An investigation of multi-dimensional evolutionary algorithms for virtual reality scenario development

Scott Papson
Rowan University

An investigation of multi-dimensional evolutionary algorithms for virtual reality scenario development

by

Scott Papson

A thesis submitted to the

graduate faculty in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Engineering (Electrical Engineering)

Approved:	Members of the Committee:
In Charge of Major Work	
For the Major Department	
For the College	

Rowan University
Glassboro, NJ
2004

ABSTRACT

Virtual reality (VR) has emerged as a powerful visualization tool for design, simulation, and analysis in modern complex industrial systems. The primary motivation for this thesis is to develop a framework for the effective use of VR in design-simulation-analysis cycles, particularly in situations involving large, complex, multi-dimensional data-sets. This thesis develops a framework that is intended to support not only the integration of such data for visual, interactive, and immersive displays, but also provides a method for performing risk analysis. Previously "static" VR environments are enhanced with time-evolutionary capabilities. Four candidate algorithms are evaluated for this purpose – deterministic modeling, auto-regressive moving average modeling, genetic algorithm modeling, and hidden Markov modeling. Benefits, drawbacks, and trade-offs are evaluated with reference to their suitability for development in a VR environment. The methods developed in this research work are demonstrated by applying them to multi-sensor data obtained during the in-line, nondestructive evaluation of gas transmission pipelines.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

ACKNOWLEDGMENTS

This work is funded in part by:

National Science Foundation
Major Research Instrumentation (MRI) Program
Award #0216348

&

National Energy Technology Laboratory (NETL)
United States Department of Energy,
Grant DE-FC26-02NT41648

CHAPTER 1 – INTRODUCTION

1.1 Advanced Scientific Visualization & Virtual Reality

1.1.1 Definitions & Explanation

Advanced scientific visualization (ASV) is the use of visual techniques to aid in problem solving and data analysis. Generally speaking, ASV can be applied to a range of fields and across broad scopes of problems [1]. Although ASV is useful for many types of problems, it is usually employed for handling problems with large or complex data sets, because multi-dimensional problems are more easily understood when the data is visualized [2]. Meteorologists use ASV to study weather phenomena, archeologists use ASV to study excavations, political analysts use ASV to study political trends, mathematicians use ASV to study complex equations, astrophysicists use ASV to study the cosmos, and so on [3].

Just as there are numerous applications of ASV, there are numerous methods of employing ASV. A simple x-y scatter plot of data is one type of scientific visualization. For different problems, different visualization techniques will be more effective then others. However, some techniques will provide certain advantages across large scopes of problems. One such ASV technique that offers powerful benefits is virtual reality (VR) [2]. Figure 1.1 below illustrates various virtual worlds.

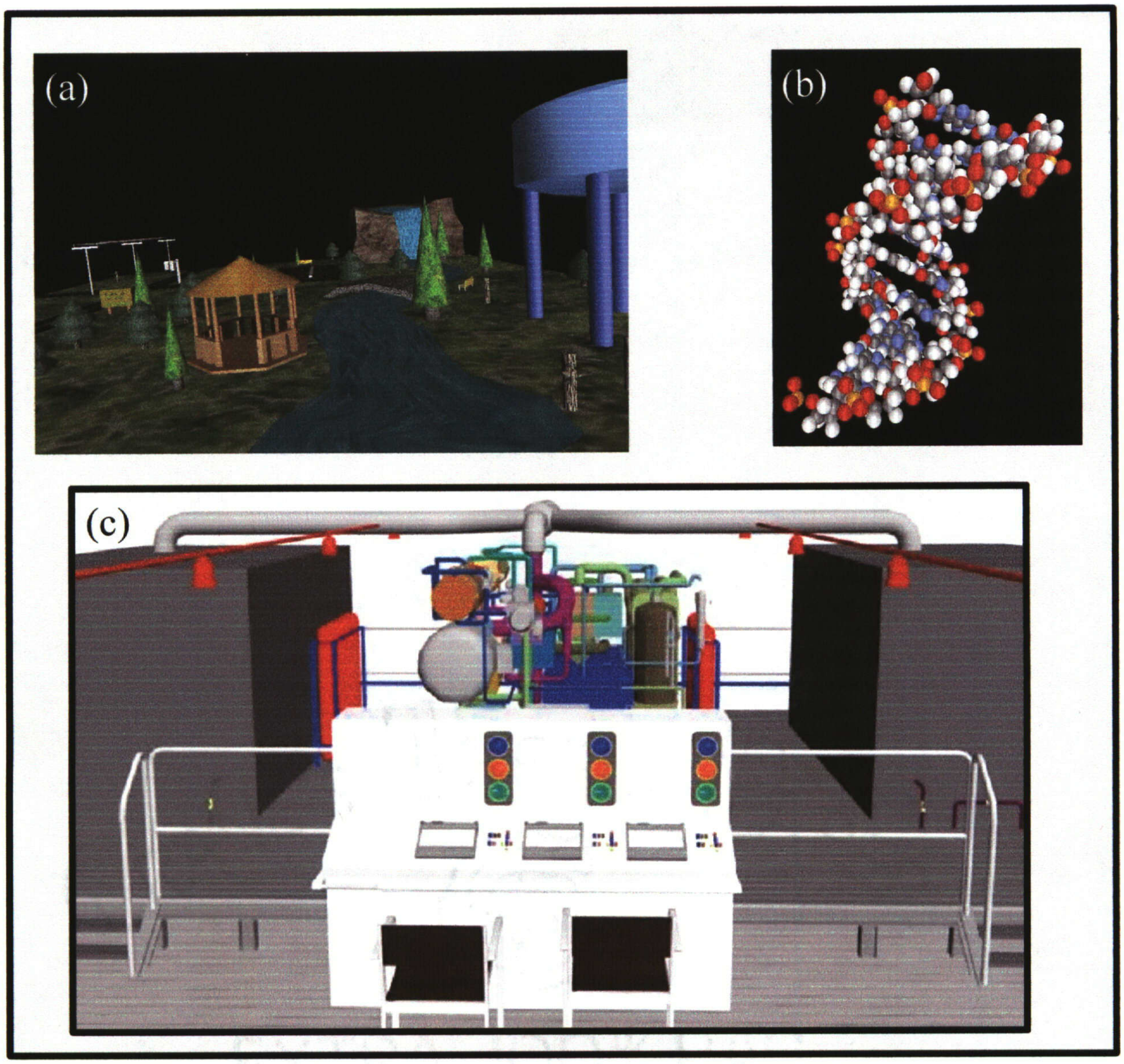


Figure 1.1 – Example VR worlds.

World (a) illustrates how a civil engineer might visualize a park to address certain environmental concerns. The visualization in (b) could be used by a geneticist to aid in the visualization of DNA. Finally, world (c) illustrates the use of VR for ship design by allowing the user to arrange components within the engine room of a battleship.

VR is a system that "gives the user the psychophysical experience of being surrounded by a virtual, that is, computer generated, environment [2]." Traditionally, there are three components to VR – immersion, navigation, and interaction [4]. It is the combination of these three components that creates a virtual world and provides the user the benefits from this type of visual analysis. This concept is illustrated below in Figure 1.2.

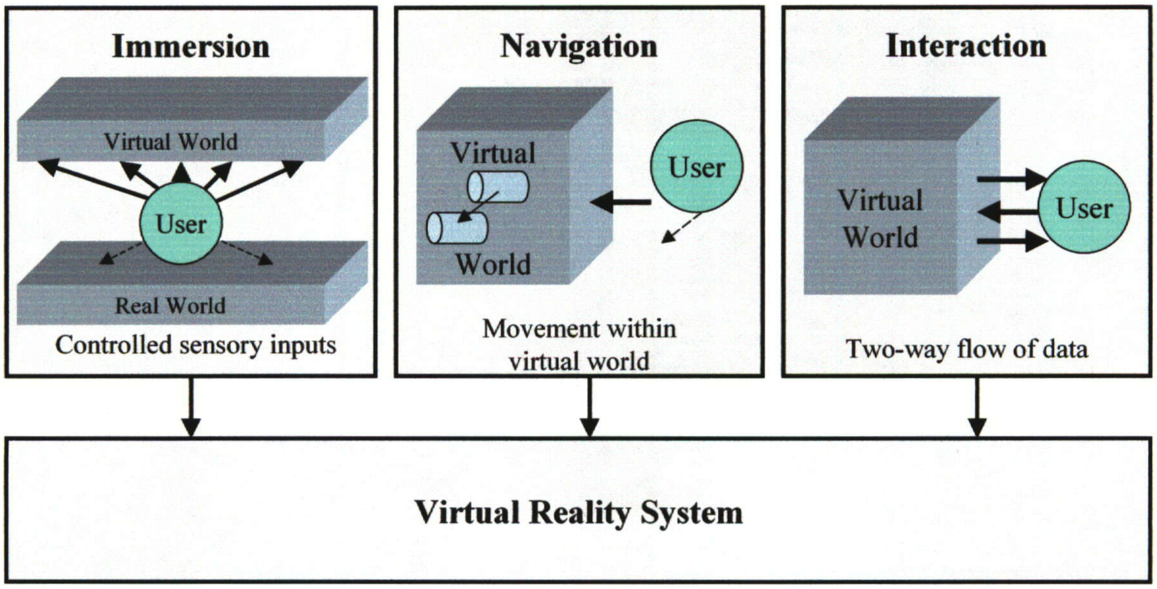


Figure 1.2 – Components of VR.

Immersion refers to the blocking of real world sensory inputs from interfering with the virtual world. A sense of immersion is usually created through the ability of the visual display system to minimize contradictions between the real world and the virtual world [5]. Navigation is the ability of the user to move within the virtual world; navigation includes manipulation of the orientation of the data being represented. Finally, interaction is the tracking of the user inside the virtual world, and the ability of the world to react to movements and commands initiated by the user [6].

Additionally, there are three other components that can be used to describe a VR world – visual display system, world components, and world data. The visual display system is responsible for handling graphical rendering. The type of display system will have a direct impact on the immersion, navigation, and interaction experienced in the virtual world. Both the world components and the world data refer to the content being displayed within the virtual world.

Since data analysis is the primary focus when VR is being used for the purpose of ASV, the conclusions drawn from the virtual world are a function of the data being displayed and the manner in which it is displayed. The sense of immersion, the components, and data being visualized combine to create presence inside the virtual world. Presence is the sense of actually being in the virtual world [1]. With greater presence, a more realistic and more useful analysis can take place. "...virtual reality works (induces presence) because it triggers exactly the same perceptive mechanisms as reality [7]."

1.1.2 Brief History

In today's world it is easy to see ASV being used to solve many different types of problems. With computer tools so accessible, it is not difficult to use these resources often. These tools, however, were not always available, but still ASV was a powerful technique for solving problems. The roots of modern day visualization techniques date back to 1854 when Dr. Snow used cartography to help quell an outbreak of cholera [8]. Faraday and Maxwell envisioned lines of magnetic flux emanating from charged bodies [9]; it was these visions that led to the development of Maxwell's equations. Kekule, a French chemist, once had a 'waking' dream as he was walking onto a bus; he saw the manner in which atoms grouped themselves in space [10]. His ideas helped lead to the development of the base structure of carbon. These types of inspiring visualizations are quite common across many disciplines.

Specifically, multi-dimensional imaging has strong roots in many applications of medicine, engineering, and physics. Medical imaging techniques range from x-rays to magnetic resonance images (MRI) and computerized axial tomography (CAT) scans. Complex algorithms were developed and are continually being developed to satisfy the high demands of these and other medical imaging systems. The visualization of nano-scale electronics has opened new paths for the development of the transistor. Finally, visualization systems such as the Hubble telescope have allowed astronomers to visually explore the depths of space.

1.1.3 VR Hardware

In today's technological age, visual display systems are advancing rapidly and allowing visions to be shared with the masses. Along with visual systems, three-dimensional object and image capturing capabilities are also evolving. Natural phenomena are being digitally reproduced in true three-dimensions. Continual advances are being made based off of the needs of industrial leaders and the visions of countless researchers. One such piece of technology is VR hardware. The VR hardware system handles the graphical rendering and display of the virtual world components. Many different types of display systems are available on market, but they can be broken down in three main groups – fully-immersive, semi-immersive, and non-immersive.

Fully-immersive displays are the most powerful of the three types of systems. A fully-immersive display requires that the user be fully immersed inside the virtual world; all contradictory real world sensations are blocked. There are two main types of fully-immersive systems that are widely used – head-mount displays (HMDs) and the CAVE®. HMDs are useful in situations where there is a limited amount of physical space that can be devoted to the hardware system. With a HMD, each eye is shown a unique picture. The picture shown to each eye corresponds to the stereo view for that eye based on the 3-D world being displayed. Dr.

Carolina Cruz-Neira at the Electronic Visualization Laboratory of the University of Illinois was integral in developing the second commonly used fully-immersive display system, the CAVE® [8]. The CAVE® is a fully-immersive display system in which the world is projected onto the four walls, ceiling, and even the floor.

Liquid crystal display (LCD) shutter glasses, similar to those shown in Figure 1.3, are worn. The shutter glasses are synchronized with the display system and project only one eye's perspective at a time. Switching between the two views allows the depth information to be conveyed to the user.

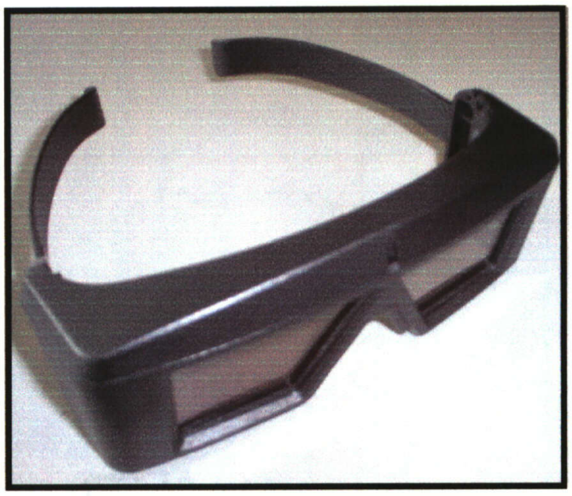


Figure 1.3 – LCD shutter glasses.

Semi-immersive displays allow some of the contradictory real world senses to be blocked. Typically, semi-immersive displays will consist of a single screen in front of the user. The VR system illustrated in Figure 1.4 is a semi-immersive unit. Again, the depth information can be conveyed through the use of an LCD synchronized stereoscopic display. Non-immersive displays are the least effective of the three types of systems. A non-immersive display gives the user little if any feeling of truly being within the virtual environment. A simple computer monitor is an example of a non-immersive display system.

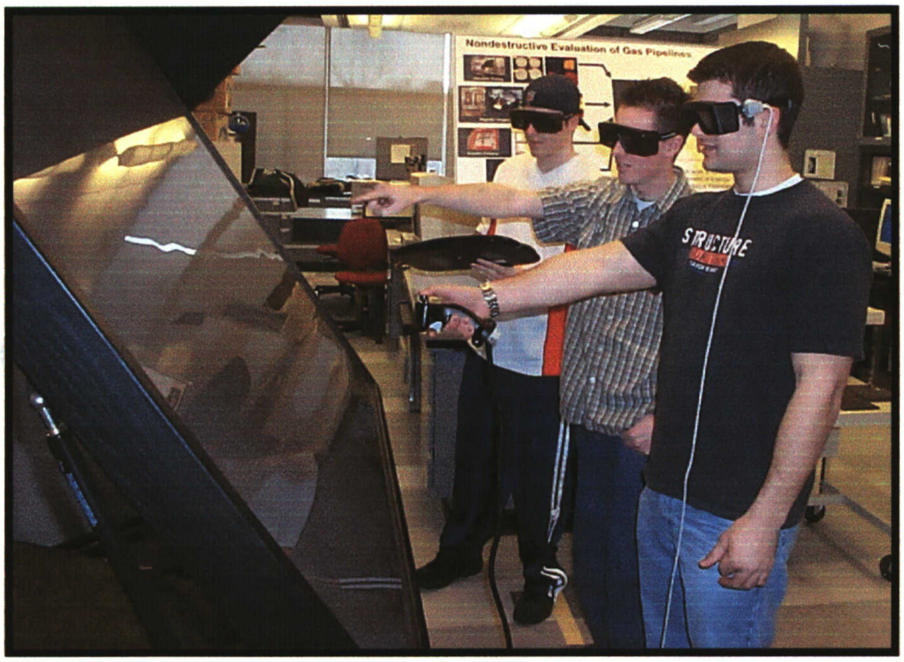


Figure 1.4 – A semi-immersive VR system.

The two additional requirements on VR are navigation and interaction. These are typically done with the use of tracking sensors. The sensors are responsible for keeping track of the user's movements. Additional peripheral devices can be used to aid in navigation. A six-degree-of-freedom (6-DOF) device would allow the user to control translation in x, y, and z, as well as rotation about the x, y, and z-axes. Haptics devices which allow for the input and output of tactile data are also used. A glove that allows users to grasp virtual objects and feel textures would be an example of a haptic interface device [15].

1.2 Nondestructive Evaluation

1.2.1 Need for Inspection

Since both people and machines are fallible, it is essential that the work done by each be evaluated. When dealing with physical inspections, two categories can be broadly defined – destructive evaluation and nondestructive evaluation (NDE). Being able to evaluate the condition of an object or process without harming or disrupting the condition is ideal; NDE tries to accomplish this. NDE is used heavily in industries where regular maintenance and testing is required. Aircraft are often subjected to non-destructive testing to ensure their integrity. Similarly underground pipelines, bridges, and medical applications require inspections where the object being investigated is not rendered unfit for service. There are three axioms for the relationship between satisfactory service and NDE – all materials contain flaws; flaws in a material do not necessarily render it unfit for service; the detectability of a flaw generally increases with size [16].

The first axiom stresses the need for reliable inspection methods. All materials, no matter how well created, will contain some number of flaws. But all flaws are not serious. As described by the second axiom, a flaw in a material does not imply that the system will be unfit for service. This axiom stresses that in the presence of an anomaly, risk analysis techniques need to be employed to judge the severity and potential risks involved for the system.

1.2.2 Inspection Methods

NDE inspection techniques generally can be described as passive or active. A passive technique is one that can be described as monitoring. A passive system will acquire a certain signal and analyze that given signal. In the presence of a defect or a fault, the signal will be altered. The signal will reflect the change, and the fault will be discovered. Visual inspections and leak testing would be considered passive techniques [16]. An acoustic emission analysis would also be categorized as an example of a passive technique [17].

Active testing techniques involve an input of energy into a specimen during testing. In active testing, a given form of energy is input into a specimen, and the specimen's response is monitored. An analysis of the output signal will aid in determining fault characteristics. The process of active NDE testing is illustrated in Figure 1.5.

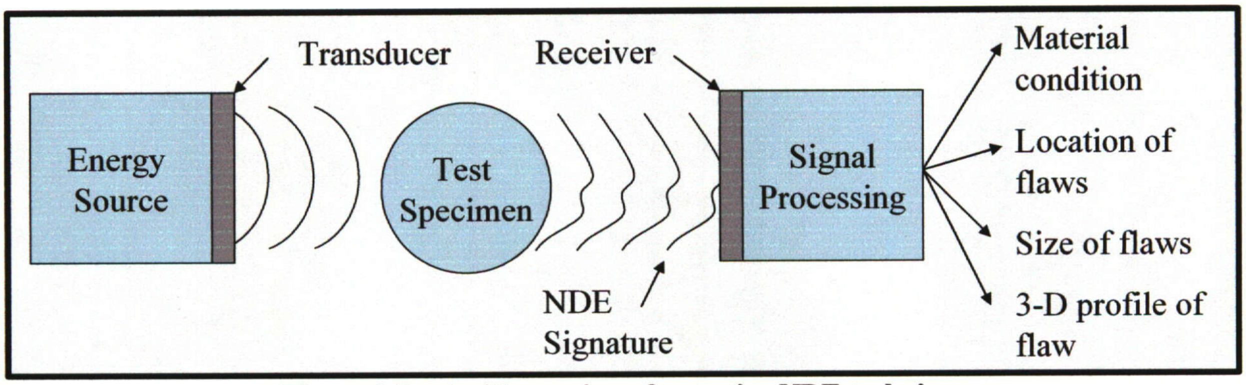


Figure 1.5 – An illustration of an active NDE technique.

Signal processing techniques are employed to characterize and draw conclusions from the received NDE signature. The signal processing techniques can aid in determining the overall condition of the test specimen. Similarly, certain techniques determine the presence of a flaw. The flaw can subsequently be characterized by its location, size, or severity. Ideally, the combination of various pieces of information will yield a 3-D flaw profile.

