in a defined fitness (objective) function [18]. A $1 \times N$ vector argument is accepted as the input to the fitness function. The variable inputs for optimization are given in Table A.1 and are listed in the order in which they are included when calling the GA function. For this thesis, the fitness function performs both the processes of simulation of the TESS™ Simulink® model and scoring of the results to generate a fitness score for the candidate solution. This thesis did not use constraints, thus the input variables to the GA function were the fitness function, the number of variables, lower and upper bounds, and the *options*.

Table A.1: GA input variables [18]

Variable	Definition
fun	Fitness (objective) function to evaluate
nvars	Number of design variables in the fitness function to evaluate (dimension)
A	Matrix for inequality constraints ($A \cdot x \leq b$)
b	Vector for inequality constraints ($A \cdot x \leq b$)
Aeq	Matrix for equality constraints ($Aeq \cdot x = beq$)
beq	Vector for equality constraints ($Aeq \cdot x = beq$)
lb	Lower bounds on design variables
ub	Upper bounds on design variables
nonlcon	Nonlinear constraint function
Intcon	Integer constraints (Note: When there are integer constraints, `ga` does not accept linear or nonlinear equality constraints, only inequality constraints)
options	Options structure (set using `optimoptions`)

The *options* structure contains the user-customizable parameters that define the operation of the GA. If required, select options are modified in the main program script prior to calling the GA function. Table A.2 provides a summary of the options available for the GA, their default values, and the values for this thesis (where modified from the default values). The reasoning for each option value modification is described in Chapter 5.

Table A.2: GA optimization options [18]

Option	**Description**	**Default / Modified Value**
ConstraintTolerance	Determines the feasibility with respect to nonlinear constraints.	Positive scalar | {1e-3}
CreationFcn	Function that creates the initial population.	{'gacreationuniform'}
CrossoverFcn	Function that the algorithm uses to create crossover children.	{'crossoverscattered'}
CrossoverFraction	The fraction of the population at the next generation, not including elite children, which the crossover function creates.	Positive scalar | {0.8}
Display	Level of display returned to the command line.	{'final'}
EliteCount	Number of individuals in the current generation that are guaranteed to survive to the next generation.	Positive integer | {`ceil`(0.05* *PopulationSize*)}
FitnessLimit	If the fitness function attains the value of *FitnessLimit*, the algorithm halts.	Scalar | {-Inf} Modified: {0}
FitnessScalingFcn	Function that scales the values of the fitness function.	{'fitscalingrank'}
FunctionTolerance	The algorithm stops if the average relative change in the best fitness function value over *MaxStallGenerations* generations is less than or equal to *FunctionTolerance*.	Positive scalar | {1e-6} Modified: {1e-3}
HybridFcn	Function that continues the optimization after `ga` terminates.	{[]}

Option	Description	Default / Modified Value
InitialPopulationMatrix	Initial population used to seed the genetic algorithm. Has up to *PopulationSize* rows and *N* columns, where *N* is the number of variables.	Matrix | {[]}
InitialPopulationRange	Matrix or vector specifying the range of the individuals in the initial population. Applies to `gacreationuniform` creation function. `ga` shifts and scales the default initial range to match any finite bounds.	Matrix or vector | {[-10;10]} for unbounded components, {[-1e4 + 1; 1e4 + 1]} for unbounded components of integer-constrained problems, {[*lb*;*ub*]} for bounded components, with the default range modified to match one-sided bounds.
InitialScoresMatrix	Initial scores used to determine fitness. Has up to *PopulationSize* rows.	Column vector | {[]}
MaxGenerations	Maximum number of iterations before the algorithm halts.	Positive integer | {100**nvars*} Modified: {100}
MaxStallGenerations	The algorithm stops if the average relative change in the best fitness function value over *MaxStallGenerations* generations is less than or equal to *FunctionTolerance*.	Positive integer | {50} Modified: {20}
MaxStallTime	The algorithm stops if there is no improvement in the objective function for *MaxStallTime* seconds, as measured by `tic` and `toc`.	Positive scalar | {Inf}
MaxTime	The algorithm stops after running for *MaxTime* seconds, as measured by `tic` and `toc`. This limit is enforced after each iteration, so `ga` can exceed the limit when an iteration takes substantial time.	Positive scalar | {Inf}