Magnetic testing and ultrasonic testing are two examples of active inspection techniques [18]. The ability of any technique to identify a fault is dependent on a number of factors. Each inspection technique will be suitable for detecting certain types of faults. Therefore to maximize the number of faults detected, a variety of tests should be performed. Simply obtaining the desired NDE signature is not sufficient, there also needs to be accurate and reliable methods for data analysis.

1.3 Previous Work

Many methods have been developed in an attempt to combine multi-sensor data. Most notably, a model was developed by the United States Joint Directors of Laboratories (JDL) Data Fusion

Group [19]. Additionally, artificial neural networks have emerged as a possible candidate for fusing large amounts of multi-sensor data [20]. Finally, VR has also demonstrated great potential as a platform in which the data can be integrated [4]. Table 1.1 illustrates some of the work performed in VR with respect to data analysis and NDE inspection.

Table 1.1 Previous work in VR applications for NDE

Research Group	Investigators	Research Focus
Brown University	A. Dam, D. Laidlaw, R.S. Simpson	Performed many experiments across broad ranges of applications using immersive VR for scientific visualization. [2,3]
Iowa State University	S. Udpa, L. Udpa, S. Mandayam, T. Hong, M. Pierce	Developed VR environments for displaying NDE gas transmission pipeline data. [4,8,20]
Clemson University	J. Vora, S. Nair, A.K. Gramopadhye, A.T. Duchowski, B. Melloy, B. Kanki	Showed the benefits of VR technology for aircraft visual inspection training. Research focused on presence and comparison studies. [29]
Hansung University	N.S. Lee, J.H. Park, K.S. Park	Investigated reality and human performance in a virtual world, and how human input can change a virtual world. [26]
University of Maine	J. Campos, K. Hornsby, M.J. Egenhofer	Investigated semantics and software definitions for exploring VR environments. Specifically, much of the research focused on time-evolutionary virtual environments. [54]
Knowledge Media Institute	T.D. Collins	Applied software visualization technology to support the use of evolutionary algorithms. [35]

1.4 Thesis Overview

1.4.1 Motivation

Virtual reality is an extremely powerful tool for data visualization applications. VR applications are often developed specifically for the visualization needs of a given project. In this respect, the VR tools developed are not universally applicable; they are used explicitly for one type of visualization for one specific application [3]. However, as VR becomes increasingly popular, a large number of visualization tools are being developed for broad ranges of visualizations [3]. Similarly, analysis techniques utilizing the benefits of VR need to be developed such that they can also be used to address the design-simulation-analysis cycle.

Data analysis by means of data integration is a common approach taken for the analysis of large complex data sets. All relevant information is pooled together and integrated such that the corresponding analysis is as thorough as possible. VR is a well-suited platform in which a visual analysis of the data could take place. There are many applications in which multiple forms of data are integrated and risk analysis is subsequently performed. NDE is one such example application; it will be examined in this thesis.

As mentioned earlier, digital signal processing techniques are used to isolate areas of interest and categorize signatures in NDE signals [21]. The development of these techniques in conjunction with sensor technologies is essential in detecting a wide variety of faults. However, faults do not just need to be recognized; the risk generated by the fault also needs to be understood. Discovering a fault before it poses a catastrophic risk is important for safety as well as for economics. It is important that all techniques support the inspector in making accurate, precise, time-efficient decisions. The NDE of gas transmission pipelines is one such example where monitoring and testing are essential to ensure safety and minimize monetary losses.

In the United States alone there are over 180,000 miles of natural gas transmission pipelines [22]. These pipelines form a major link in the nation's energy supply infrastructure. Given that the pipelines are an integral part of the nation's infrastructure, it is essential that the network is inspected and checked for integrity. Also, since much of the network is over fifty years old, the system is subjected to periodic in-line inspections. A device called a 'pig' is sent into the pipeline to inspect the integrity of the pipeline. There are many different methods for inspecting the pipelines. The most common are magnetic and ultrasonic testing techniques [21]. In addition to the magnetic and ultrasonic testing techniques, many new methods are being developed to monitor the pipeline conditions.

As of 1999, the sensors of a pig averaged more then 12 GB of data for every 100 miles of pipeline that was inspected [4]. Many algorithms and techniques have been developed specifically to aid in the analysis of these signals. For example, it is important to isolate areas of the signals with certain characteristics. An NDE signal could correspond to a benign occurrence (a tap, a T-section, a weld, a valve, etc...) or the signal could correspond to a potentially dangerous anomaly (a crack, pitting corrosion, mechanical damage, etc…) [4]. In the presence of an anomaly, gaining an understanding of the anomaly and potential failures resulting from that anomaly are essential in determining risk. This thesis illustrates stress corrosion cracking and the potential failures from such an anomaly.

One of the challenges in assessing risk is the integration of all relevant data. When evaluating the risk of a potential failure it is necessary that the location of the pipeline be taken into consideration. For example, a pipeline located beneath a field is much easier to excavate then one underneath a city; also the danger to human life would be higher in a city environment. Unfortunately, extensive, accurate logs and maps for the locations of the pipelines do not exist.

Also, all relevant signal processed data should be taken into consideration. A VR platform for integration of this multi-sensor data is illustrated in Figure 1.6.

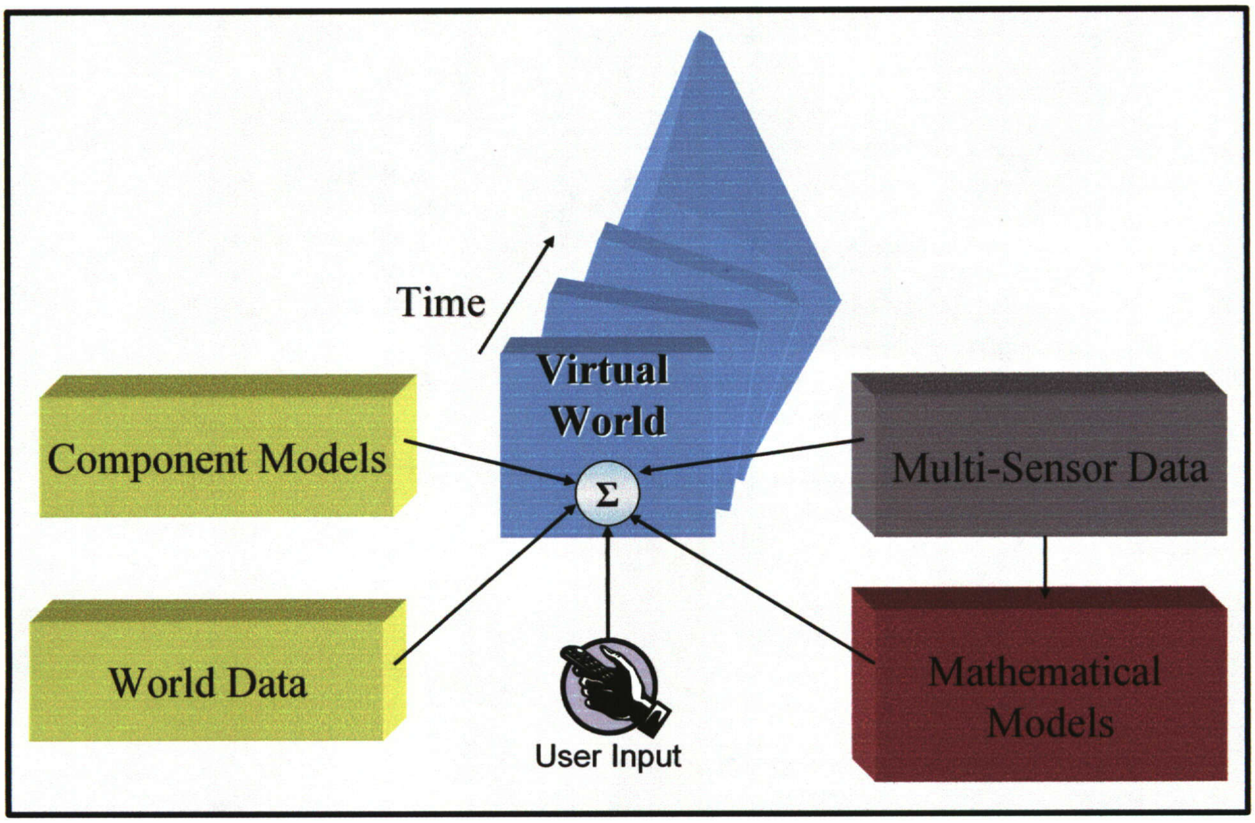


Figure 1.6 – Evolutionary VR world.

Within the virtual world the user has the ability to visualize the relevant data. Additionally, the user is given the ability to evolve the data over time and create various scenarios. An evolutionary virtual world allows the user to ask "What if…?" The virtual world will then evolve over time as a function of the user inquiry and system data. Using the evolutionary scenario that has unfolded, the user is empowered to more thoroughly analyze the data and draw relevant conclusions.

1.4.2 Objectives of Thesis

The main objectives of this thesis are:

1. *To develop a framework for multi-sensor data visualization in virtual reality.* The framework should support diverse applications and should operate effectively over several hardware and software platforms.
2. *To develop a method for implementing time-evolutionary models for virtual reality.* Both an overall framework and specific evolutionary modules are to be developed. Evolutionary modules include deterministic modeling algorithms, digital filtering techniques, genetic algorithms, and hidden Markov models.
3. *To compare the suitability of the various algorithms for virtual reality scenario development.* The comparison should focus on features that have the most significant impact in both the development and deployment of the VR environment.

1.4.3 Expected Contributions

This thesis outlines a method for visualizing multi-sensor data in a VR environment. The goal is to formalize the integration process to support the use of VR as both a data visualization and data analysis tool. The formalization of the integration process is important so that the techniques used in VR data analysis can be applied to many different applications. The thesis also outlines a framework for the evolution of that data. The framework is to provide a three-way bridge between the VR world, the user, and the mathematical models being developed to analyze the data. In conjunction with the evolutionary structure, individual algorithms are implemented to obtain multi-input time evolutionary virtual worlds.

Specific implementations of the individual algorithms (deterministic modeling, filtering, genetic algorithms, and hidden Markov models) will illustrate the potential of each method for time-evolutionary VR analysis. Finally, an analysis is performed to determine the strengths and weaknesses of each algorithm for the VR evolutionary visualization and analysis. Recommendations for future work are given based on the performance of each algorithm. Additionally, recommendations are provided for work regarding the use of VR in the design-simulation-analysis cycle.

1.4.4 Scope and Organization

The scope of this thesis pertains to multi-sensor data visualization and the evaluation of evolutionary algorithms used for risk assessment in a VR environment. Specifically, the application being demonstrated represents the NDE of stress corrosion cracks in natural gas pipelines.

Chapter 2 provides the background for the techniques used in accomplishing the thesis objectives. First, VR visualizations are described. Next, the four types of evolutionary algorithms being investigated are presented. This is followed by the description of traditional methods for performing risk analysis. Finally, the methodology used for the evaluation of the algorithms is presented.

Chapter 3 provides an explanation of the methodological approach, assumptions, and techniques used to accomplish the outlined objectives. The methodology for multi-sensor data integration is presented first. This is followed by the data representation form for time-evolutionary data predictions. Next, the overall evolutionary structure is described, followed by the model design and development for each algorithm. Finally, the evaluation criteria are described.

Chapter 4 presents the results for a number of different evolutionary scenarios. The overall data integration process is first presented. Evolutionary worlds for providing risk analysis are then discussed. Finally, the comparison of the algorithms is performed.

Chapter 5 provides a summary of accomplishments, conclusions from this work, and recommendations for future work.

CHAPTER 2 – BACKGROUND

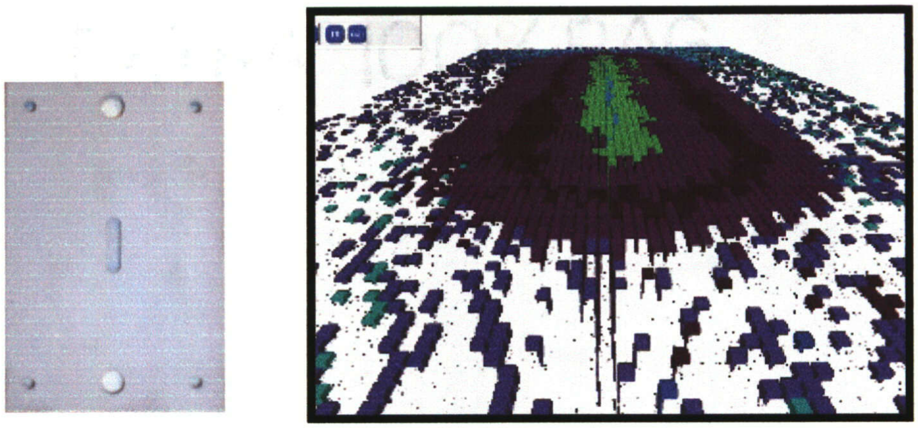
2.1 Virtual Reality

2.1.1 Data & Object Representations

When performing a visual analysis it is essential that the data and object representations be optimized for analysis. Optimization is typically performed with reference to the number of frames per second displayed, and the complexity and number of components. For the virtual world to appear smooth and respond to the user's commands in real time, the number of frames per second must be high. Approximately 60 frames per second is an ideal rate; rates at or below 10 frames per second are noticeably jerky and can result in simulation sickness [1]. At the same time the frame rates are being optimized, the objects within the world need to be optimally displayed with suitable detail and complexity to convey the desired image. Because of the finite amount of machine power available to drive the hardware, there exists a tradeoff between frame-rate and complexity. The goal is to find a representation of suitable design that requires the least amount of overhead. For both data and object representations there are many methods of representation.

Object representations can be broken down into three main groups – raw data, surfaces, and solids. Each representation type has both positive and negative aspects; no single representation is superior for all applications. For a specific application, the best representation is a function of the type and form of the data along with the purpose of the representation.

Raw data representations are the most basic form of data visualization. Each data point is represented visually with a point or glyph [23]. The point cloud will give a three dimensional depiction of the raw data. Each point in a point cloud can have an x, y, and z coordinate along with a representative value. For object data, i.e. an object scanner application, there will be no value, just a cloud of points that approximates the shape of an object. For data / sensor applications, the point cloud will represent the location of each sensor and the corresponding value. One such representation of raw data is illustrated in Figure 2.1 below. The data points in (b) represent time-of-flight measurements of an ultrasonic scan of object (a).



(a) Specimen **(b) Data visualization**

Figure 2.1 – Data point representation.

As seen in Figure 2.1, the values are represented in a 2-D pseudo-color mapping at the measured sensor location. Raw data representations can take forms other than a cloud of points. A visualization of vector fields with arrows is also a raw data representation.

Similarly, surface representations can be depicted with various forms. Pseudo-color slices are representations that are often used to visualize data. A color slice is a surface representation of data that interpolates between data points to convey a smooth transition between measured values. Pseudo-color maps like that used in Figure 2.1 can be used to represent the data values. The same scan depicted in Figure 2.1, is displayed with a pseudo-color slice in Figure 2.2 (a). Additionally, a topology can be displayed where the grid level is also a function of the data value. A topology is useful in generating a true 3-diemsnional representation of the scanned data; Figure 2.2 (b) illustrates this concept.

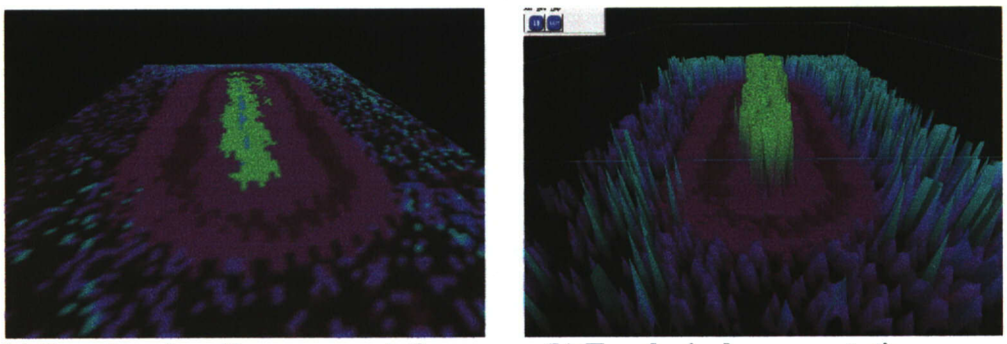

(a) Pseudo-color slice representation. (b) Topological representation.

Figure 2.2 – Pseudo-color representations.

Mesh grids are also common representations used for the display of surface objects. A mesh grid represents the contours of an object through the display of a grid of the object's surface. Furthermore, mathematical functions can be used to describe the surface of an object.

Raw data and surface representations are most often used to display data. However, solid representations can be useful for displaying objects, because real world objects are solids. The most basic of the solid object representations is a voxel. A voxel is a 3-dimensional cube unit that describes the smallest unit of volume that can be discretely represented by the VR system. In this respect, a voxel is to a 3-dimensional representation what a pixel is to a 2-dimensional representation.

The next type of solid object representation is 'geons'. Geons refer to the most basic and primitive of shapes [24]. Sets of geons can be used in various combinations to make increasingly complex shapes. Geons can be manipulated in terms of scale, translation, skew, etc., in an attempt to create the advanced objects. Figure 2.3 illustrates basic geons and a few corresponding complex objects that can be created from basic shapes.

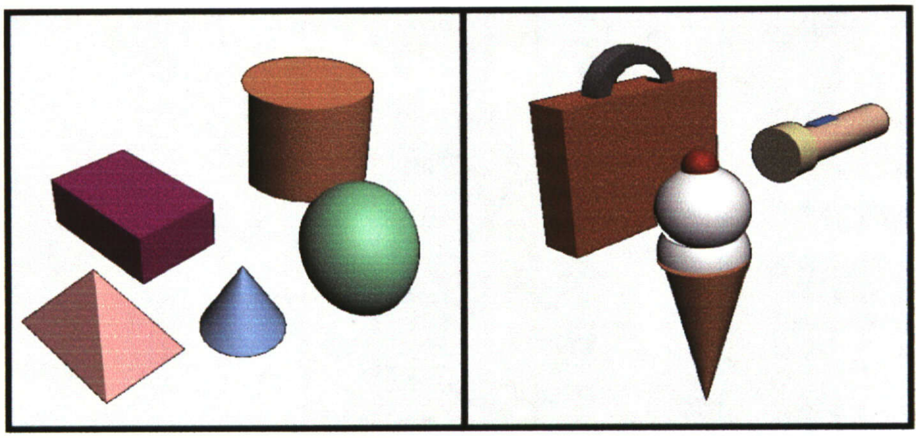


Figure 2.3 – Object representations using geons.

The virtual reality modeling language (VRML) is one structure that uses the same concept of geons and a hierarchical structure to compose complex objects. The VRML standard was first designed to allow for graphics to be rapidly transferred from and displayed in different systems. VRML 1.0 was used quite extensively, and its roots can be traced back as early as 1989. To incorporate features such as interaction and animation, the VRML 2.0 standard was developed and published in 1997 [25]. The VRML standard has advanced beyond simple shapes and can incorporate lighting, sound, animation, and scripts. Because of the power and diversity, VRML files are used extensively in VR applications as the base for physical modeling.

2.1.2 Design for Interaction

As described in Chapter 1, there are three components that define VR – immersion, navigation, and interaction. These components all incorporate different aspects of human-computer interaction. There has been extensive research in the field of graphical user interfaces and the types of visualizations that work best for a given set of requirements [27]. Much of the research regarding the interaction on a traditional computer can be directly translated to a virtual environment. One such aspect of interactions is information hiding. A user should be given the amount of control needed to perform a task and no more [26]. If presented with too much information and too many options, the user can become confused and the process will suffer. Therefore, when designing a human driven virtual environment it is essential that care be taken to determine the amount and types of controls that the user will have.

Another aspect of human-computer interaction is navigation within the virtual world. Normally, the navigation controls are governed by the hardware being used. A joystick or a control pad would provide the user with a unique method for exploring the virtual world. Although the hardware is usually fixed, there are some software techniques that can be employed to aid the user's ability to navigate the world. Visual cues and standards within the world will greatly aid the user's ability to control navigation and placement within a virtual world. Gridded references serve as the simplest form of registration. With a grid for reference, the user can easily become oriented within the virtual world and easily compare lengths and distances. To aid in the orientation process color-codes can be added to the grid. For example, a grid representing land could have white at the top and a brown or green towards the ground. Intuitively, users would be able to orient themselves in this world very easily. As users feel more comfortable and natural in a virtual world a greater sense of presence will be created.

2.1.3 Data Integration

As sensor technologies have advanced, data integration has become increasingly popular; with this advancement of sensor technologies the amount of data that can be gathered has increased sharply. Data fusion methods have therefore been used as one method for combining and analyzing these large data sets. There are a number of different broad categories that traditional estimation algorithms can be divided into –classification methods, inference methods, and artificial intelligence methods [28]. For example, a simple weighted averaging method can provide ample processing for certain applications, whereas classification methods attempt to divide the feature space into distinct regions. Each region will uniquely represent a certain class or phenomenon within the data. Inference methods can define boundaries based on geometrical or statistical observations within the data. Bayesian inference methods are those that are based on Bayes probability rule [40]. The final group of algorithms is the artificial intelligence (AI) methods. AI methods represent a high-level and more complex decision making process. Expert systems, fuzzy logic, and artificial neural networks are all methods that utilize a complex logic or a high-level probabilistic structure to perform an action or classification.

Within each technique there are additional ways in which the data can be fused. The most common division within each of the groups is the level at which the data fusion occurs. Data can be fused at the sensor level; sensor level fusion means that each individual reading will act upon the output of the system. Other methods utilize high-level structures in which the fusion takes place. High-level fusion includes groups or sections of data points.

Data integration that takes place in a virtual environment is an experiential type of data fusion. The data being integrated are fused visually. VR also offers a host of complementary techniques employed by the virtual hardware such as tactile or auditory outputs. Experiential data fusion allows the data fusion to take place on a platform where the human user is empowered to make critical decisions. Instead of attempting to mimic the human decision process, a VR platform provides the human user additional tools that he / she can use to make decisions. The focus of VR data fusion is therefore integrating the physical world (graphical data), the data from sensors (measurement data), data obtained from any analysis (functional data), and user inputs.

2.1.4 Benefits of VR

The main benefit from VR is that it empowers users with a tool for analyzing and understanding complex structures. Human vision has evolved specifically to allow for complex visual decisions to be made. More then 50% of the neurons in the human brain are devoted to vision [2]. VR taps into users' innate capabilities and enhances their abilities. Utilizing this skill is imperative because there are many examples of phenomena that humans can visualize, but are difficult to describe statistically.

A benefit of visualization with VR is that it allows users to visualize data and phenomena in true 3-dimensional forms. Therefore, the natural state of data can be replicated. Many studies have shown how immersion, navigation, and interaction all allow for complex 3-dimensional structures to be more thoroughly understood [26,27]. As mentioned earlier, VR is not only a tool for visualization but it is also a tool for analysis. In a study regarding NDE inspection it was shown that using VR can increase both the speed and accuracy of the inspection process [29].

Since a virtual platform offers such a range of parameters to be adjusted it is important certain design aspects be taken into consideration. The optimal benefits of a VR world will vary with the creator of the world, the user of the world, and also the application being investigated. As described in section 2.1.1 data representations are world variables that can greatly impact the benefits gained from using a virtual world. Similarly, the human-computer interactions described in section 2.1.2 play an important role in determining the benefits from a virtual environment. It is therefore essential that all algorithms support the required user interaction with the data.

2.2 Evolutionary Algorithms

2.2.1 Deterministic Modeling

Deterministic modeling techniques use models that represent the precise physics of a given environment. The environmental parameters are directly input into the functional model. From the functional model and the environmental parameters, a description or predicted behavior is output. This basic I/O representation is illustrated in Figure 2.4.

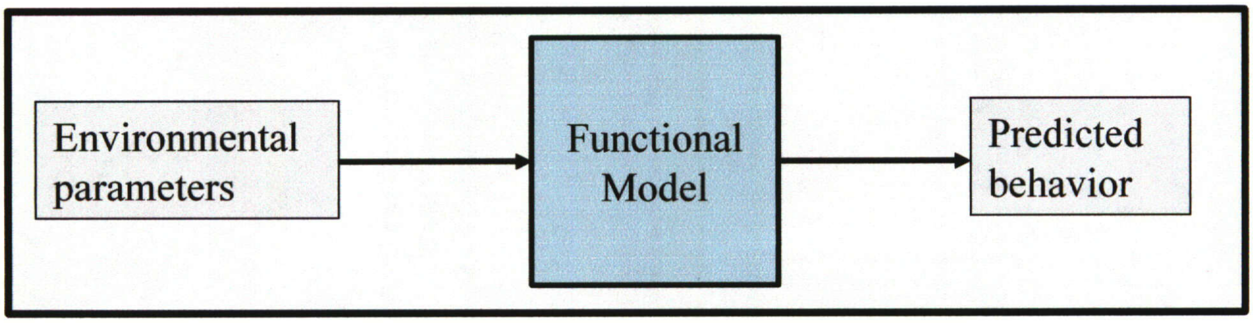


Figure 2.4 – Deterministic modeling I/O

Within the larger functional model, smaller components in the model are usually analyzed separately. The collection of components is assembled and analyzed to obtain a model for the overall system. Breaking the overall model into its component pieces allows for a more direct representation of the environmental effects and the physics being modeled. Additionally, the smaller models have less variability then their large complex counterparts, therefore the smaller models will be simpler to develop and easier to solve.

Most simply, a deterministic model can be described as a system of equations or functions from which a solution gives practical insight into the problem being investigated. Since most of the deterministic models involve varying parameters, differential equations are a common form for the models to take [30]. The differential equations will vary in complexity depending on the nature of the problem.

Assumptions are essential in simplifying the problem in order to obtain a manageable model. A deterministic model will not provide meaningful results if the assumptions made are inaccurate. The solutions obtained need not match reality exactly, but should be accurate to the degree required for solving a specific problem. Selecting appropriate numerical methods is also essential in deriving a deterministic model. Numerical methods introduce error in a number of ways including rounding and propagation errors. Although deterministic models can produce accurate results, the modeling of complex phenomena requires extensive derivation and experimentation.

2.2.2 Filtering Techniques

Evolution can be simulated through the use of a parametric model, an IIR filtering technique. The parametric modeling technique is a system identification procedure [32]. A signal *y* is estimated and predicted through the weighted sum of previous inputs and outputs. This is known as an auto-regressive moving average model (ARMA). Figure 2.5 below illustrates basic I/O of an ARMA model.

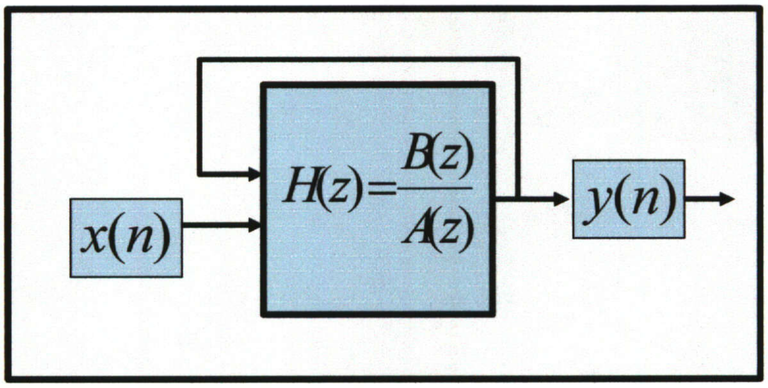


Figure 2.5 – ARMA model I/O

Where *H(z)* represents the filter's transfer function in the z-domain. The difference equation for a discrete-time system with output y and input x is given in equation 2.1 below

$$y(n)+a_1y(n-1)+...+a_py(n-p)=b_ox(n)+b_1x(n-1)+...+b_qx(n-q) \qquad (2.1)$$

Typically, an ARMA model has an input corresponding to zero-mean white noise. The use of a white-noise input effects applications differently, but the additional input usually allows for closer match of an unknown system. The filter is considered a pole-zero filter of order (p,q). When $p=0$, the corresponding filter is an FIR, all-zero filter. The all-zero filter is known as a moving-average (MA) model and is described by:

$$y(n)=b_ox(n)+b_1x(n-1)+...+b_qx(n-q) \qquad (2.2)$$

When $q=0$, the filter is an all-pole filter. The all-pole filter is an autoregressive (AR) model and is described by:

$$y(n)=-a_1y(n-1)-...-a_py(n-p)+x(n) \qquad (2.3)$$

For non-zero values of p and q the ARMA model is described in equation 2.1. Physical signals do not typically obey the model exactly, but the ARMA representation serves as a suitable approximation. The model serves as a suitable approximation, because the filter can often match real world systems to within a prescribed error tolerance.

Calculating the filter coefficients for the model is a complicated task and therefore there are a variety of methods that can be employed. The calculation of the ARMA components focuses on finding an IIR filter whose output matches the given output under a known input. When the input is noise the all-zero filter is referred to as a MA model. However, if the input is assumed to be arbitrary, the model is referred to as X. Certain implementations for calculating ARMA coefficients can be more accurately described as ARX, since the algorithm calculates the coefficients for an arbitrary input [33].

2.2.3 Genetic Algorithms

2.2.3.1 History of Genetic Algorithms

Throughout the field of signal processing it can be seen that a number of algorithms have been inspired by biological origins. At the forefront of this research is the chromosome, the encoder of life. Throughout earth's history there has been reproduction and the propagation of a wide variety of species. Through natural selection and mutations, nature has had the ability to populate the gene pool with the individuals that are most fit for survival. This concept of biologic evolution has sparked the attention of signal processors, most notably John Holland.

In the 1970's John Holland realized that nature's model could be used to solve complex mathematical problems. Just as natural selection acts on organisms, a mathematical system could manipulate strings of numbers [34]. Through this structure, complex problems could be solved just as complex organisms have developed in nature. The mathematical systems could mimic nature and correspondingly manipulate the component members. The overall I/O relations for genetic algorithms are depicted in the figure below.

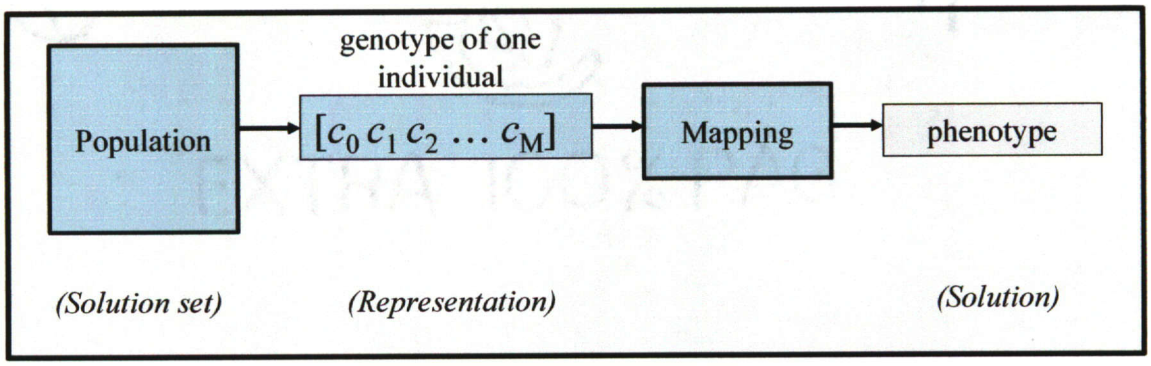


Figure 2.6 – Genetic Algorithm I/O

Through this idea, genetic algorithms grew in popularity. Although genetic algorithms share their name with the natural process, there is no direct relation between the two. The natural process of genetic manipulation sparked the base idea for genetic algorithms, but the mathematical interpretation has developed independently. The mathematical algorithms are extensively used for optimization problems. However, the overall flow and structure of the model is very suitable to a wide range of mathematical applications [35,36,37,38,39].

2.2.3.2 Algorithm Overview

The overall concept for genetic algorithms is described as follows. A population of individuals is representative of a solution set, where each individual is an encoded version of the solution. Each of the individuals is then ranked based on the quality of the solution it would provide. From the entire population, a given number of individuals are selected to the mating pool. Individuals with higher rankings are more often selected. These individuals are combined together to form new solutions. After the mating process, the offspring are subjected to mutations. Finally, the new individuals are added to the population and a new population emerges. This process is shown in Figure 2.7 below

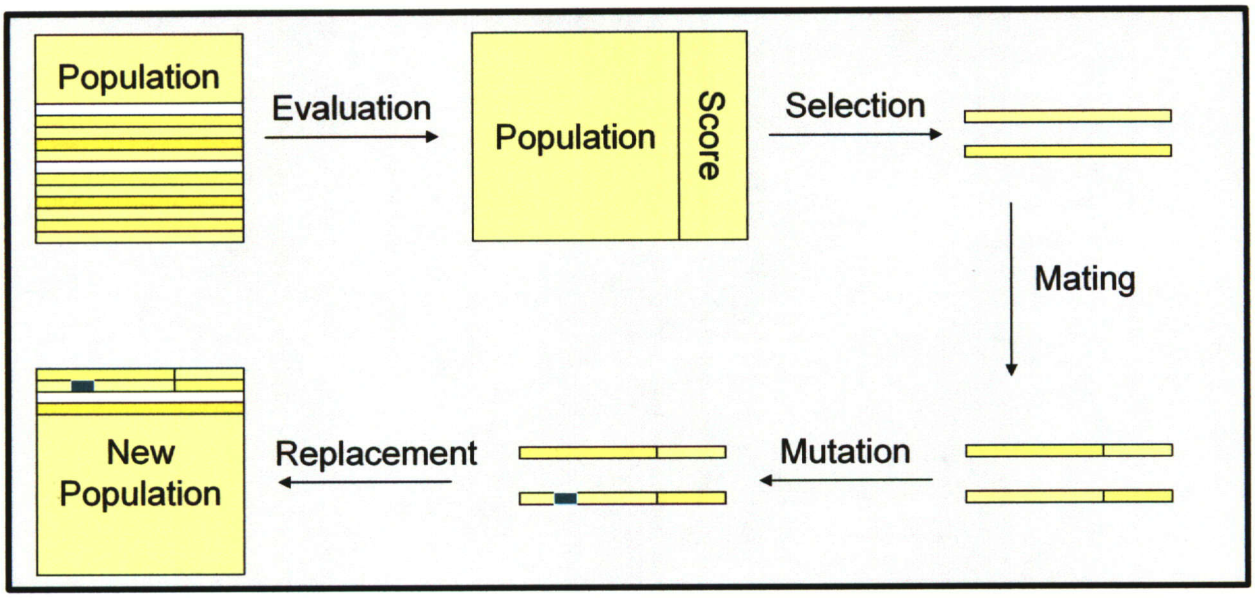


Figure 2.7 – The overall process for genetic algorithms.

The figure above depicts the overall process of genetic evolution, but does not give insight into the variety of ways that each step can be realized. The following sections outline some of the popular techniques that are used to carry out the various processes within the overall genetic algorithm.

2.2.3.3 Genotype and Phenotype Relationship

The first step in setting up a genetic algorithm is to create the chromosome. The chromosome is the code that represents a solution. The information located directly within the chromosome is known as the genotype. The genotype is the encoded information about the appearance of the larger whole, the phenotype. The phenotype represents the outward visual appearance of the individual, the solution.

There are a variety of structures that are used for the genotype representation. A binary coding is one such representation. The binary digits in the genotype would be representative of features within the phenotype. A feature would be coded through a string of 1's and 0's.

Conversely, a string of 1's and 0's could be decoded to create a representative phenotype. The genotype representation does not have to be binary, a numerical (base 10) representation could also be used. The most important factor in creating a genotype is that is accurately encodes the phenotype, and that the phenotype can be recovered from a known genotype.

2.2.3.4 Population Generation

Each genotype represents only one individual. Groups of individuals create a population. When initializing the genetic algorithm, an initial population must be set. If there is no known information about a previous population of similar individuals, a population is typically populated with random individuals. If the genetic algorithm is being used to evaluate a known set of solutions then the population will represent the genotypes of those known solutions. The initialization of the population is important because it is a factor in the time required to reach a desired result.

2.2.3.5 Evaluation Function

Each individual within the population is given a weight or a score. The score of each individual is based on the evaluation of its chromosome. A cost function is used to determine the fitness of each genotype. The cost function or fitness evaluation operates on the genotype. Based on the genotype, the function estimates how adept the phenotype would be for the given application.

2.2.3.6 Chromosome selection

Natural selection takes place when specific chromosomes are chosen for mating. The chromosomes that are chosen most often are those that have a higher fitness. If the strongest individuals mate, the new population will be more fit then the original population. The most popular method for choosing the individuals for mating is the Roulette Wheel Selection technique. Using this technique, the probability that a given individual will be selected is directly proportional to its fitness score. The natural selection process allows the possibility of any member mating, but weights the superior individuals with a greater chance. The concept of chromosome selection is illustrated in the Figure 2.8.

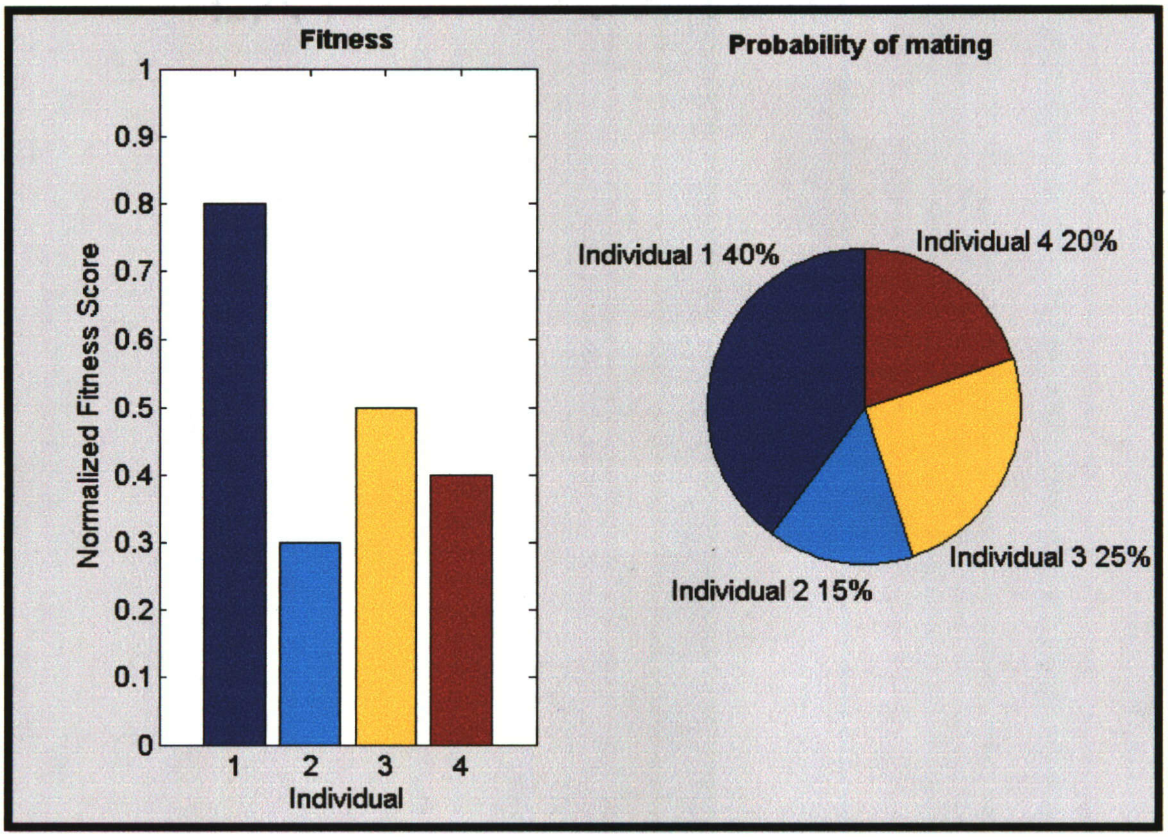


Figure 2.8 – Roulette-wheel parent selection technique.

This concept can also be expressed mathematically as:

$$P(i) = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.4}$$

Where f_i, is the fitness score for the i-th individual, and N is the total number of individuals.

2.2.3.7 Chromosome mating

The process of combining two chromosomes allows for the creation of two new individuals. The information contained in the two chromosomes is combined such that the new individual contains a part of both the original parents. In one-point crossover, a random point within the chromosome is chosen. The first section of the first individual is combined with the second part of the chromosome in the second individual and vice versa. This process will create two new individuals that may each have certain characteristics from the two parents. An example of one-point cross-over is described in equation 2.5.

$$\begin{aligned}
&\text{Cut-point}: c_p \\
&parent_1 = [c1_0 c1_1 c1_2 ... c1_{M-1} c1_M] \\
&parent_2 = [c2_0 c2_1 c2_2 ... c2_{M-1} c2_M] \\
&child_1 = [c1_0 c1_1 ... c1_p c2_{p+1} ... c2_{M-1} c2_M] \\
&child_2 = [c2_0 c2_1 ... c2_p c1_{p+1} ... c1_{M-1} c1_M]
\end{aligned} \tag{2.5}$$

2.2.3.8 Mutation

The mutation function operates on newly formed individuals. The possibility of mutation allows the algorithm to introduce new features that were otherwise not present in the population. If the new trait is favorable, then that individual will mate often and create more individuals with that same trait. The idea of generating a population through a series of mutations is often referred to as genetic computing [39].