Option	Description	Default / Modified Value
MutationFcn	Function that produces mutation children.	{'mutationgaussian'} Modified: {'mutationuniform', rate: 0.05}
NonlinearConstraintAlgorithm	Nonlinear constraint algorithm.	{'auglag'}
OutputFcn	Functions that `ga` calls at each iteration.	Function handle or cell array of function handles | {[]} Modified to use function handle of user-defined output function
PlotFcn	Function that plots data computed by the algorithm.	Function handle or cell array of function handles | {[]}
PopulationSize	Size of the population.	Positive integer | {50} when *nvars* <= 5, {200} otherwise Modified: {48}
PopulationType	Data type of the population.	{'doubleVector'}
SelectionFcn	Function that selects parents of crossover and mutation children.	{'selectionstochunif'}
UseParallel	Compute fitness and nonlinear constraint functions in parallel when `true`.	Boolean | {false} Modified: {true}
UseVectorized	Compute functions in vectorized fashion when `true`.	Boolean | {false}

The output arguments generated by the GA, all of which were used for this thesis to record algorithm performance and results, are given in Table A.3 and are listed in the order in which they can be requested.

Table A.3: GA output arguments [18]

Variable	Definition
x	Solution, returned as a real vector. *x* is the best point that `ga` located during its iterations.
fval	Objective function value at the solution, returned as a real number.
exitflag	Reason that `ga` stopped, returned as an integer.
output	Information about the optimization process, returned as a structure.
population	Final population, returned as a $PopulationSize \times nvars$ matrix. The rows of population are the individuals.
scores	Final scores, returned as a column vector.

The first two and last two output arguments provide the results of the optimization problem; the best solution and score values were printed to the command window and stored for later analysis. The population and scores were saved using a custom output function at the end of each generation. The *exitflag* variable provides a scalar integer indicating the reason for algorithm termination. The *exitflag* values and their descriptions are summarized in Table A.4.

Table A.4: GA *exitflag* descriptions [18]

Exit Flag	**Meaning**
1	Average cumulative change in value of the fitness function over *MaxStallGenerations* generations is less than *FunctionTolerance*, and the constraint violation is less than *ConstraintTolerance*.
3	Value of the fitness function did not change in *MaxStallGenerations* generations and the constraint violation is less than *ConstraintTolerance*.
4	Magnitude of step smaller than machine precision and the constraint violation is less than *ConstraintTolerance*.
5	Minimum fitness limit *FitnessLimit* reached and the constraint violation is less than *ConstraintTolerance*.
0	Maximum number of generations *MaxGenerations* exceeded.
–1	Optimization terminated by an output function or plot function.
–2	No feasible point found.
–4	Stall time limit *MaxStallTime* exceeded.
–5	Time limit *MaxTime* exceeded.

For this thesis, the best fitness score attainable by a candidate solution was 0 (i.e. the *FitnessLimit*). Thus, the ideal reason for algorithm termination was if the minimum fitness limit was reached, indicated by exit flag 5 for the GA. Only exit flags 1 and 5 were encountered during this thesis, indicating that the algorithm either stalled at a local minimum (exit flag 1) or converged to the global minimum (exit flag 5).

The *output* variable provides performance specifications as a result of each execution of the algorithm. The parameters from the *output* structure are described in Table A.5. The *rngstate* variable permits re-creation of the results by using the same seed variables when executing the optimization algorithm. This was useful when validating the parallelized implementation with the serial program to ensure identical results were generated. The number of generations and function evaluations was useful when comparing the GA to the PSO in terms of optimization efficiency and execution time.

Table A.5: GA output structure descriptions [18]

Variable	**Definition**
problemtype	Problem type, one of: *unconstrained*, *boundconstraints*, *linearconstraints*, *nonlinearconstr*, or *integerconstraints*.
rngstate	State of the random number generator, just before the algorithm started.
generations	Number of generations computed.
funccount	Number of evaluations of the fitness function.
message	Reason the algorithm terminated.
maxconstraint	Maximum constraint violation, if any.

A.2 Particle Swarm Optimization

The MATLAB® Global Optimization Toolbox™ [10] implements the PSO as `particleswarm.m`, which finds the local unconstrained minimum in a defined fitness (objective) function [19]. A $1 \times N$ vector argument is accepted as the input to the fitness function. The variable inputs for optimization are given in Table A.6 and are listed in the order in which they are included when calling the PSO function. Unlike the GA implementation, the PSO cannot accept constraints as variable inputs. The fitness function, number of variables, and bounds were identical for both the GA and PSO.

Table A.6: PSO input variables [19]

Variable	Definition
fun	Fitness (objective) function to evaluate
nvars	Number of design variables in the fitness function to evaluate (dimension)
lb	Lower bounds on design variables
ub	Upper bounds on design variables
options	Options structure (set using `optimoptions`)

As with the GA, the PSO *options* structure contains user-customizable parameters that define the algorithm's operation. If required, select options are modified in the main program script prior to calling the PSO function. Table A.7 provides a summary of the options available for the PSO, their default values, and the values for this thesis (where modified from the default values). The reasoning for each option value modification is described in Chapter 5.

Table A.7: PSO optimization options [19]

Option	**Description**	**Default / Modified Value**
CreationFcn	Function that creates the initial swarm.	{'pswcreationuniform'}
Display	Level of display returned to the command line.	{'final'}
FunctionTolerance	Iterations end when the relative change in best fitness function value over the last *MaxStallIterations* iterations is less than *FunctionTolerance*.	Positive scalar | {1e-6} Modified: {1e-3}
HybridFcn	Function that continues the optimization after `particleswarm` terminates.	{[]}
InertiaRange	Two-element real vector with same sign values in increasing order. Gives the lower and upper bound of the adaptive inertia. To obtain a constant (non-adaptive) inertia, set both elements of *InertiaRange* to the same value.	{[0.1,1.1]}
InitialSwarmMatrix	Initial population or partial population of particles. *M*-by-*nvars* matrix, where each row represents one particle. If *M* < *SwarmSize*, then `particleswarm` creates more particles so that the total number is *SwarmSize*. If *M* > *SwarmSize*, then `particleswarm` uses the first *SwarmSize* rows.	Matrix | {[]}
InitialSwarmSpan	Initial range of particle positions that `pswcreationuniform` creates. Can be a positive scalar or a vector with *nvars* elements. The range for any particle component is -*InitialSwarmSpan*/2, *InitialSwarmSpan*/2, shifted and scaled if necessary to match any bounds. *InitialSwarmSpan* also affects the range of initial particle velocities.	Positive scalar or vector | {2000}

Option	**Description**	**Default / Modified Value**
MaxIterations	Maximum number of iterations before the algorithm halts.	Positive integer | {200*nvars} Modified: {100}
MaxStallIterations	The algorithm stops if the average relative change in the best fitness function value over *MaxStallIterations* iterations is less than or equal to *FunctionTolerance*.	Positive integer | {20}
MaxStallTime	The algorithm stops if there is no improvement in the objective function for *MaxStallTime* seconds, as measured by `tic` and `toc`.	Positive scalar | {Inf}
MaxTime	The algorithm stops after running for *MaxTime* seconds, as measured by `tic` and `toc`.	Positive scalar | {Inf}
MinNeighborsFraction	Minimum adaptive neighborhood size, a scalar from 0 to 1.	Positive scalar | {0.25}
ObjectiveLimit	If the fitness function attains the value of *ObjectiveLimit*, the algorithm halts.	Scalar | {-Inf} Modified: {0}
OutputFcn	Function handle or cell array of function handles. Output functions can read iterative data, and stop the solver.	{[]} Modified to use function handle of user-defined output function
PlotFcn	Function name, function handle, or cell array of function handles. Plot functions can read iterative data, plot each iteration, and stop the solver.	{[]}
SelfAdjustmentWeight	Weighting of each particle's best position when adjusting velocity.	Finite scalar | {1.49}
SocialAdjustmentWeight	Weighting of the neighborhood's best position when adjusting velocity.	Finite scalar | {1.49}
SwarmSize	Number of particles in the swarm.	Positive integer greater than 1 | {min(100,10**nvars*)} Modified: {48}