The mutations that occur are randomly determined. If the chromosome has a binary representation, during a mutation, a 1 will be changed to a 0 and vice versa. The probability of a chromosome being mutated is traditionally a predefined characteristic of the algorithm. The mutation rate will then determine how often a sudden change in a chromosome takes place.

2.2.3.9 Population Replacement

Finally, after the new individuals are created they are inserted into the overall population. The introduction of these new individuals can be performed in a number of ways. The simplest method of performing genetic replacements is a replace-all strategy. In a replace-all strategy the newly generated population replaces the entire old population. This technique does not take the fitness of each individual into consideration. If the individual's fitness is to be taken into consideration, other replacement techniques can be employed.

A selective replacement allows for the least fit members of a population to be replaced by the new members regardless of the fitness of the new members. Finally, a top-selection process can be used to obtain a population that has the top members. After the new members are created the old and new individuals are tested and only the most fit of all the members survive to join the next generation population.

2.2.4 Hidden Markov Modeling

There are many applications that utilize Hidden Markov Modeling (HMM), but time-series evolution and risk assessment are two fields that have heavily used HMMs. The input of a HMM is a set of probabilities, where the probabilities can be obtained from a prior knowledge or through a learning process. Using these probabilities, an HMM will output a sequence of states. The sequence of states corresponds to the solution being represented. This concept is illustrated in Figure 2.9 below.

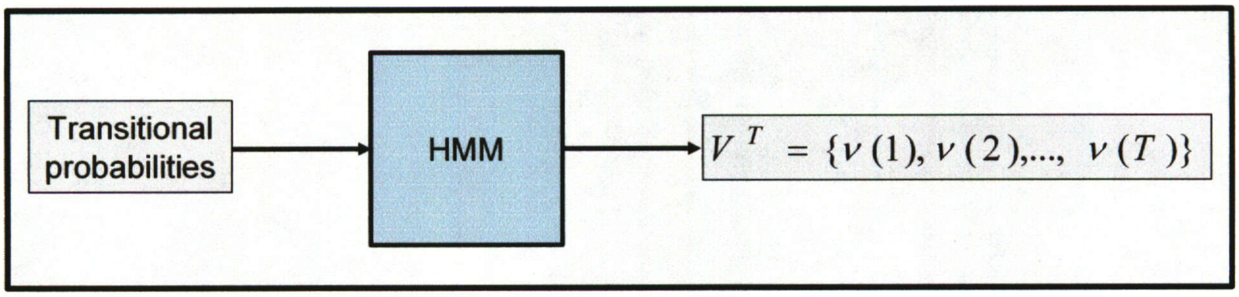


Figure 2.9 – HMM I/O

A first order, discrete time, Markov Model is illustrated in the figure below [40].

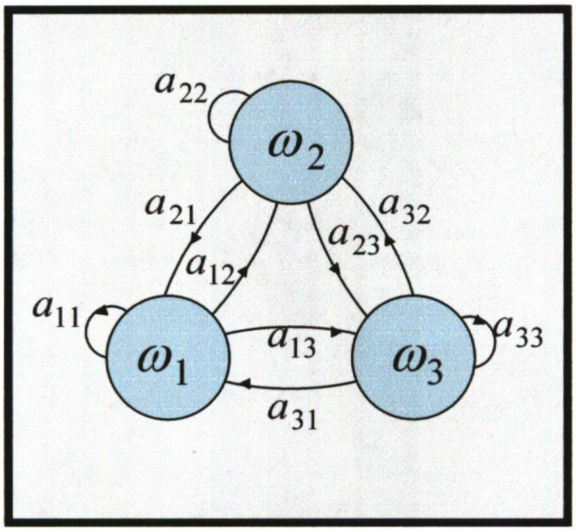


Figure 2.10 – A first order, discrete-time Markov model.

A Markov model is a state machine where each transition is associated with a probability. A particular sequence of states, of length T, is denoted as:

$$\omega^T = \{\omega(1), \omega(2),..., \omega(T)\} \tag{2.6}$$

where, $\omega(1)$ is the state at time-step 1. Each subsequent state is a function of the previous state. The transitional state probabilities can therefore be given as:

$$a_{ij} = P(\omega_j(t+1) \mid \omega_i(t)) \tag{2.7}$$

From state *i* to state *j*, the transitional probability, a_{ij}, is defined as the probability of being in state ω_j at the subsequent time-step, given that that system is in state ω_i at the current time-step. For a given instance, the sequence of states would be given as seen in the example below:

$$\omega^4 = \{\omega_1, \omega_3, \omega_3, \omega_1\} \tag{2.8}$$

It is important to note that on subsequent time steps the system need not change states; the system can remain in a single state for multiple time-steps. Also, for a given instance, the system is not required to enter into every state.

The Markov model is a suitable state machine model, but it is not applicable to a large number of real world applications. In real world applications, the user cannot access states directly. However, some measured or emitted feature from a state is accessible. A hidden Markov model can therefore be used to illustrate that at every time t, the hidden state $\omega(t)$, will be associated with a visible state $\upsilon(t)$. The figure below illustrates a first order, discrete time HMM [40].

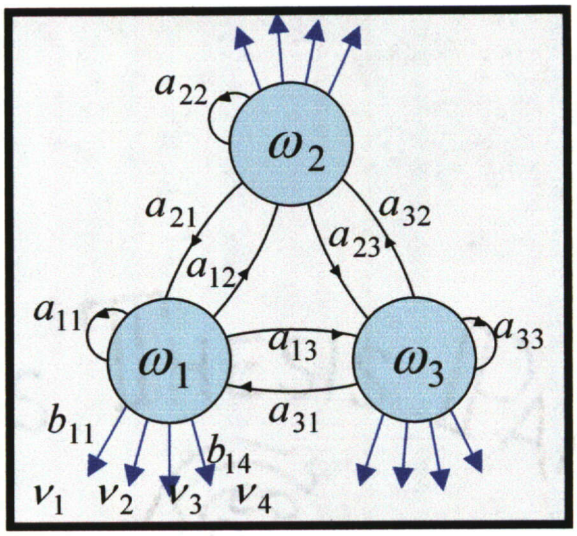


Figure 2.11 – A first-order, discrete time HMM.

From this model, a particular set of visual states will be defined – rather then a set of hidden states. The set of visible states observed is defined as:

$$V^T = \{v(1), v(2), \ldots, v(T)\} \tag{2.9}$$

A given instance of observed values can therefore be represented as seen in the example below:

$$V^4 = \{v_1, v_3, v_4, v_1\} \tag{2.10}$$

This sequence represents the order in which the visible states were observed. At each time-step a given visible state was emitted. The state emitted was a function of the current hidden state. The transitional probabilities for a visible state are given by:

$$b_{jk} = P(v_k(t)) \mid \omega_j(t)) \tag{2.11}$$

where the transitional probability from hidden state j to visible state k, b_{jk}, is equal to the probability of visible state k given hidden state j. The hidden state transitions are still defined by equation 2.7 for a HMM.

There are four ways HMMs can be used – sequence determination, evaluation, decoding, and learning [40]. Sequence determination is the primary focus of HMMs in this thesis. Sequence determination means that all transitional probabilities are known. A resulting set of sequences V^T is then determined. In evaluation, the transitional probabilities and the visible sequences are known. The calculations then focus on determining the probability that the sequence occurred given the specific set of transitional properties. In decoding, the transitional probabilities and the sequence of visible states are again given. The focus is determining the most likely sequence of hidden states that led to the sequence of visible states. Finally, in learning a set of visible sequences is given. From these instances, the set of most likely transitional probabilities is determined.

There are a number of algorithms that have been developed for each of the four usages of HMMs. The parameters can be compiled in a static or adaptive state. Much of the recent works outline methods by which the state transition probabilities are adaptively calculated. Other work has focused on the ability to adaptively determine the number of hidden states in a network [41].

2.3 Scenario Development & Risk Assessment

For almost any industry the process of risk management is extremely important. The process of risk analysis is performed to avoid potentially dangerous and costly situations. Risk analysis tools and risk measurement techniques allow for decisions to be made such that the various cause and effect chains can be taken into consideration. There are a wide variety of approaches that can be taken towards performing risk analysis, so this section presents the important concept common to many techniques.

A traditional definition of risk is: "A factor, thing, element, or course involving uncertain danger; a hazard [42]". Risk can be considered "an uncertainty surrounding the loss from a given undesirable event resulting from a hazard [43]". Similarly, risk is "the combined effect of the probability of occurrence of an undesired event and the magnitude of the event [44]". Within all three definitions there are common threads. Risk is associated with the occurrence of an event. This event is to some degree unexpected and has unwanted consequences. Risk is associated with uncertainty and therefore risk analysis or risk predictions will not be absolutes, but insight into possibilities.

2.4 Evaluation Criteria

Typically, signal processing techniques use a form of error criteria or a cost function to assess the functionality of an algorithm. Just as a genetic algorithm uses a cost function to assess the fitness of an individual, a digital image restoration technique would use a mean squared error to quantitatively describe the algorithm's performance [13]. However, to achieve such a measure, the algorithm output needs to be compared to a known correct or acceptable output. In applications such as risk assessment there is no given output that is necessarily superior to any other [45]. For such applications other evaluation measures need to be considered.

In this thesis, the evaluation of each algorithm is not based on the performance of the algorithm with respect to the output, but rather to a predefined set of goals or criteria. Since the evaluation is with respect to a predefined set of goals, quality management and quality control methodology can be applied [46]. There are seven tools, known as the seven management tools, that can be employed throughout the development process to ensure algorithms meet the required specifications. These same principles can be applied to an end-evaluation of the evolutionary algorithms.

The first tool, a matrix diagram, can be taken from the seven management tools. A matrix diagram supports a graphical representation that focuses on the relationship between factors [47]. Relationships can be illustrated in terms of positive or negative correlation in addition to the overall strength of the relationship. Through the use of this tool, an analysis can be made about the overall interactions between each algorithm and the goal it is trying to reach.

The second management tool that can be taken from the seven management tools is a prioritization matrix [47]. A prioritization matrix combines both qualitative and quantitative analysis in the same procedure. Using a prioritization matrix, each algorithm can be given a score based on a set of evaluation criteria. The evaluation criteria are weighted such that the most important, critical factors are given the largest values. The overall score is then a reflection of not only the algorithm's performance with respect to a specific goal, but of the algorithm's performance with respect to its overall operation. This type of comparison will allow for the evaluation of the various evolutionary algorithms.

CHAPTER 3 – APPROACH

3.1 Overall Visualizations

3.1.1 Data Integration

In this thesis, the main purpose for developing a virtual environment is to perform data inspection and data analysis. In general, the goal of an inspection system or measurement system is to gather data that represents the true state of reality. A virtual world is the proposed platform in which the collected data can be integrated with other forms of data for subsequent analysis. The virtual platform allows for both the experiential integration and evolution of a range of data types. In Figure 3.1 the overall process of world development is shown. The figure illustrates the processes by which reality can be modeled and a virtual world created in its likeness.

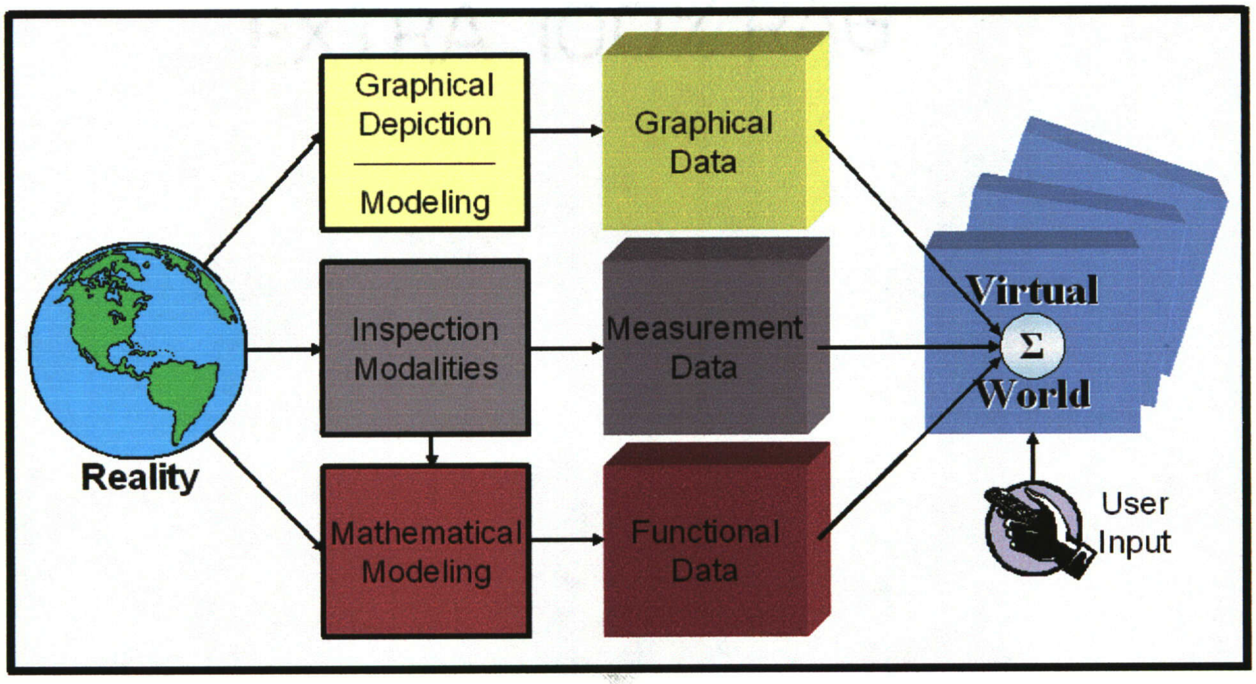


Figure 3.1 – Creation methodology of a virtual world.

The first step in creating a virtual world is to obtain a graphical depiction of reality. A graphical depiction forms the basis or background for the entire virtual world. The models are obtained through physical modeling means; dimensioned drawings or images can be used to formulate the graphical data. Graphical data can be represented with a variety of file formats, but most traditionally VRML files are used. The detail needed in the graphical data depends on the specific object being modeled and its relevance in the overall analysis.

The second form of data is measurement data. Measurement data is any data that were obtained using sensors. It is the simultaneous visualization of multiple sensors that allows VR to become a platform for data integration and inspection. The main goal in the visualization of measurement data is to construct a representation that is as close as possible to the true nature of the data.

The third form of data in the visualization process is functional data. Functional data representations are derived from mathematical operations. Phenomena in the real world can be mathematically modeled and described using functional data. Functional data also include results derived from the analysis of measurement data. The purpose of functional data is to provide insight or additional information about the virtual world.

All three modalities of data can be integrated on a VR platform. In addition to these three forms of data, user interaction is possible. The user input is used to guide the evolution of the virtual world over time. This overall framework provides a basis for outlining the visualization and analysis needs of world development. Additionally, the framework allows for some standardization in the process of world creation.

In this thesis, the VR approach is used for the integration of various data types for the nondestructive evaluation of gas transmission pipelines. First, graphical representations were created to provide an overall background for visualization. Next, the multi-sensor data were obtained to allow for an inspection of the condition of a specimen. After that, the data were processed to gain additional insight about the raw sensor data. Finally, a method for user control was developed for aiding in the decision making process. This data integration scheme is illustrated in Figure 3.2 below.

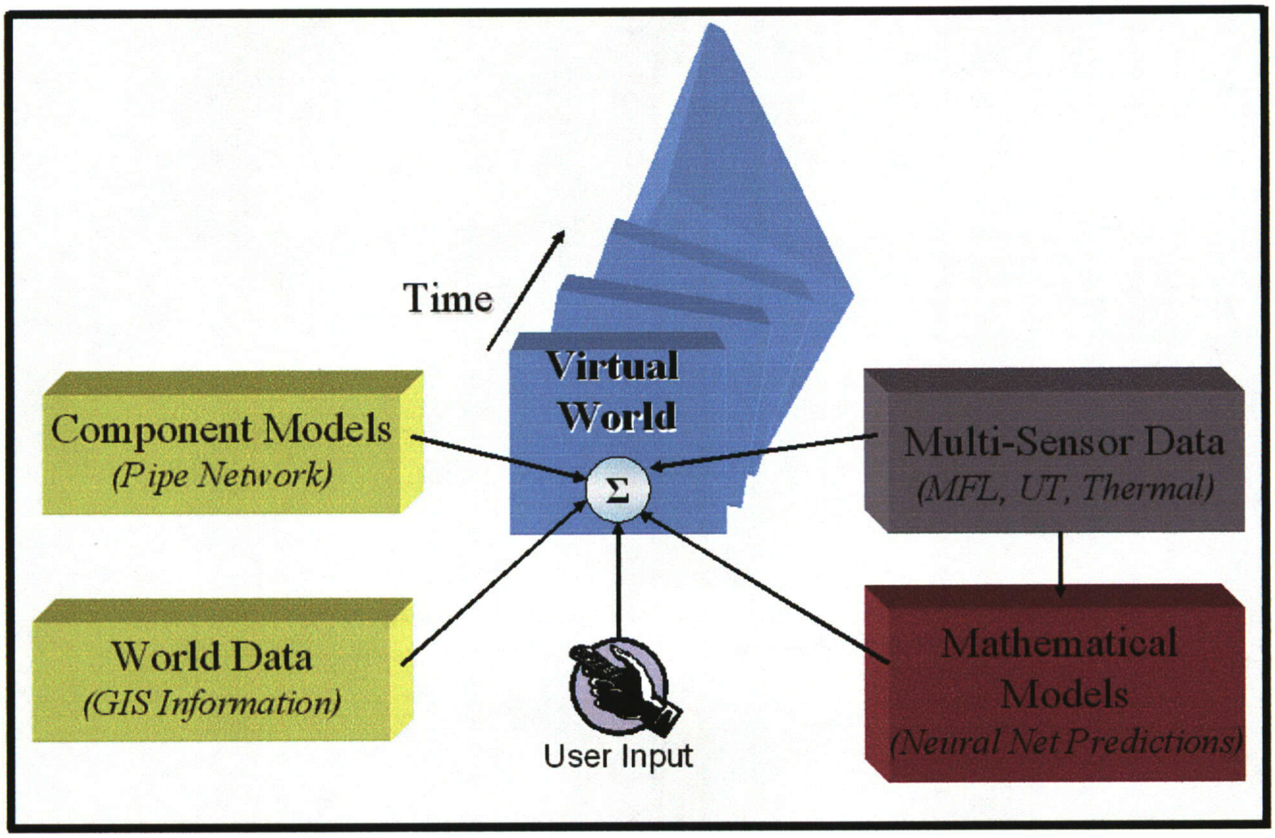


Figure 3.2 – Multi-sensor data integration for the non-destructive evaluation of gas transmission pipelines.

In this application, the graphical data for the virtual world is divided into two sections – component models and world data. The component models are the individual pieces that combine to form the pipeline network. There are a wide variety of components that can be found within a pipeline network. These models include pipe sections, flanges, welds, sleeves, T-sections, taps, check valves, ball valves, anchors, and anomalies (crack, pit, etc.). The component models are representations of the objects that are under direct investigation. The world data form a representation of the environment in which the pipeline exists. The geographic information system (GIS) can be used to obtain such variables as the land topology, population figures, land usage, location of roads, rivers, etc. The world location data is useful in dealing with remediation measures.

The multi-sensor data is representative of the information obtained through an inspection of a pipeline. Various techniques can be used to determine the condition of a pipeline. These tests include, but are not limited to, magnetic flux leakage data, ultrasonic testing data and thermal imaging data. Various graphical representations such as color slices and point clouds can be used to visualize this measurement data within the virtual world.

The final data format is functional data. An example of functional data is a neural network prediction. Neural networks can be designed to perform an automated classification of the measurement signal signatures. For example if an MFL scan was performed, a neural network could be used to identify the areas of benign pipeline sections versus areas that contain anomalies.

Once all of these data formats are integrated into a virtual environment the user has the ability to visually inspect not only the data obtained, but also the environment from which the data was obtained. Additionally, the user can initiate an inquiry wherein the user will be able to see the evolution of the world over time. As an example, the propagation of a stress corrosion crack within the pipe wall will be integrated into the virtual world. The integrated world with evolutionary components is extremely advantageous for performing a risk analysis or a remediation recommendation.

3.1.2 Data Processing

There are a variety of methods for data processing of complex virtual environments. Ideally, all processing techniques would occur simultaneously and instantaneously. This goal is obviously not attainable. Therefore it is important that the processing tasks be prioritized. For simplicity, the approach taken assumes all graphical, functional, and measurement data processes have occurred prior to their visualization. The graphical representations of such data can be created on the fly, but the processing of the core data occurs prior to the visualization of the world.