Option	Description	Default / Modified Value
UseParallel	Compute fitness function in parallel when `true`.	Boolean | {false} Modified: {true}
UseVectorized	Compute fitness function in vectorized fashion when `true`.	Boolean | {false}

The output arguments generated by the PSO are given in Table A.8 and are listed in the order in which they can be requested.

Table A.8: PSO output arguments [19]

Variable	Definition
x	Solution, returned as a real vector that minimizes the objective function subject to any bound constraints.
fval	Objective value, returned as the real scalar *fun(x)*.
exitflag	Algorithm stopping condition, returned as an integer identifying the reason the algorithm stopped.
output	Solution process summary, returned as a structure containing information about the optimization process.

Unlike the GA, which returns the values of the entire population as well as each member's fitness, the PSO only returns the position and fitness of the global best particle in the swarm at function exit. A custom output function was used to save all the swarm positions and fitness scores at the end of each iteration. Similar to the GA, an *exitflag* variable is provided, as described in Table A.9.

Table A.9: PSO *exitflag* descriptions [19]

Exit Flag	**Meaning**
1	Relative change in the objective value over the last *MaxStallIterations* iterations is less than *FunctionTolerance*.
0	Number of iterations exceeded *MaxIterations*.
–1	Iterations stopped by output function or plot function.
–2	Bounds are inconsistent: for some i, $lb(i) > ub(i)$.
–3	Best objective function value is at or below *ObjectiveLimit*.
–4	Best objective function value did not change within *MaxStallTime* seconds.
–5	Run time exceeded MaxTime seconds.

As with the GA, the best fitness score attainable by a candidate solution was 0 (i.e. the *ObjectiveLimit*). Thus, the ideal reason for algorithm termination was if the minimum fitness limit was reached, indicated by exit flag –3 for the PSO. Only exit flags 1 and –3 were encountered during this thesis, indicating that the algorithm either stalled at a local minimum (exit flag 1) or converged to the global minimum (exit flag –3).

Finally, the *output* variable is included to provide information about the optimization process. The parameters from the *output* structure are described in Table A.10. As with the GA, the *rngstate* variable permits re-creation of the results by using the same seed variables when executing the optimization algorithm. This was useful when validating the parallelized implementation with the serial program to ensure identical results were generated. The number of iterations and function evaluations was useful when comparing the PSO to the GA in terms of optimization efficiency and execution time.

Table A.10: PSO output structure descriptions [19]

Variable	**Definition**
iterations	Number of solver iterations.
funccount	Number of objective function evaluations.
message	Reason the algorithm stopped.
rngstate	State of the default random number generator just before the algorithm started.

B ECM Technique Generation Results

This appendix presents the results of each optimization round for select engagement scenarios conducted during this thesis, in table form. Each engagement scenario is briefly described and variables unique to the scenario are stated. The results from each optimization round are provided in separate tables for the GA and PSO. An analysis of the simulation results is provided in Chapter 5.

B.1 Fighter vs. Non-Coherent TTR

The fighter airborne target was flown at an approach angle of 0 degrees against the threat system with a non-coherent TTR. The optimization bounds were those listed in Table 5.1. The results of the optimization are in Table B.1 and Table B.2 for the GA and PSO, respectively.