During analysis, the evolution of the world is a function of user input. If user input were to occur in real-time, all evolutionary predictions, models, and graphical representations would also need to occur in real-time. However, the same effects can be modeled if the user input is obtained prior to world rendering. For algorithm comparison purposes, user-input collected prior to rendering requires a significantly simpler VR implementation interface. In a final implementation the user would desire a real-time implementation, but for comparison purposes real-time implementation is not required.

3.1.3 Hardware / Software

Hardware and software choices also have a significant impact on the visualization of any virtual world. The complexity of the hardware will dictate the level of immersion that a user will experience. Although important to the user, the virtual world should be designed such that it is independent of the hardware used for visualization. Ideally the same independence should exist for the visualization software. However, certain software configuration files are needed for world visualization. To this end, the software dictates certain implementation features.

For the comparison and evaluation being performed, a semi-immersive VR system was used. The world visualizations occur in a stereoscopic, navigable environment. The software used for visualization is *vGeo*® [23]. *vGeo*® handles the graphical representations and rendering of all objects. Lighting, shading, and navigation are also controlled under this same software platform. Tracking is obtained via a flock-of-birds sensor array. Evolutionary predictions and model analysis calculations are not supported by *vGeo*® and therefore take place on a different platform prior to run-time.

3.2 Data Representation

The data representation format is an integral part of the evolutionary algorithm implementation. The evolutionary parameter represents real world phenomena that are input to the evolutionary algorithms. Subsequently, the output will be represented graphically in VR. It is therefore important to define a variable that is conducive to real world representations, input into an evolutionary algorithm, and representations in a VR environment used for both visualization and analysis.

One method in which this can be realized is through the use of a representative decision tree structure. The decision tree being used in this thesis allows for each branch of the tree to grow from a node. Nodes can represent decisions, actions, or physical features. For each time step, the branches will grow from their respective nodes a certain distance in N-dimensions. Figure 3.3 below illustrates a 2-dimensional tree structure.

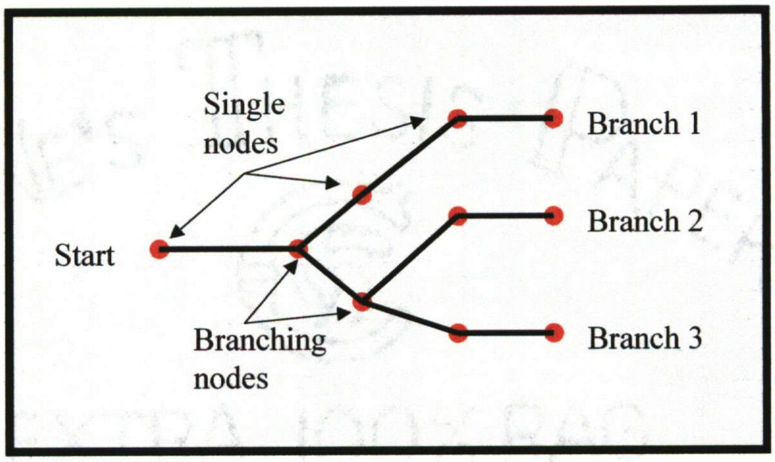
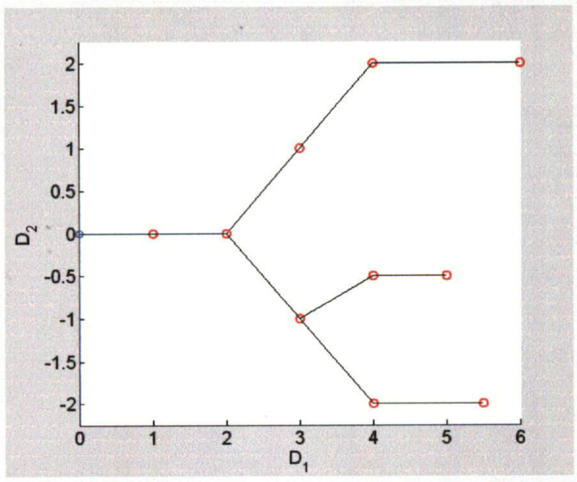


Figure 3.3 – Illustration of a decision tree variable structure.

The format of the data type is (Branches x Nodes x Dimensions). Therefore a tree with 3 branches, 5 nodes per branch, and 2 dimensions would be a 3 x 5 x 2 array. These are the dimensions of the tree displayed in Figure 3.4 below. The 2 dimensional tree on the left is described numerically by the matrix on the right. Each row of matrices in the array corresponds to 1 of the 3 branches. Each column of matrices in the array corresponds to 1 of the 5 nodes. The inner matrix (2 x 1) describes the growth rate for the given node and branch in 2 dimensions, x and y. In this thesis, growth rate is being defined as the node value; this value represents the growth (distance) from node_{t-1} to node_t.



$$\begin{bmatrix} \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}1\\1\end{bmatrix} & \begin{bmatrix}1\\1\end{bmatrix} & \begin{bmatrix}2\\0\end{bmatrix} \\ \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}1\\-1\end{bmatrix} & \begin{bmatrix}1\\.5\end{bmatrix} & \begin{bmatrix}1\\0\end{bmatrix} \\ \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}1\\0\end{bmatrix} & \begin{bmatrix}1\\-1\end{bmatrix} & \begin{bmatrix}1\\1\end{bmatrix} & \begin{bmatrix}1.5\\0\end{bmatrix} \end{bmatrix}$$

Figure 3.4 – Realization of data format.

The mapping of the tree will dictate the behavior for each node inside the virtual world. For testing each application, 1, 2, and 3 dimensional evolutionary variables will be used. For the application of modeling stress corrosion cracking in a pipeline, a mapping can be made between the tree structure and the physical crack profile. For a direct physical map, each node would represent the location of the crack at a given time-step in an x-y-z coordinate system. However, the mapping made in this thesis is used for crack growth along a pipe wall, and therefore defines 2 distinct spatial dimensions – propagation along the axis of the pipe, propagation along the circumference of the pipe. This mapping scheme makes the assumption of uniform crack depth along the length of the crack. This scheme also assumed a uniform time-step between nodes. Additionally, a third dimension could be incorporated into this structure to the represent temporal data.

3.3 Evolutionary Models

3.3.1 Algorithm Goals

The implementation of all the evolutionary algorithms follows a number of goals and assumptions. In general, the goals of the VR evolutionary algorithms focus on providing a powerful tool by which a user can perform an analysis and visually observe the corresponding changes in the virtual world. The goals for the algorithms are:

1. *Applicable over diverse datasets and components*: This includes diversity of both form and function. An example of diversity is the dimensionality capabilities of the algorithms. The algorithms are sequentially built from 1 to 3-dimensional datasets so that the deduction for an *N*-dimensional dataset is possible. Also, as described in Section 3.2 a generic model can be used such that parameterization of the model output can be formulated separately.

2. *Exhibit real-time deployment and integration capabilities*: Each algorithm should have a mechanism to allow for multiple-levels of interaction. The algorithms should actively involve the user in the necessary evolution calculations. Although desirable, real-time deployment of the algorithm need not coincide with real-time deployment of the visualization of the evolutionary model. Also, system dependencies and computational costs should be kept to a minimum.

3. *Non-phenomenological*. As an initial step, the algorithms should adhere to basic physical laws, but do not need to be phenomenological. The algorithms do not need to explicitly model the physics of the environment. The algorithms are to operate on a meta-level instead of specific physical models.

4. *Contiguous, but stochastic.* The evolutionary path should evolve smoothly; sudden jumps in the state of the world are nonphysical and therefore undesirable. Paths should be controllable, but stochastic. A user should be able to reproduce similar results in subsequent runs, but should not obtain identical results.

5. *Awareness.* Awareness is defined as the algorithm's knowledge of the environment in which it is operating. Such knowledge allows for greater complexity of the evolutionary worlds. Also, algorithm awareness makes the communication between evolutionary algorithms possible. The user is in control of the predictions and path of the models. While controlling these paths, the amount of information hidden from the user will depend on the level of awareness of the given algorithm.

6. *Easily interfaced with the user.* The parameter I/O should be intuitive and efficient. A user within a virtual world is assumed to have a limited understanding of the calculations of the evolutionary algorithm. A front-end should be built into the algorithms to allow the user to select from a limited number of adjustable parameters.

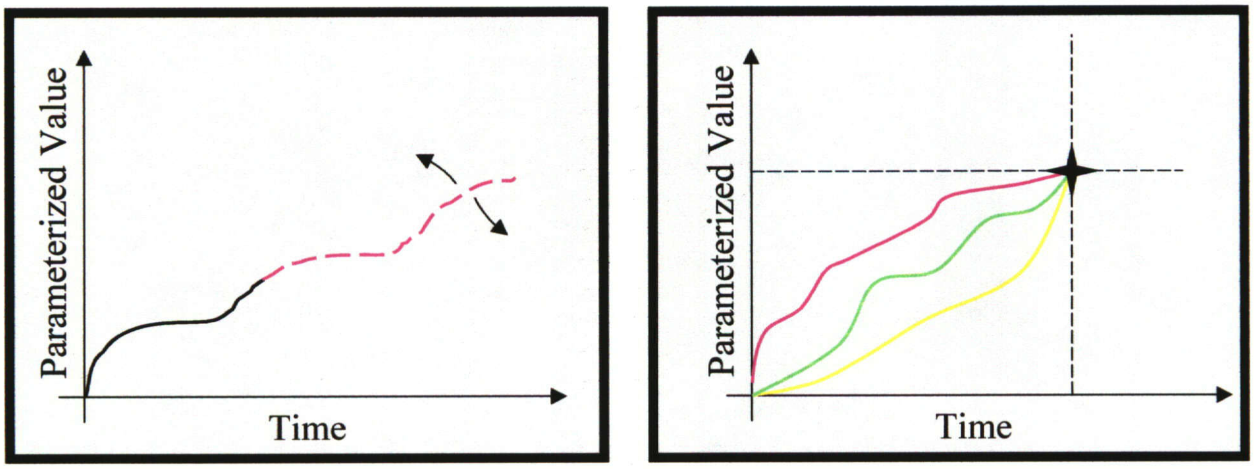
7. *Simple I/O to the VR visual rendering core.* The output of the algorithms should be in a format that can easily be translated into a visual representation. This factor is highly dependent on the evolutionary prediction format.

8. *Not application or implementation specific.* The algorithms should not hinge on a single command or implementation structure of a software platform. The translation of the algorithms from one software platform to another should not introduce a noticeable change in the performance of the algorithm. Again, this reiterates the idea that the algorithm must have a simple interface by which the predictions can be translated into a

visual representation. The visual representation should be adjustable depending on the requirements of the display software.

3.3.2 High-Level Structure

In this thesis, evolution is defined as a process by which a user inquiry initiates a corresponding temporal change within a virtual world. Using this definition, evolution can be subdivided into two different categories – single-path evolution and multi-path evolution. Single-path evolution allows the user to evaluate a solitary possible path as it changes over time. During the evolutionary process the user can interact with variable data and influence the direction and nature of the evolution. In multi-path evolution a user would define a desired value and / or a duration in time. The evolutionary algorithm would then generate possible data paths to illustrate different means by which these desired values could be obtained. The figure below illustrates the concept of single and multi-path evolution.



(a) Single path evolution. **(b) Multi-path evolution.**

Figure 3.5 – Evolution.

In order to implement and perform an unbiased evaluation of various evolutionary algorithms a high-level structure is needed. The high-level structure allows for uniformity among the algorithms with respect to the process of generating the desired predictions. Additionally, a high-level structure allows for a simple interface between the user and each of the evolutionary modules. There are two core functions common in both single and multi-path evolutions –prediction and correction. For this thesis, prediction is defined as the process by which calculations for the output of evolutionary nodes are made. Equation set 3.1 below illustrates the outcome from calculating a prediction.

$$\begin{aligned}&[y_0 \;\; y_1 \;\; y_2 \;\; y_3\;] = [f_A(x_0) \;\; f_A(x_1) \;\; f_A(x_2) \;\; f_A(x_3)\;] \\ &\{\text{Calculate Prediction}\} \\ &y_4 = f_A(x_4)\end{aligned} \tag{3.1}$$

where f_A is the function that describes the nature of the model, and x_n is the model input. Equation set 3.1 illustrates how a prediction would be calculated at time-step 4. The outputs $y_{0\text{-}3}$ are calculated in the same manner. Each evolutionary algorithm will have a unique method of calculating the prediction, but all of the algorithms will have the output similar to Equation set 3.1. The second function, correction, is the process by which the evolutionary model is updated. This is illustrated in Equation set 3.2.

$$\begin{aligned}&y_4 = f_A(x_4) \\ &\{\text{Parameter Correction}\} \\ &[y_5 \;\; y_6 \;\; y_7 \;\; y_8\;] = [f_B(x_5) \;\; f_B(x_6) \;\; f_B(x_7) \;\; f_B(x_8)\;]\end{aligned} \tag{3.2}$$

These equations illustrate how model f_A is updated to model f_B. Subsequent model calculations for inputs $x_{5\text{-}8}$ use the new function f_B. Again, each evolutionary algorithm will have a unique method for performing a correction. By using a combination of both functions, single and multi-path evolution can be implemented without concern for the manner in which the prediction and correction are performed.

3.3.2.1 Single-path Evolution

Single path evolution allows the user to initiate a prediction based on the dataset at a given instant. The user is given an evolution adjustment parameter, α, to control the prediction path. Additionally, the user has the ability to either generate a new prediction or update the prediction model, correction. The flowchart illustrated in Figure 3.6 provides a conceptual model of the implementation for high-level single-path evolution. The overall structure is independent of the prediction and correction model implementations.

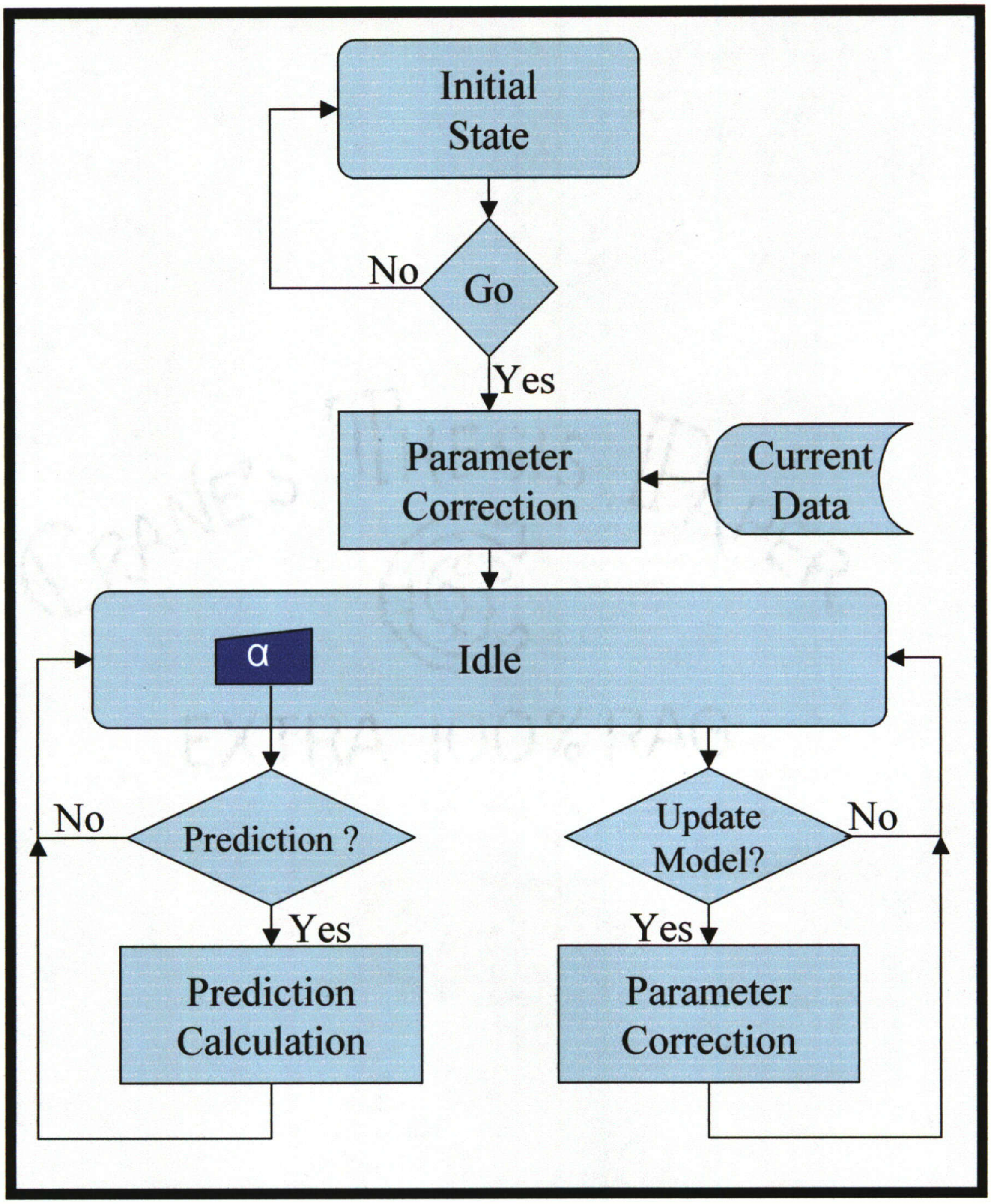


Figure 3.6 – Single-path evolution algorithm.

3.3.2.2 Multi-path Evolution

In this thesis, multi-path evolution allows the user to specify both a desired parameter value and a desired duration in time, only a desired parameter value, or only a desired duration in time. The model has tolerance parameters that can be adjusted to determine the acceptable difference between the predicted values and the desired values. In this thesis, fully constrained evolution is defined by the user specification of both parameter and time values. Alternately a user can input only a value or only a time; these are value-constrained evolution and time-constrained evolution respectively. The equation set below, 3.3, defines the three types of evolutionary constraints.

$$\begin{array}{l} \text{Fully} - \text{Constrained} \\ f_n(t_c) = f_c \\ \text{Value} - \text{Constrained} \\ f_n(t) = f_c \\ \text{Time} - \text{Constrained} \\ f_n(tc) = f \end{array} \qquad (3.3)$$

Where f_c and t_c are the value and time constrains, and f and t are a value and time at an arbitrary state. Like single-path evolution, multi-path evolution can also be implemented using the prediction and correction functions. Figure 3.6 illustrates the dataflow model for the multi-path evolutionary algorithm. First, the user initiates a multi-path evolution and defines the goal. The goal consists of a desired value and/or time constraint along with the number of paths to be generated. From this information, the high-level algorithm corrects for a set of parameters. Predictions are generated to construct a path. After each prediction the overall path is checked. If the goal was reached, the path is saved. If no additional paths are required, the algorithm is finished. However if additional paths are required, a new set of parameters is obtained. If the goal is not reached to within the predefined tolerances, the path is evaluated against the goal.

The comparison focuses on determining whether or not it is still possible for the path to reach the goal. If it is determined for the given parameters the goal will never be reached, the parameters are corrected in an attempt to gain more suitable parameters.

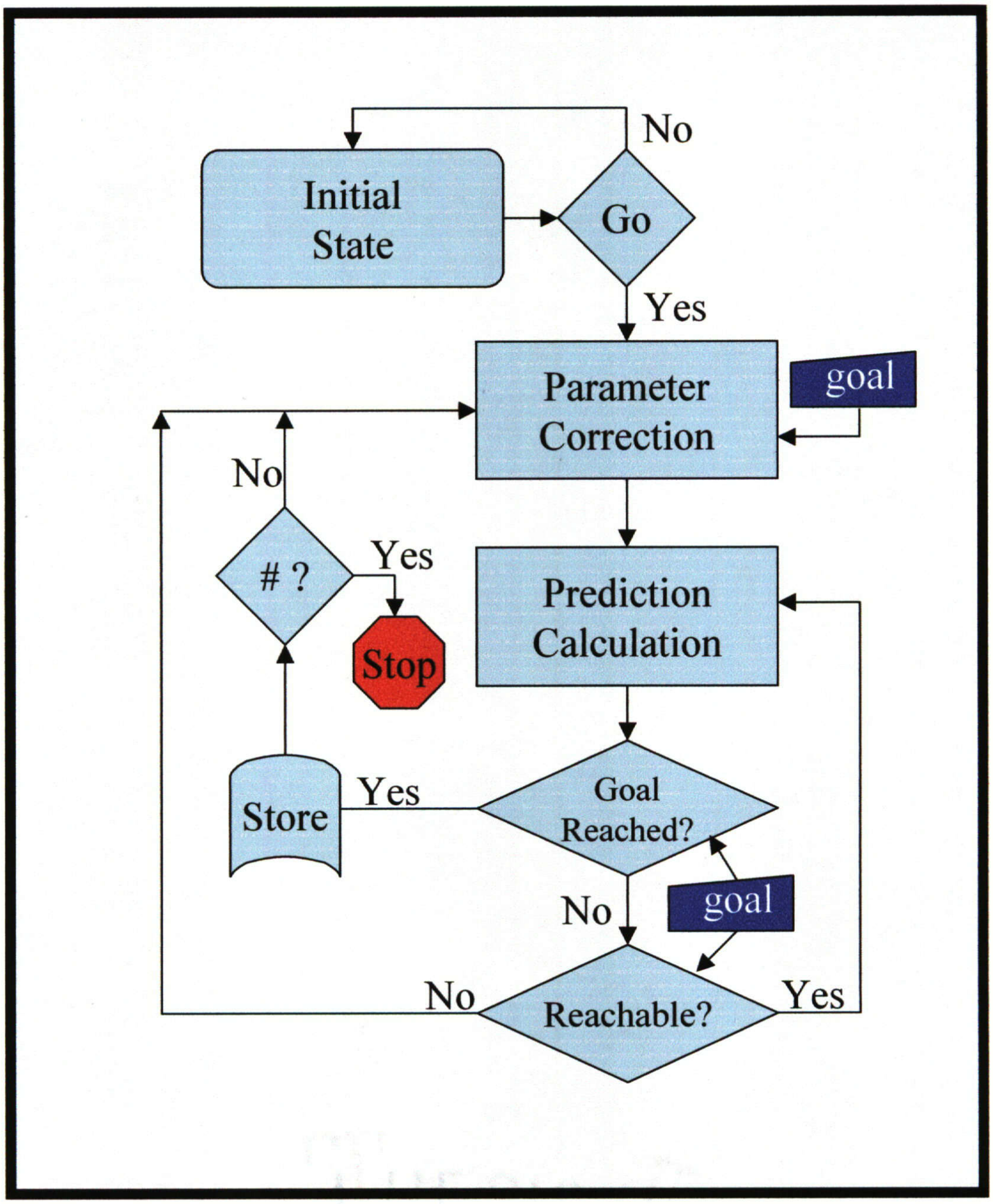


Figure 3.7 – Multi-path evolution algorithm.

The dimensionality of the algorithm is determined by the dimensionality of the dataset and the individual modules for prediction and correction. The high-level algorithm is independent of dimension so it will remain constant even as the dimensionality of the modules is altered.

3.3.3 Evolutionary Algorithms

3.3.3.1 Deterministic Models

The deterministic representation in this thesis is modeled using known growth rates. Essentially, a deterministic implementation will resolve to the calculation of a desired parameter. In order to simply the process, the model implementation did not use first principles, but rather assumed that the results of the required rate calculations were known. The figure below illustrates how the prediction and parameter correction functions are utilized.

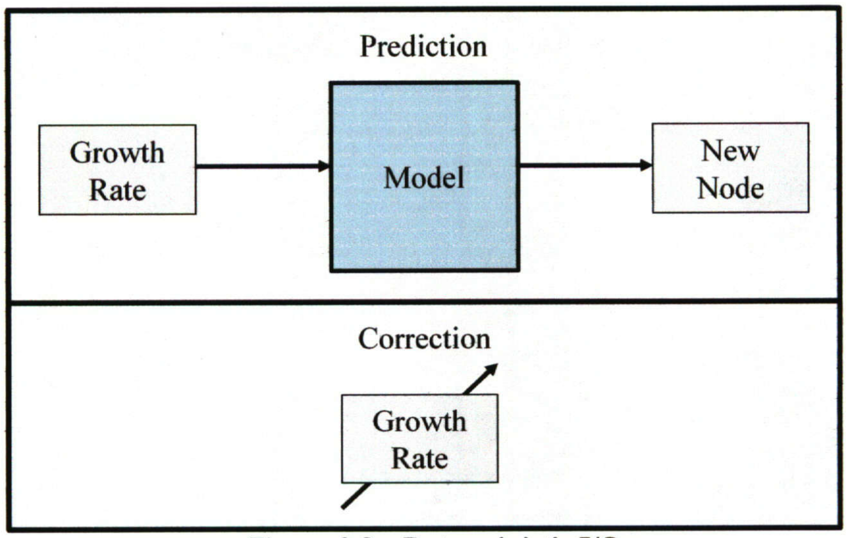


Figure 3.8 – Deterministic I/O.

The input of the prediction function is the known growth rate of the evolutionary parameter. The deterministic model translates this growth rate directly into a single new node. The parameter correction function requires that the growth rate be changed. The growth rates were obtained from a priori information about the nature of crack growths. Figure 3.9 below illustrates a typical trend of stress corrosion cracking growth versus the stress intensity factor.

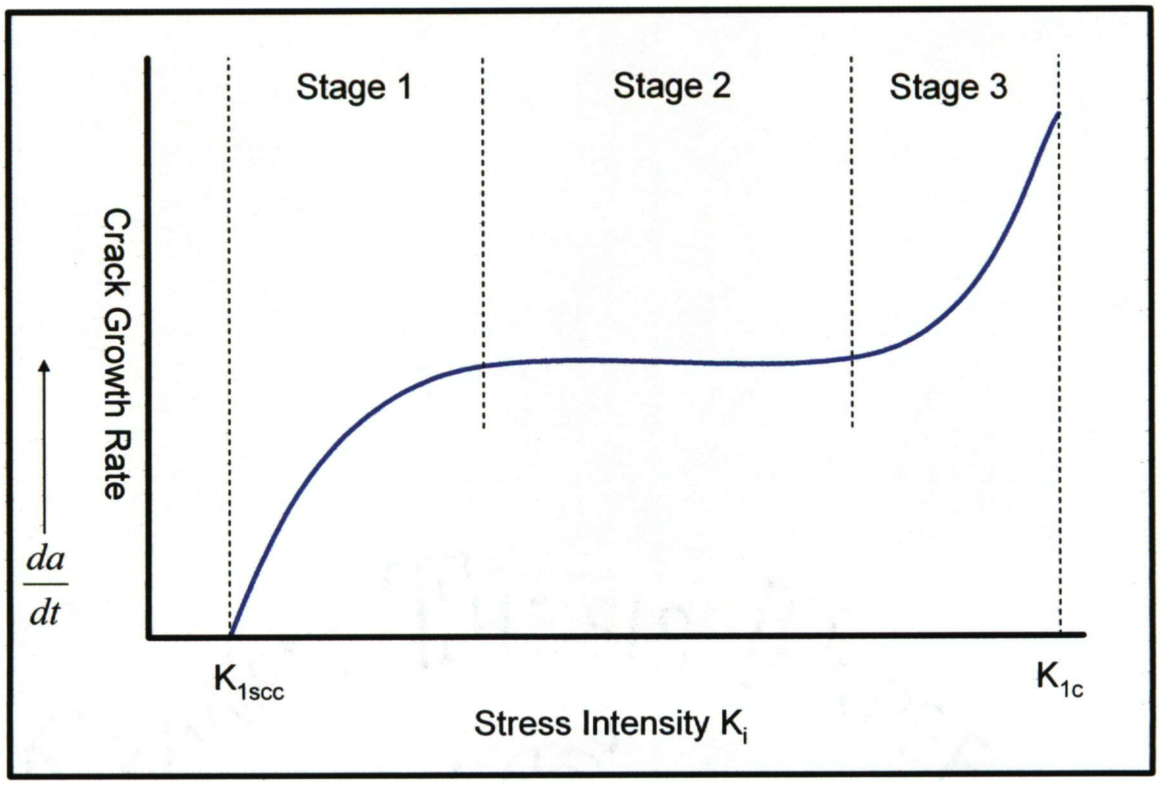
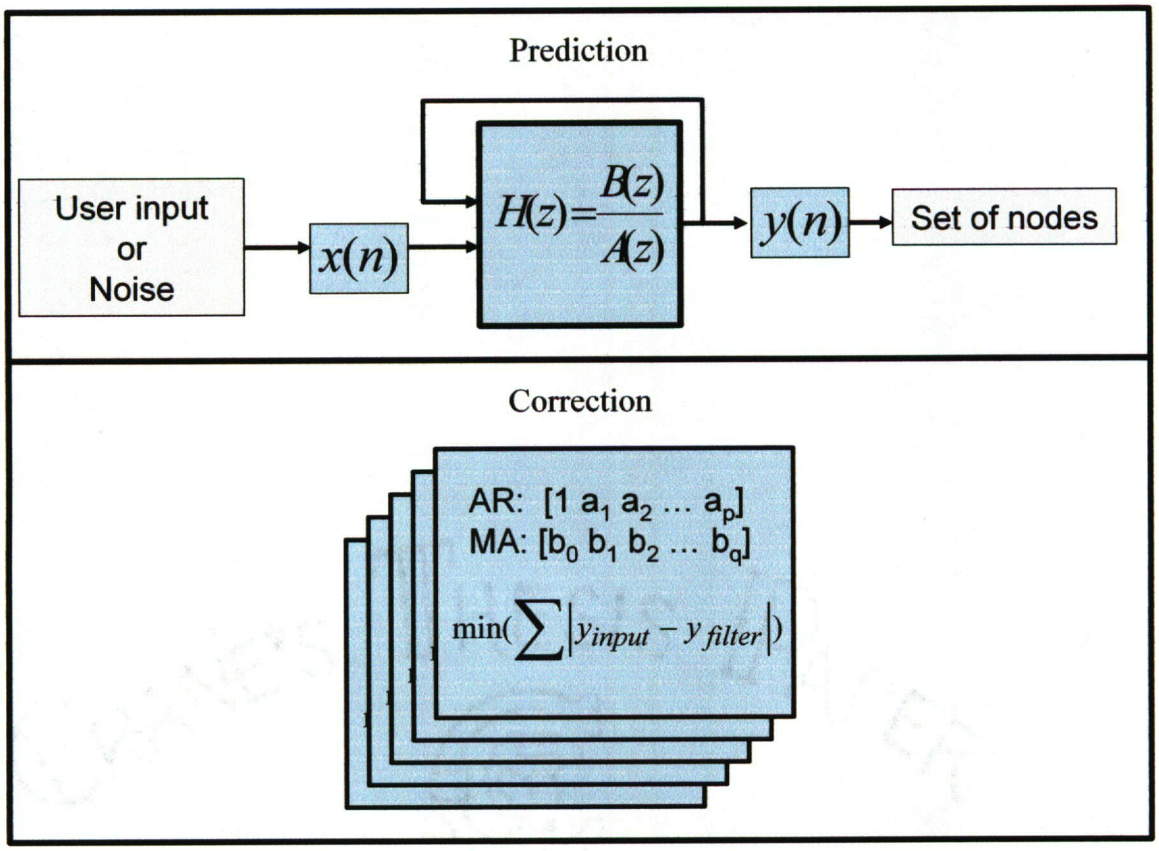


Figure 3.9 – Crack Growth Chart.

The stress intensity factor describes the concentration of stress at the tip of the crack. There are typically three stages within the cycle of crack growth. Stage 1 begins when the stress intensity reaches a threshold, K_{1scc}. In stage 2, the crack growth is independent of the intensity factor; the rate of crack growth is determined from environmental factors such as temperature and environmental corrosives. Once the intensity has reached a high enough level, the growth enters the third state which is characterized by a rapid increase in the growth rate. Failure occurs at the material's fracture toughness, K_{1c}.

3.3.3.2 Autoregressive Moving Average Models

The second evolutionary module investigated was an IIR filter model, specifically an ARMA model. The figure below illustrates how the ARMA model was used to implement the prediction and correction functions.



Prediction
User input or Noise
$x(n)$
$H(z)=\frac{B(z)}{A(z)}$
$y(n)$
Set of nodes
Correction
AR: [1 a_1 a_2 ... a_p]
MA: [b_0 b_1 b_2 ... b_q]
$\min(\sum|y_{input} - y_{filter}|)$

Figure 3.10 – ARMA I/O

The prediction function utilizes the natural I/O structure of the ARMA model for use within the evolutionary environment. The input of the model is either a user input or random noise. The use of user input as the *x(n)* is more accurately referred to as an ARX model. The system uses the input signal and filter coefficients to obtain an output *y(n)*, the *y(n)* represents a set of nodes for the evolutionary parameter.

The correction function is responsible for calculating the filter taps. The Steiglitz-McBride iteration was used as a means of calculating the desired coefficients. This technique uses a combination of different methods. The system identification or parametric modeling first uses Prony's method; Prony's method is a sequential technique, in that it first finds one set of coefficients (poles) and then calculates the second set (zeros). The filter is not necessarily stable, but it can find the coefficients exactly if the signal being modeled is truly an ARMA signal of proper order.

In order to calculate the coefficients, Prony's method uses a variation of the covariance method of AR modeling. The overall steps for calculating the coefficients are described below [33]:

- Perform an AR fit using a variation of the covariance method
- Filter the roots
- Use a least squares fit
- Use signal thresholding to create the final model

This method is equivalent to Shank's method in that it computes the denominator coefficients by minimizing the equation below:

$$J_S^2(\hat{\theta}) = \sum_{n=n_b+1}^{\infty} (\hat{a} * h(n) - \hat{b}(n))^2 \tag{3.4}$$

The equation represents finding the poles by ignoring, or 'skipping', the zeros that are present in the impulse response. This approach is more commonly known as a modified Yule-Walker method. Since the poles are known, the remaining coefficients are found by minimizing equation 3.5

$$\left\| H(e^{j\omega}) - \frac{\hat{B}(e^{j\omega})}{\hat{A}*(e^{j\omega})} \right\| \tag{3.5}$$

The Steiglitz-McBride iteration is different from Prony's method in that it is an iterative approach for calculating the system coefficients. The method provides a fast iterative technique for finding the numerator and denominator coefficients simultaneously. The algorithm also attempts to minimize the mean squared error between the known output and the predicted output. The Steiglitz-McBride iteration is described below [33]:

- Pre-filters h and x using $1/a(z)$
- Solves a system of linear equations to solve for a and b

Again, the algorithm attempt to minimize the function described in equation 2.6

$$\min_{a,b} \sum_{i=0}^{\infty} |x(i) - h(i)|^2 \tag{3.6}$$

Like the Prony method, the Steiglitz-McBride iteration provides a direct fitting approach. Due to this approach, the filter is not guaranteed to be stable. The Steiglitz-McBride iteration can be run for any number of iterations in an attempt to more closely match the given set of coefficients. The Steiglitz-McBride algorithm uses the Prony method as an initial estimation. The various implementation parameters – number of poles, number of zeros, and the number of iterations for the algorithm – are all adjustable design parameters. These design parameters are not altered by the correction function; the correction function only updates the filter taps.

3.3.3.3 Genetic Algorithm Models

Genetic algorithm models were also considered as an evolutionary prediction module. Genetic algorithms have a great deal of parameters that must be defined in order to obtain the desired functionality. The entire population of individuals within the genetic algorithm represents multiple solution sets. When a given individual is chosen, the chromosome is decoded via a mapping function, and a set of node values for the evolutionary parameter is obtained. The prediction function generates solutions by evolving the entire population and extracting a representative individual. The correction function keeps the genetic algorithm design parameters constant, but updates a set of user defined values. These concepts are depicted in Figure 3.11 below.

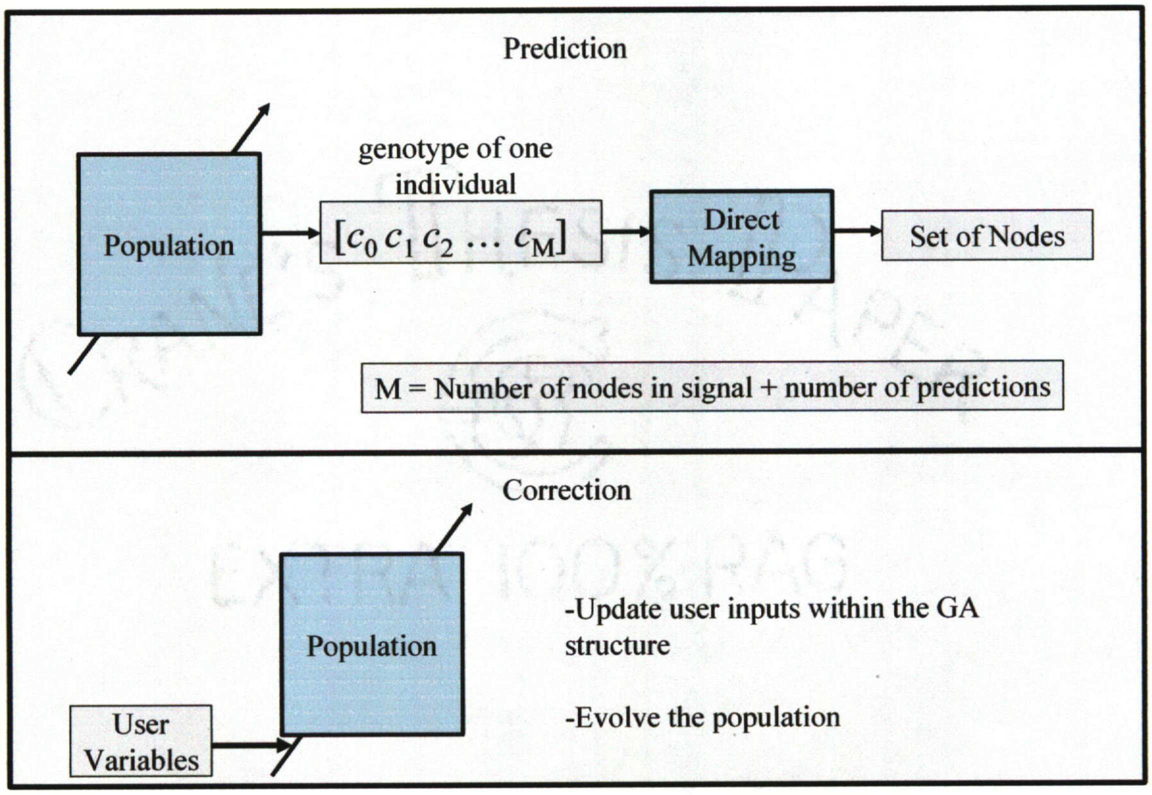

Prediction
Population
genotype of one individual
$[c_0\, c_1\, c_2 \ldots c_M]$
Direct Mapping
Set of Nodes
M = Number of nodes in signal + number of predictions
Correction
Population
-Update user inputs within the GA structure
-Evolve the population
User Variables

Figure 3.11 – Genetic algorithm I/O

The overall process of genetic evolution is described in Section 2.2.3. As seen in the description, genetic algorithms have a great deal of variability to them. The ensuing paragraphs will outline the methods that were implemented in this thesis to realize the genetic algorithm structure. The first step in setting up a genetic algorithm is to determine the genotype to phenotype mapping. Since the evolutionary parameter is already a coded version of the solution, the evolutionary parameter format was used directly as the input to the genetic algorithms, the genotype. A single individual is a set of nodal values; these node values include known growth rates as well as the predictions. The solution set, the entire population, is therefore multiple sets of nodal values as seen in the equations below

$$\begin{array}{ccccc} c_0^1 & c_1^1 & c_2^1 & \dots & c_M^1 \\ c_0^2 & c_1^2 & c_2^2 & \dots & c_M^2 \\ c_0^3 & c_1^3 & c_2^3 & \dots & c_M^3 \\ \vdots & & & & \vdots \\ c_0^N & c_1^N & c_2^N & \dots & c_M^N \end{array} \tag{3.7}$$

where the superscript denotes the individual (1 to N), and the subscript represents the chromosome number (0 to M). In the case of multiple dimensions and multiple branches, c_j^i will be in matrix form. When initializing the algorithm, the population is randomly generated. Each chromosome is generated using a uniform distribution for a discretized set of values between a prescribed upper and lower bound. Random generation for the initial population ensures diversity among the solution sets.

The next step in a genetic algorithm is the evaluation of the population. The evaluation function is based on a Gaussian fitness curve. A Gaussian curve was used over a Euclidean distance estimation because the use of a Gaussian allows for more degrees of freedom within the evaluation. Also, in an attempt to more closely match a natural solution set, it was desirable for the solution set to exhibit a Gaussian distribution of the individuals. The process for evaluating a single individual, ith, is given below

$$Eval([c_0^i \quad c_1^i \quad c_2^i ... c_M^i]) = f^i = \sum_{j=0}^{M} f_j^i \tag{3.8}$$

The fitness of each individual is the sum of the evaluation of each chromosome of that given individual. For a given chromosome, the user can define the mean and variation of the Gaussian curve. Typically, the mean is equal to a known value of an a priori signal. The fitness score is then based on the value the solution exhibits along the Gaussian member function with the user-given mean and standard deviation. An example of this process for the *ith* individual is illustrated in Figure 3.12.