Table B.1: GA optimization results – fighter vs. non-coherent TTR, 0 degrees

Round	1	2	3	4	5	6	7	8	9	10
Exit Flag[1]	1	1	1	1	1	1	1	1	1	1
Time [min]	299.16	270.17	270.90	270.90	270.88	270.33	270.69	269.76	270.31	270.95
Generations	21	21	21	21	21	21	21	21	21	21
Mean Time Per Generation [min]	13.60	12.28	12.31	12.31	12.31	12.29	12.30	12.26	12.29	12.32
Function Evaluations	1056	1056	1056	1056	1056	1056	1056	1056	1056	1056
Fitness	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
R_0 **[µs]**	1.80	-4.42	-0.41	2.54	-2.72	-3.60	0.35	-0.45	3.87	1.98
R_{max} **[µs]**	-14.64	-14.48	-14.47	-14.68	-14.86	-14.86	-14.79	-13.84	-13.39	-14.95
T_i **[s]**	0.42	3.09	2.30	0.48	2.85	2.15	0.43	1.41	1.69	0.12
T_f **[s]**	4.51	0.18	0.65	4.82	3.91	1.13	2.82	3.41	2.55	3.14
v **[m/s]**	405	380	340	425	540	380	445	275	445	595
a **[m/s^2]**	55	30	50	35	35	35	50	40	45	30
Frequency Coord	1	0	0	0	0	1	0	0	0	1
Power Reduction [–dB]	2.21	2.01	2.04	1.90	2.34	1.53	2.90	0.37	0.18	2.79
PW **[µs]**	1.50	1.90	1.10	0.60	1.20	1.40	1.10	1.40	0.60	2.00

Note: 1. GA exit flag definitions may be found in Appendix A, Table A.4.

Table B.2: PSO optimization results – fighter vs. non-coherent TTR, 0 degrees

Round	1	2	3	4	5	6	7	8	9	10
Exit Flag[1]	1	1	1	1	1	1	1	1	1	1
Time [min]	270.09	269.82	269.91	270.23	270.09	270.34	269.98	269.51	270.38	270.41
Iterations	21	21	21	21	21	21	21	21	21	21
Mean Time Per Iteration [min]	12.28	12.26	12.27	12.28	12.28	12.29	12.27	12.25	12.29	12.29
Function Evaluations	1056	1056	1056	1056	1056	1056	1056	1056	1056	1056
Fitness	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
R_0 **[µs]**	-5.00	3.12	-5.00	2.73	-5.00	-5.00	0.47	-1.73	-4.86	4.85
R_{max} **[µs]**	-15.00	-14.40	-13.42	-15.00	-15.00	-15.00	-15.00	-14.88	-15.00	-15.00
T_i **[s]**	2.75	0.14	1.30	1.67	1.38	1.85	1.49	4.73	0.00	1.47
T_f **[s]**	4.56	4.09	2.68	4.90	4.37	5.00	5.00	4.93	3.32	0.00
v **[m/s]**	600	270	465	600	345	600	360	600	365	365
a **[m/s^2]**	50	50	35	40	25	25	40	60	30	60
Frequency Coord	1	0	1	0	1	1	0	1	1	1
Power Reduction [–dB]	0.34	0.97	0.00	0.00	3.00	2.32	0.00	2.70	1.49	2.47
PW **[µs]**	1.40	0.80	0.80	1.20	1.00	0.80	1.20	0.50	0.70	1.00

Note: 1. PSO exit flag definitions may be found in Appendix A, Table A.9.

B.2 Fighter vs. Coherent TTR

The fighter airborne target was flown at an approach angle of 45 degrees against the threat system with a coherent TTR. The optimization bounds were those listed in Table 5.1. The PW was held constant at 0.5 μs. Five additional optimization rounds of the GA were completed using the mutation function *mutationuniform* with a mutation rate of 0.05. The results of the optimization are in Table B.3 and Table B.4 for the GA, and Table B.5 for the PSO, respectively.

Table B.3: GA optimization results – fighter vs. coherent TTR, 45 degrees

Round	1	2	3	4	5	6	7	8	9	10
Exit Flag	1	1	1	1	1	1	1	1	1	1
Time [min]	148.99	151.03	149.16	152.51	149.12	153.65	154.61	150.95	150.78	151.05
Generations	51	51	51	51	51	51	51	51	51	51
Mean Time Per Generation [min]	2.87	2.90	2.87	2.93	2.87	2.95	2.97	2.90	2.90	2.90
Function Evaluations	2496	2496	2496	2496	2496	2496	2496	2496	2496	2496
Fitness	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
R_0 **[μs]**	-3.42	4.77	-2.25	-1.97	2.31	-0.82	-3.59	-3.6	0.95	2.71
R_{max} **[μs]**	-4.79	-9.09	-9.81	-14.73	-13.21	7.23	-14.65	-6.11	-9.63	-14.55
T_i **[s]**	0.83	1.42	0.85	2.34	3.35	1.37	0.49	0.49	1.1	0.61
T_f **[s]**	0.25	0.13	0.41	4.8	0.94	4.83	1.09	2.72	1.31	4.29
v **[m/s]**	525	500	450	500	385	235	245	410	505	335
a **[m/s^2]**	45	40	50	15	45	50	25	40	50	60
Frequency Coord	1	1	1	1	1	1	1	1	1	1
Power Reduction [–dB]	1.02	2.76	1.83	2.55	0.86	1.39	1.75	1.07	1.06	1.58

Table B.4: GA optimization results, *mutationuniform* with mutation rate 0.05

Round	1	2	3	4	5
Exit Flag	1	5	1	1	1
Time [min]	149.40	19.07	151.91	155.33	148.86
Generations	51	4	51	51	51
Mean Time Per Generation [min]	2.87	3.81	2.92	2.99	2.86
Function Evaluations	2496	240	2496	2496	2496
Fitness	0.1	0	0.1	0.1	0.1
$\boldsymbol{R_0}$ **[µs]**	-0.63	2.44	0.23	3.33	1.8
$\boldsymbol{R_{max}}$ **[µs]**	-12.74	4.03	-10.05	-12.83	-7.38
$\boldsymbol{T_i}$ **[s]**	2.89	0.4	3.13	2.94	4.22
$\boldsymbol{T_f}$ **[s]**	1.35	0.09	3.75	4.96	4.06
$\boldsymbol{v}$ **[m/s]**	460	305	535	390	515
$\boldsymbol{a}$ **[m/s^2]**	15	60	55	60	25
Frequency Coord	1	1	1	0	1
Power Reduction [–dB]	1.66	0.79	0.97	1.74	1.72

Table B.5: PSO optimization results – fighter vs. coherent TTR, 45 degrees

Round	1	2	3	4	5	6	7	8	9	10
Exit Flag	1	1	-3	1	-3	-3	1	1	-3	-3
Time [min]	67.44	66.33	12.07	67.18	11.25	25.35	67.19	67.21	31.13	37.18
Iterations	21	21	1	21	1	6	21	21	8	10
Mean Time Per Iteration [min]	3.07	3.02	6.04	3.05	5.62	3.62	3.05	3.06	3.46	3.38
Function Evaluations	1056	1056	96	1056	96	336	1056	1056	432	528
Fitness	0.1	0.1	0	0.1	0	0	0.1	0.1	0	0
R_0 **[µs]**	3.83	-5	5	-3.82	3.79	5	-5	-2.08	3.02	2.57
R_{max} **[µs]**	-12.18	-15	7.06	-9.74	4.87	5.43	-9.82	-15	3.98	1.88
T_i **[s]**	0	4.62	0	4.58	0	0	4.92	4.59	0	0
T_f **[s]**	0	0	2.33	0	0.74	3.84	2.3	5	0.2	5
v **[m/s]**	200	480	200	230	200	200	280	200	325	200
a **[m/s^2]**	60	20	60	40	60	60	60	55	60	60
Frequency Coord	1	1	1	1	1	1	1	0	1	1
Power Reduction [–dB]	2.05	3	2.52	1.63	0	0	2.41	1.38	0	3

B.3 Rotary-Wing vs. Coherent TTR

The rotary-wing airborne target was flown at an approach angle of 90 degrees against the threat system with a coherent TTR. The optimization bounds were those listed in Table 5.1, with fighter velocity and acceleration bounds. The results of the optimization are in Table B.6 and Table B.7 for the GA and PSO, respectively.