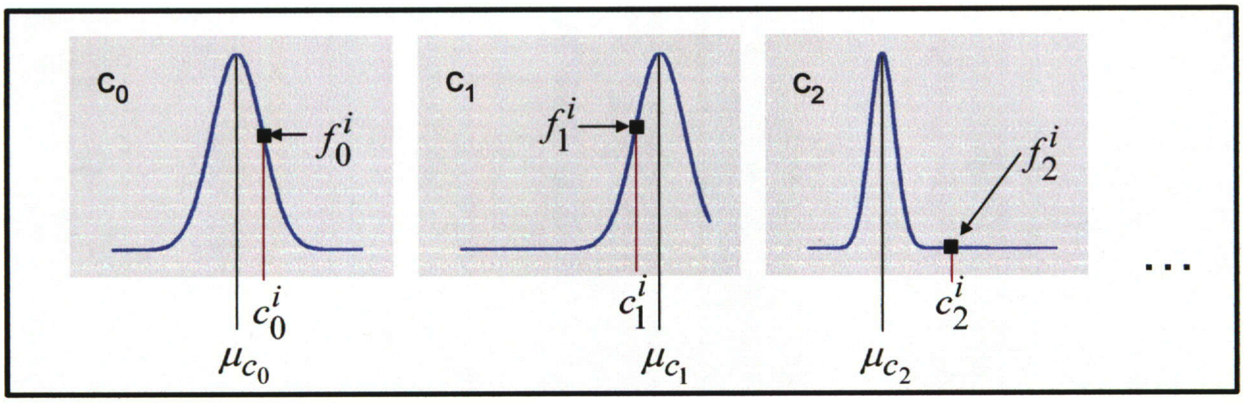


Figure 3.12 – Gaussian fitness evaluation

A roulette wheel parent selection technique is then used to select the individuals that will mate. The probability that the ith individual will be chosen is given by Equation 3.9

$$P(i) = \frac{f_i}{\sum_{j=1}^{N} f_i} \tag{3.9}$$

After the individuals are selected for mating, a random one-point cross-over per dimension is then used to combine the individuals to create the children.

A bit-wise mutation function was implemented. The defined mutation rate is applied to each chromosome of each newly generated individual. If selected for mutation, the given chromosome is replaced by a random value within the predefined lower and upper bounds. The equations below describe a mutation for chromosome with index value 1 of the ith individual.

$$M([c_0^i \quad c_1^i \ldots c_M^i]) = [c_0^i \quad f(c_1^i) \ldots c_M^i] \tag{3.10}$$

$$f(c_1^i) = rand$$

After being subjected to the mutation function, the new individuals are inserted into the new population. Since a replace all strategy is implemented, *N/2* matings are performed to create *N* new individuals. These *N* individuals create the entire population at the subsequent time step. Once the population has evolved over several generations, a representative sample needs to be output. This representative individual will represent a set of nodes that can be used as output of the evolutionary function. In order to select only 1 of *N* individuals, the same process used for mate selection is employed. However, instead of selecting 2 mates, only 1 representative individual is selected.

The parameter correction function requires changing only a small number of the total parameters used within the genetic algorithms. The majority of the parameters are kept consistent to assure that the results are reproducible over a number of runs. The main change that can be made via the parameter correction function is an update of the user input; only the Gaussian fitness evaluation parameters are altered. Also, the user has the ability to select the number of iterations of the overall evolution. By selecting a low number, the population will have a large amount of random individuals, whereas with a high generation number, the individuals will exhibit a lower amount of randomness and a higher correlation to the known inputs.

3.3.3.4 Hidden Markov Models

Hidden Markov Models are the final module being investigated. For this model, each dimension will have individual visible-state probabilities. Transitional and visible-state probabilities will be determined prior to run-time, but the visible state transitional probabilities can be altered using the parameter correction function. As an initial assumption, this thesis assumes that the transitional probabilities will be based on the structure of the HMM, not as a fit from acquired data. The model developed uses three hidden states and three visible states. Figure 3.13 below depicts this model.

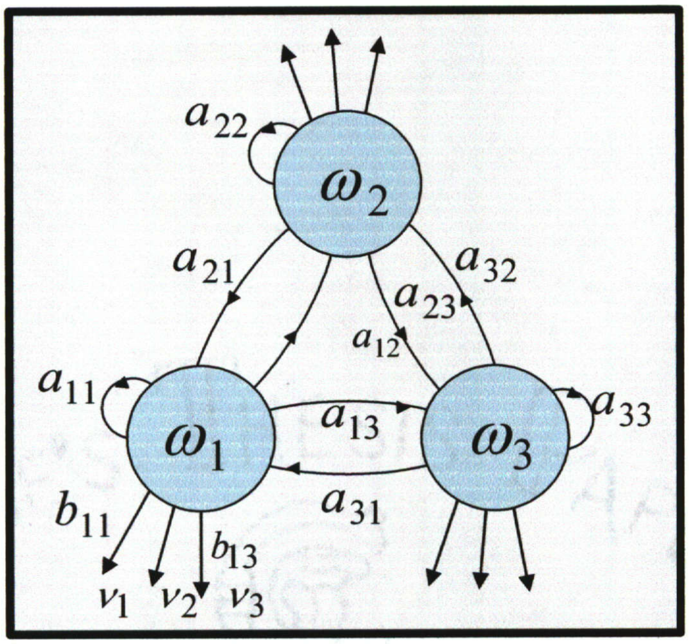


Figure 3. 13 – HMM implementation structure.

This model assumes that the largest growth rates occur for the low hidden state values. I.e. all visible states from hidden state 1 will exhibit higher nodal values then the visible states from hidden state 2. Within a given hidden state, the higher visible states will exhibit higher nodal values. This means that hidden state 2, visible state 3 will have higher nodal values then hidden state 2, visible state 2. This concept is illustrates in the prediction function description of Figure 3.14.

The prediction function uses the transitional probabilities input into the model to generate visible states; each visible state corresponds to a single nodal value. Therefore, a set of visible states will correspond to a set of nodal values, the evolutionary parameter. The parameter correction function allows the user to alter the transitional probabilities of the visible states. The hidden state transitional probabilities remain unaltered. The concept for the prediction and correction functions are illustrated in the figure below.

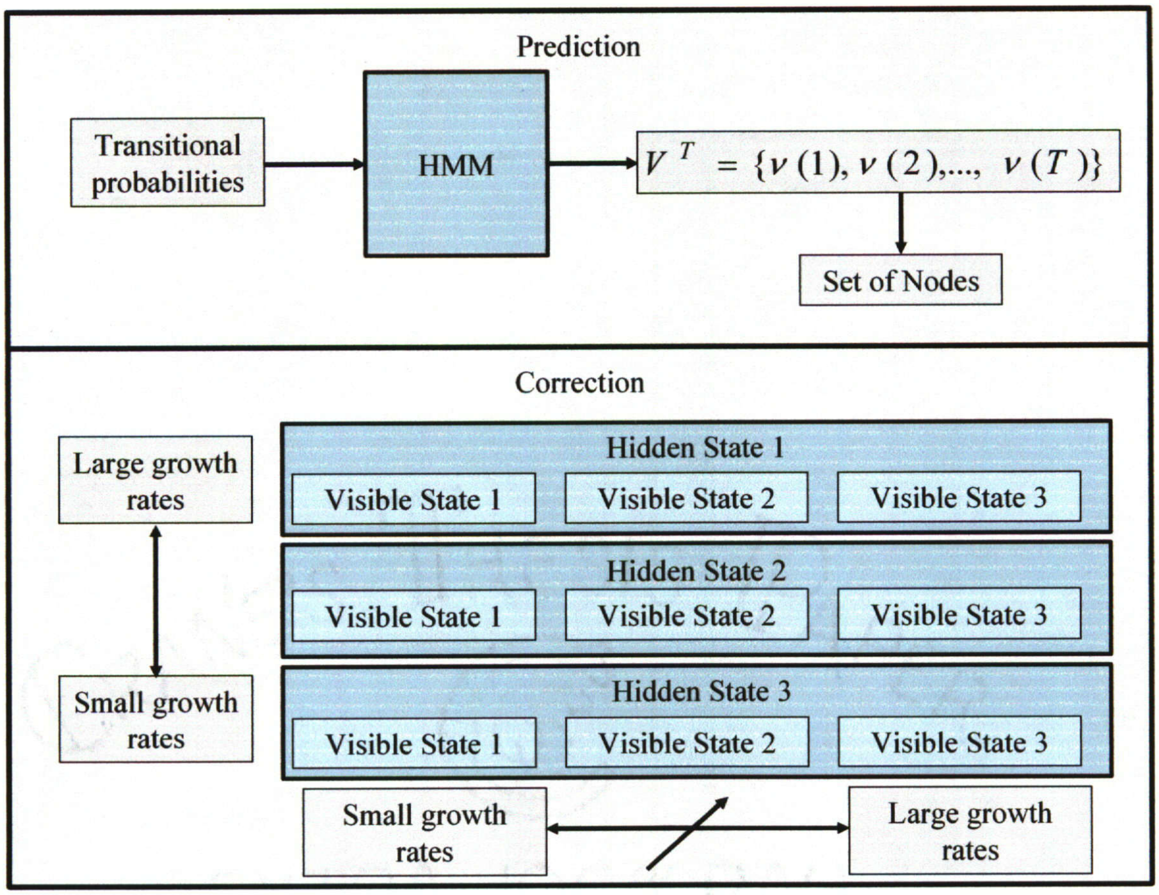


Figure 3.14 – HMM I/O.

3.4 Evolutionary Visualizations

The evolutionary visualizations will take place in the same virtual worlds as the multi-sensor data fusion. In addition to the multi-sensor data, the evolutionary worlds will display the evolutionary predictions and the algorithm inputs and outputs. The inputs and outputs will be displayed as 3-dimensional slider bars. Both sets of I/O will be color coded and labeled according to functionality. The I/O displays will be fixed on the screen; the coordinate system will be static and non-navigable. Fixing the displays will allow them to be visible regardless of the orientation of the world itself.

3.5 Evaluation Criteria

The goal in comparing the four algorithms is not to evaluate the algorithms for predictive coding applications, but to determine the benefits and drawbacks of each of the algorithms for VR evolutionary scenario development. Each algorithm has traditional positive and negative features based on previous work (Section 2.2). The objective of the evaluation is to further the understanding of these features when the algorithms are implemented for VR development.

In order to ensure a fair comparison, the algorithm modules were tested using both the single path and the multi-path high-level implementation algorithm. Each of the four prediction algorithms was used as a prediction and correction module within this overall structure. For both single and multi-path evolution, each algorithm will operate in 1, 2, and 3 dimensions. From these implementations, an impartial comparative analysis can be performed.

There are two main divisions for the evaluation of the evolutionary models – setup and runtime. The setup evaluation criteria take into consideration the implementation specifics that need to be set before the algorithm is run. When dealing in an immersive, navigable, and interactive environment there are certain variables that need to be setup to allow for a favorable analysis in VR. Versatility and model I/O are the two implementation criteria being investigated.

The runtime evaluation criteria take into consideration the factors that affect the functionality of the algorithm during calculations. Since VR is the platform of consideration, the evaluation will also analyze the runtime performance. The runtime evaluation criteria are, speed, memory usage, algorithms' interaction with the user (ease of use, and usefulness), and software specific considerations. Figure 3.15 illustrates the relationship between the algorithm goals, the evolutionary prediction variable, and the evaluation criteria. The areas shaded in yellow illustrate which goal(s) each of the evaluation criteria is targeting. The following sections, 3.5.1 – 3.5.6, detail each of the evaluation criteria.

	Implementation Characteristics			Run-Time Characteristics			
Goals	Evolutionary Prediction	Versatility	Model I/O Framework	Speed	Memory	Interaction with User	Software Specific
Applicable over diverse data	🟨	🟨					
Real-Time deployment			🟨	🟨	🟨	🟨	
Non-Phenomenological	🟨						
Contiguous							
Awareness			🟨				
Interaction with user	🟨		🟨			🟨	
I/O to VR core	🟨						
Software specific				🟨			🟨

Figure 3.15 – Relation between algorithm goals and evaluation criteria.

3.5.1 Versatility

In this thesis, versatility is defined as the ability of the algorithm to adjust to various types of VR evolutionary prediction formats. The versatility criterion is a measure of the algorithms' abilities to input and output to various evolutionary formats. Versatility is being measured by analyzing the changes required in the implementation if the format of the evolutionary variable changed.

Versatility is important in VR simulations because there is a direct relation between the evolutionary parameter and the required visualization. In order to maintain versatility in the visualizations, versatility in the evolutionary parameter must also be maintained. Secondly, optimization of visualizations may require small changes in the evolutionary parameters. Small changes in the evolutionary parameters should not necessitate large changes in the algorithms' implementations. Finally, versatility of the evolutionary parameter will allow for a variety of scenarios to be analyzed.

3.5.2 Model I/O Framework

In this analysis, the model I/O framework takes into account the ease of implementing the required inputs and outputs from the algorithms. The evolutionary parameter input, specifically, is excluded from this analysis since the implementation concerns for that particular variable are addressed in *Versatility*, Section 3.5.1. The model I/O framework takes into consideration the multi-level nature of the implementation required for interaction with the user.

Model I/O framework is being measured by analyzing the required changes in the implementation if a new input or output feature is to be added. For example, what are the implementation concerns to add a new variable that allows the user to control the amount of randomness within a given prediction? The model I/O framework is important to VR applications because it reflects the algorithms' abilities to interact within a larger environment. This ability is critical because VR risk analysis is not a stand-alone application, but rather one piece of a larger model.

3.5.3 Speed

Speed is a run-time characteristic that has two components for this analysis – the measure of the time required to calculate an evolutionary prediction and the time required to perform the parameter correction function. Specific timing runs are not used for the evaluation because they rely heavily on the specific implementation of the algorithm. Instead the speed comparison is performed with respect to a typical implementation.

Speed is an important variable in most algorithm analysis applications, especially VR implementations. Since there is substantial calculation overhead for VR display systems it is important that the evolutionary algorithms running in the background do not hinder the other system components. Also, it is desirable to the user that there not be a noticeable delay after an evolutionary inquiry. The ability of the system to quickly respond to a user command is essential.

3.5.4 Memory

Memory is another run-time characteristic. The memory requirements are important because, like speed, they affect the overall world performance. The memory evaluation criterion is a measure of the number and size of the variables needed in order for an algorithm to perform the required evolution. Like speed, the memory requirements fluctuate according to the specific implementation used. Therefore the analysis is performed with respect to the overhead required for a typical implementation of a given algorithm. Since processing power and memory are critical to VR applications memory is an important run-time characteristic to take into consideration.

3.5.5 Interaction with User

For this analysis, the interaction of the various algorithms with the user during runtime is measured by two separate criteria. The first criterion is the ease by which the user can interact with the system. This includes all the abilities a user could be given in order to support the evolutionary process. The second factor represents the usefulness of that interaction. The second factor can also be described as the magnitude of the effect that can be gained through the interaction.

User interaction is extremely important for VR analysis. One of the key features of a VR environment is the ability of the user to interact with the data. Without interaction, the analysis would be static and offer little additional benefits to the user over traditional methods. In order to accomplish a fair analysis, all four of the algorithms were implemented into the same hierarchical structure. It is this structure that handles the user interaction directly.

3.5.6 Software Specific

For this analysis, the software specific criterion is considered the ability to perform the described algorithms on a variety of platforms. An algorithm that has strong ties to a given platform is given a low rating. Ties can be defined by both functionality and performance. Often, algorithms use built-in functions tied to a specific software platform. When the algorithm is re-implemented into another platform the integrity of the algorithm may be sacrificed if the required built-in function is not present. During the implementation of the four types of models, careful consideration was spent on limiting such functions.

CHAPTER 4 – RESULTS

4.1 Overall Visualizations

The first step in creating the VR world is to model the physical features within the world. For the NDE analysis of gas transmission pipelines, the components that need to be modeled are the pipeline components, including pipe sections, flange, weld, sleeve, T-section, taps, check valve, ball valve, anchor, and anomalies. Each component was modeled separately using SolidWorks®. A script was written to automate the assembly of the component pieces. The goal is to assemble the pipeline components such that the virtual model matches the actual pipeline network as closely as possible. One such pipeline network is seen below in Figure 4.1.

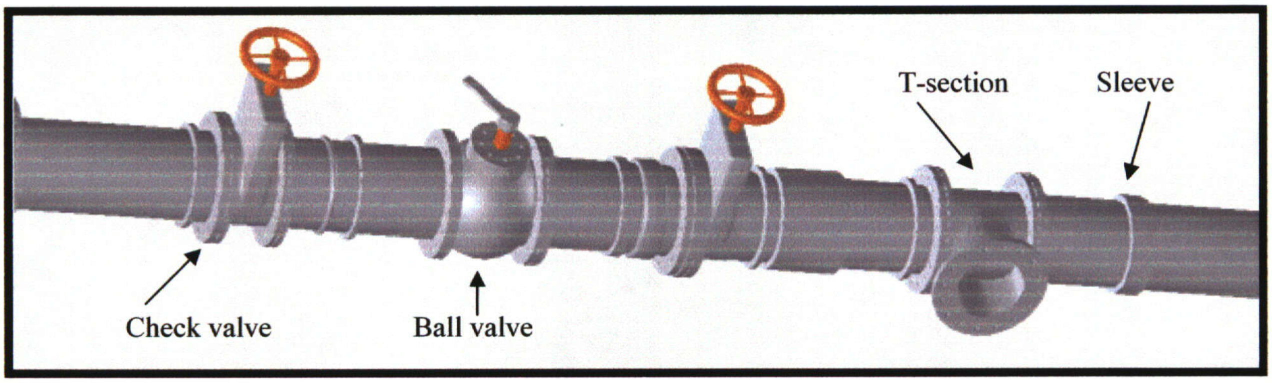


Figure 4.1 – Graphical representation of a pipeline section.

Figure 4.1 illustrates various components combined together to form a representation of a subsection of an entire pipeline network. Each piece was built such that it had suitable detail for realistic viewing. However, graphical data should be represented using the lowest amount of computational intensity possible. Therefore, multiple iterations of the design process were performed in order to minimize component complexity while preserving model realism. During the first set of changes, the overall file size of each component was reduced by approximately 50%, with relatively no loss of visible detail.

Next, the measurement data was obtained. The condition of the pipeline was inspected using a magnetic flux pig. The MFL pig measured the flux inside of the pipe with 83 circumferential sensors. A circumferential grid was constructed to correspond to sensors' locations on the pig. At each data point the base value of the grid was scaled according to the reading obtained at that data point, where large values extend further from the original grid. Figure 4.2 below illustrates a section of the flux data. The upper portion of the figure illustrates a portion of a pipe-scan being viewed from outside the pipe. The zoomed in section of the pipeline shows how a user inside of the pipeline sees the flux data.

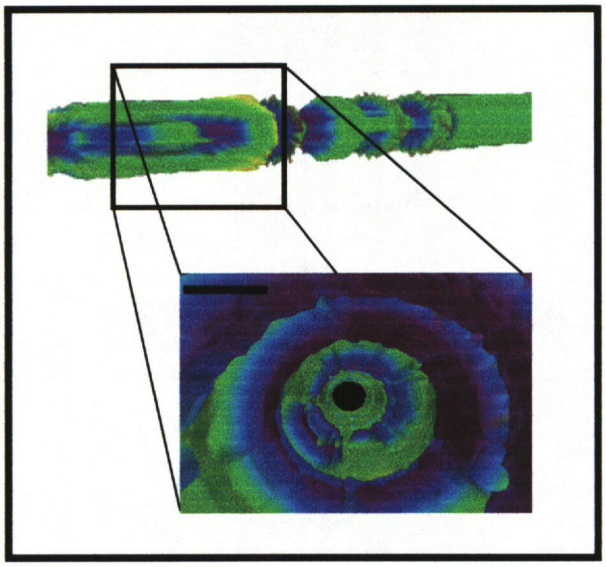


Figure 4.2 – Measurement data from the inspection of a pipeline section.

One set of functional data associated with this world is a simulated neural network classification of the flux surface. The goal of the classification is to represent each of the pipeline components as a distinct class. This functional data is represented using a custom color table, where a unique color is assigned to each component / class. Next, the classification data can be integrated into the same world as the graphical and measurement data. An opaque color-slice is created for viewing internally to the pipe. As the user navigates through the center of the pipeline, the component prediction is viewable. The color-slice is registered concentric to the translucent graphical representation of the actual pipeline components. The three-dimensional translucent representation of the pipeline components allows the user to look through the pipeline to see data both internal and external to the pipeline itself. Finally, the magnetic flux data, using the distorted grid, is visualized with an opaque pseudo-color mapping. This world is displayed in Figure 4.3 below.