Table B.6: GA optimization results – rotary-wing vs. coherent TTR, 90 degrees

Round	1	2	3	4	5	6	7	8	9	10
Exit Flag	1	1	1	1	1	1	1	1	1	1
Time [min]	364.96	321.21	321.55	378.98	320.43	321.05	321.57	364.34	321.34	321.03
Generations	24	21	21	25	21	21	21	24	21	21
Mean Time Per Generation [min]	14.60	14.60	14.62	14.58	14.57	14.59	14.62	14.57	14.61	14.59
Function Evaluations	1200	1056	1056	1248	1056	1056	1056	1200	1056	1056
Fitness	0.3399	0.3268	0.3234	0.3032	0.3308	0.3258	0.3272	0.3206	0.3321	0.3268
R_0 **[µs]**	-3.58	-0.83	-0.58	-1.12	-1.84	-0.09	-1.00	-1.08	-0.85	-2.75
R_{max} **[µs]**	11.73	0.85	-2.05	10.28	12.89	-1.51	5.54	0.76	-8.40	0.20
T_i **[s]**	0.25	0.09	4.58	1.37	1.80	0.08	2.76	4.78	0.20	1.11
T_f **[s]**	0.86	1.06	2.86	1.75	0.69	1.31	4.08	1.81	0.00	1.04
v **[m/s]**	575	505	545	355	560	200	220	240	500	295
a **[m/s^2]**	35	10	50	25	40	40	40	20	10	15
Frequency Coord	0	0	0	1	0	0	1	1	0	0
Power Reduction [–dB]	0.81	0.23	0.10	0.10	0.04	0.00	0.45	2.63	0.25	0.13
PW **[µs]**	2.00	1.60	1.30	1.40	1.90	1.20	1.30	1.50	1.20	1.60

Table B.7: PSO optimization results – rotary-wing vs. coherent TTR, 90 degrees

Round	1	2	3	4	5	6	7	8	9	10
Exit Flag	1	1	1	1	1	1	1	1	1	1
Time [min]	851.26	391.13	1182.6	636.51	1007.8	679.06	782.44	1040.5	666.51	1001.4
Iterations	58	26	81	43	61	46	53	71	45	64
Mean Time Per Iteration [min]	14.43	14.49	14.42	14.47	16.25	14.45	14.49	14.45	14.49	15.41
Function Evaluations	2832	1296	3936	2112	2976	2256	2592	3456	2208	3120
Fitness	0.3228	0.3336	0.323	0.325	0.3171	0.327	0.2788	0.2901	0.3278	0.3278
R_0 **[µs]**	4.66	-1.92	-1.21	-0.87	3.74	3.05	-1.45	-1.40	-1.00	-0.78
R_{max} **[µs]**	-9.48	-1.26	-3.31	-2.66	-7.15	-2.64	5.06	14.73	-3.66	-6.12
T_i **[s]**	0.91	2.49	0.38	0.61	0.14	1.90	0.88	0.14	4.23	2.85
T_f **[s]**	1.08	3.04	1.83	0.54	4.06	0.83	0.00	3.37	3.60	2.01
v **[m/s]**	460	600	385	395	410	405	455	525	410	255
a **[m/s^2]**	60	10	60	40	60	40	10	30	55	35
Frequency Coord	0	0	0	0	0	0	1	1	0	0
Power Reduction [–dB]	0.66	0.71	0.05	0.14	0.05	0.41	1.24	0.01	0.35	0.63
PW **[µs]**	1.80	1.90	1.90	1.70	1.40	1.40	1.60	2.00	1.70	2.00