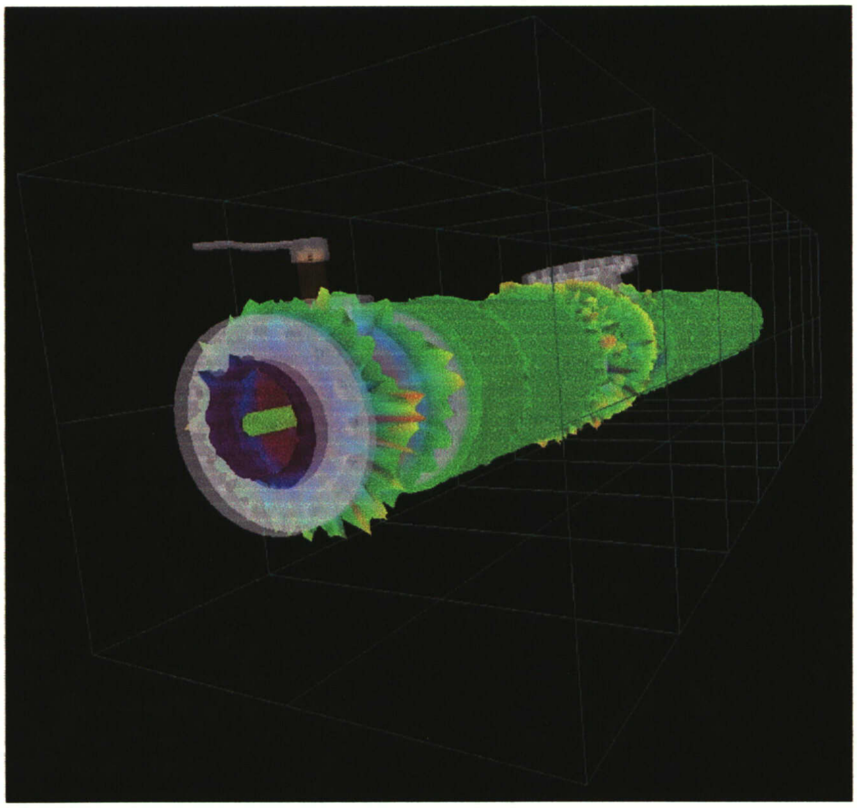


Figure 4.3 – Integration of graphical, functional, and measurement data.

Figure 4.3 illustrates a world in which graphical, measurement, and functional data are visually integrated such that a VR user could carefully analyze the associated pipeline. A similar world was built from the same data files, but using a different set of visual representations. This world is represented in Figure 4.4. The graphical data in this world is represented with an opaque model. The neural-net classifications are displayed externally using a translucent color slice. Both worlds display the same pipeline data, but do so in different ways.

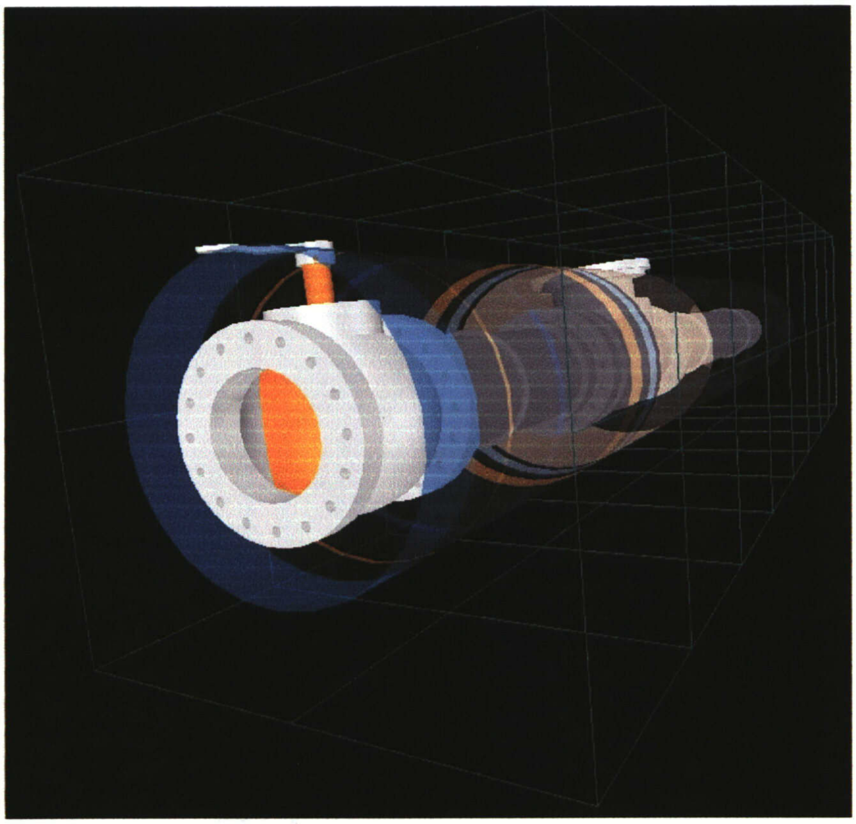


Figure 4.4 – Integration of graphical and functional data.

Both of those worlds, Figure 4.3 and 4.4, are lacking references to the larger environment in which the pipeline resides. Figure 4.5 shows a virtual world that integrates the pipeline location information into the virtual environment. The figure shows a topology, beneath which a pipeline network would reside. Also, the grid above the topology is a representative display of GIS information.

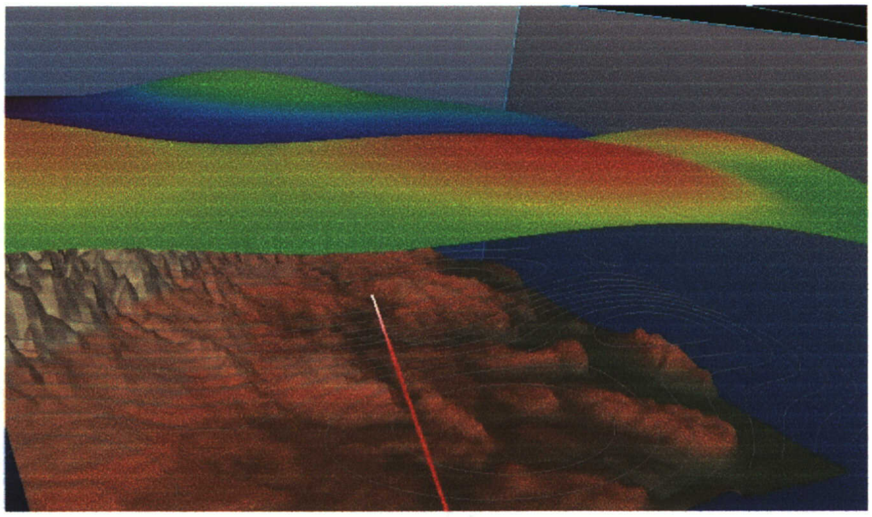


Figure 4.5 – A virtual environment used for remediation analysis.

The world pictured above, and also the one pictured in Figure 4.6, would be useful for a user considering remediation measures or performing a risk analysis. If a potentially dangerous anomaly were located in the data, the user would have the ability to break free from the pipeline and visually analyze the areas surrounding the pipeline. It is also important to notice the grid present in the background of the virtual world. The grid is color coded to allow the user to easily become oriented within the virtual world. Figure 4.6 shows a pipeline network registered beneath the topology. The GIS information is represented using contour slices. Additionally satellite imagery is incorporated into the model to allow for further visual identification. The α values and output are used in conjunction with the evolutionary predictions. The function of such inputs is described in the following sections.

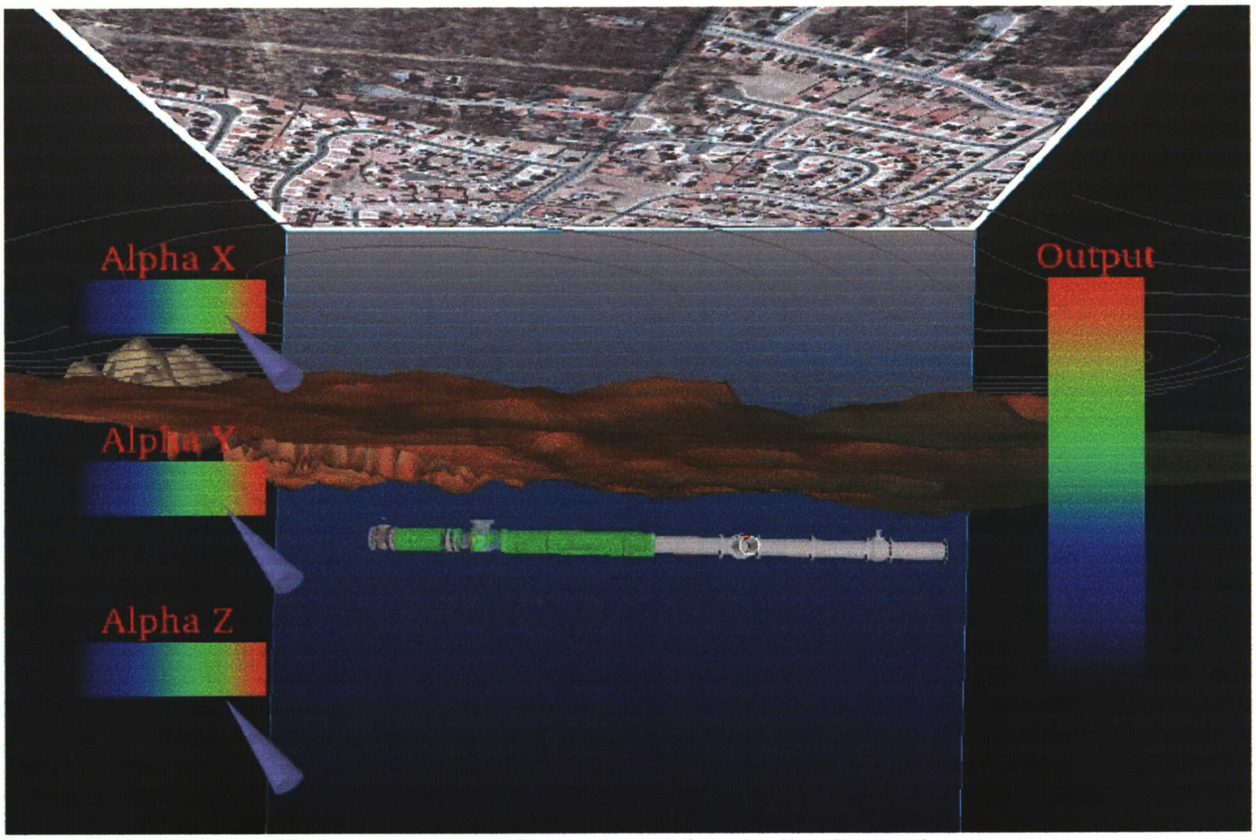


Figure 4.6 – Virtual world used for pipeline inspections.

4.2 Evolutionary Algorithms

The following sections describe the implementation of each of the four evolutionary algorithms – deterministic modeling, auto-regressive moving average models, genetic algorithm models, and hidden Markov models. Each algorithm was implemented in 1, 2 and 3-dimensions for both single and multi-path predictions. The high-level structures, single and multi-path evolution, described in Chapter 3 formed the basis for these implementations. Stress corrosion cracking is the evolutionary application being investigated.

4.2.1 Deterministic

As described in Section 3.3.3.1, the implementation of a deterministic model consisted of simulating a deterministic model as known growth rates. The growth rates were based off a typical stress intensity factor versus crack growth rate model. Various sections of the curve were used to simulate slowly growing cracks, rapidly growing cracks, as well as variable rates of crack growth. The phenomena of crack branching was also incorporated into the simulated deterministic model by splitting the crack at a designated node.

Evaluation of the prediction function required iterating the model in order to obtain a new set of node values, growth rates. The parameter correction function consisted of updating the known deterministic rates. Using these functions in the single-path model, an implementation of 1-dimensional single path crack growth can be simulated. The result is illustrated in the figure below.

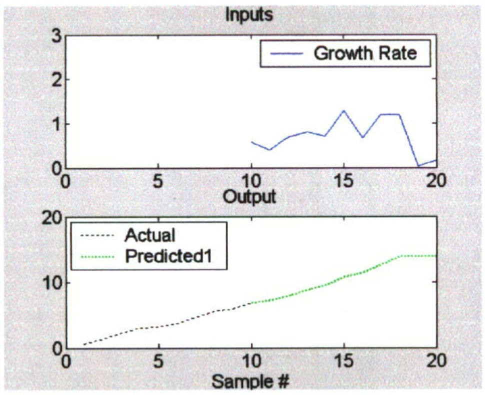


Figure 4.7 – One-dimensional single-path deterministic growth.

The simulation in Figure 4.7 is split into two sections – the first 10 samples form the basis for the evolution, and the second 10 samples consist of the evolutionary predictions. During the first 10 samples the output is seen to increase steadily, this is representative of a priori knowledge. Then the evolutionary model is used to obtain the second set of samples, where the growth rates correspond to the input, seen in the top plot, as described in the prediction function.

The same procedure can be used for an illustration of 2-dimensional single path growth. Figure 4.8 represents two such realizations of this 2-dimensional growth. In each of the displays only the output is shown. In order to generate each node, two inputs were used – growth rates along the x-axis and growth rates along the y-axis.

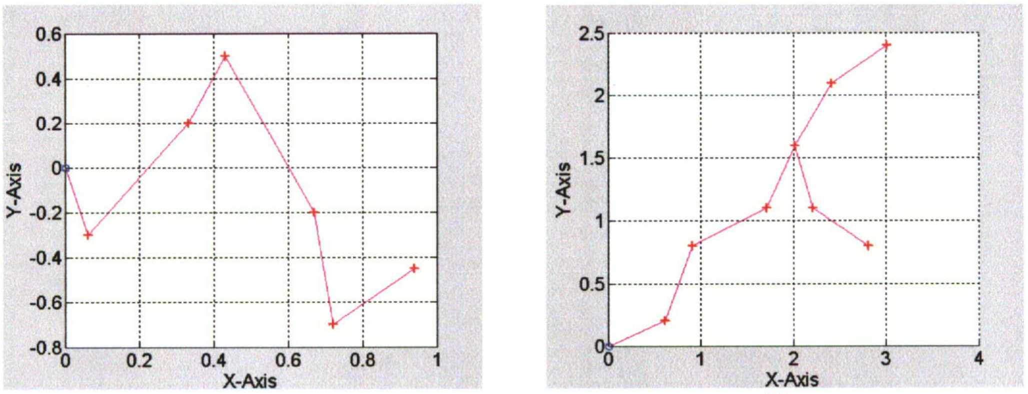


Figure 4.8 – Two-dimensional single path deterministic growth.

Figure 4.8 (a) illustrates evolution without the presence of branching. Figure 4.8 (b) illustrates how a single node, located at approximately (2, 1.5), can branch into two different and distinct paths. Finally, single path growth using deterministic modeling was implemented in 3-dimensions. The only addition from 2-D to 3-D was the incorporation of a growth rate along the z-axis. The outputs are displayed in the figure below.

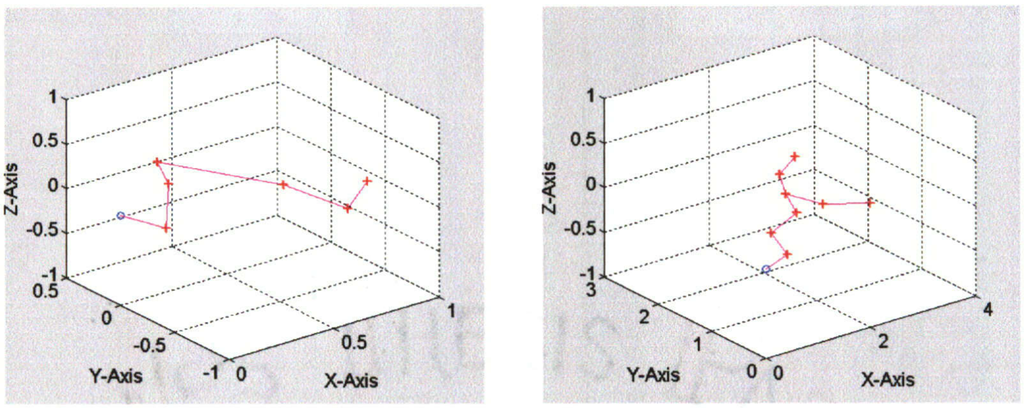


Figure 4.9 – Three-dimensional single path deterministic growth.

Using the same functions for prediction and correction, multi-path evolution can be realized. Figure 4.10 illustrates three 1-dimensional paths. All 20 points are obtained via the prediction process; all points are obtained from known, but varying growth rates. The 1-dimensional, multi-path evolution was fully constrained, as both a time and a target value were specified.

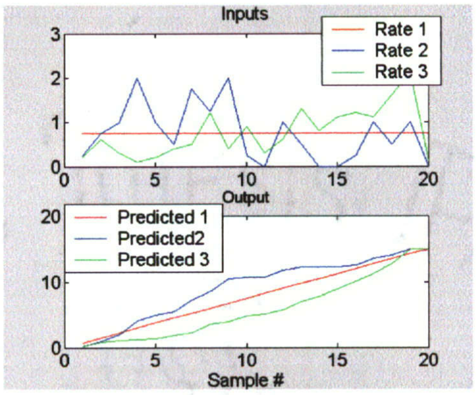


Figure 4.10 – One-dimensional multi-path deterministic growth.

The inputs, represented in the top plot, are the known growth rates. The outputs, in the bottom plot, represent the corresponding growth profile. Although each path has a unique growth signature they all terminate at the specified target values, length of 15 at 20 samples.

Two-dimensional, fully constrained, deterministic, multi-path evolution can be obtained in the same manner as 1-dimensional evolution. Subsequently, 3-dimensional evolution can also be realized. Figures 4.11 and 4.12 represent 2 and 3-dimensional evolution respectively.

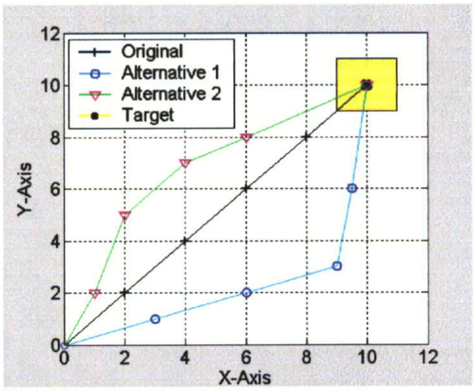


Figure 4.11 – Two-dimensional multi-path deterministic growth.

In the figure above, the target values were set to 10 units along the x and y axes; the tolerance was set to 1 unit. The yellow box therefore represents acceptable termination values. For this simulation, since the evolution was time constrained, the path needs to not only terminate within the specified target region, it must consist of exactly 4 points. All 3 paths in Figure 4.11 exhibit those characteristics and were therefore saved. The same process was performed for the 3-dimensional evolution. The only addition to the simulation was the specification of a target value of 10 units along the z-axis.

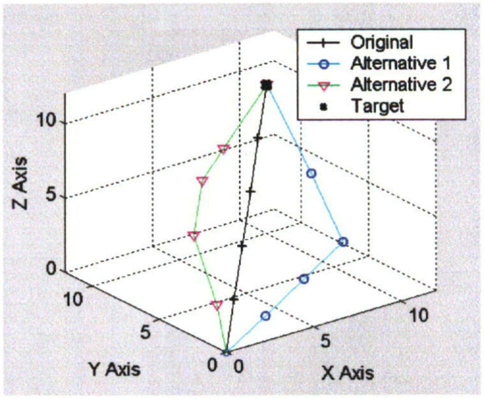


Figure 4.12 – Three-dimensional multi-path deterministic growth.

From the dimensionality progression of both single and multi-path growth, deterministic modeling in N dimensions can be realized through the prediction and correction modules. The requirement on N-dimensional growth is that N growth rates must be known.

4.2.2 Autoregressive Moving Average Models

Using an ARMA model, the prediction function required evaluating a set of filter coefficients. The filter coefficients are obtained via the correction function. The correction functions were implemented using a Steiglitz-McBride method. The ARMA models utilized normally distributed noise as the filter input. Additional details about the function implementations are located in Section 3.3.3.2.

The first investigation that was performed on the ARMA implementation was a test to see if a given impulse response could be simulated. The impulse input was 10 samples long. Additionally, random Gaussian noise was added to the impulse. Random noise excitation was used as the filter input. The simulations were run and averaged for 1,000 iterations. The mean-squared error of the predicted signal was compared to the known input signal. The mean-squared errors were plotted over a range of filter orders. The results are displayed in Figure 4.13 below.

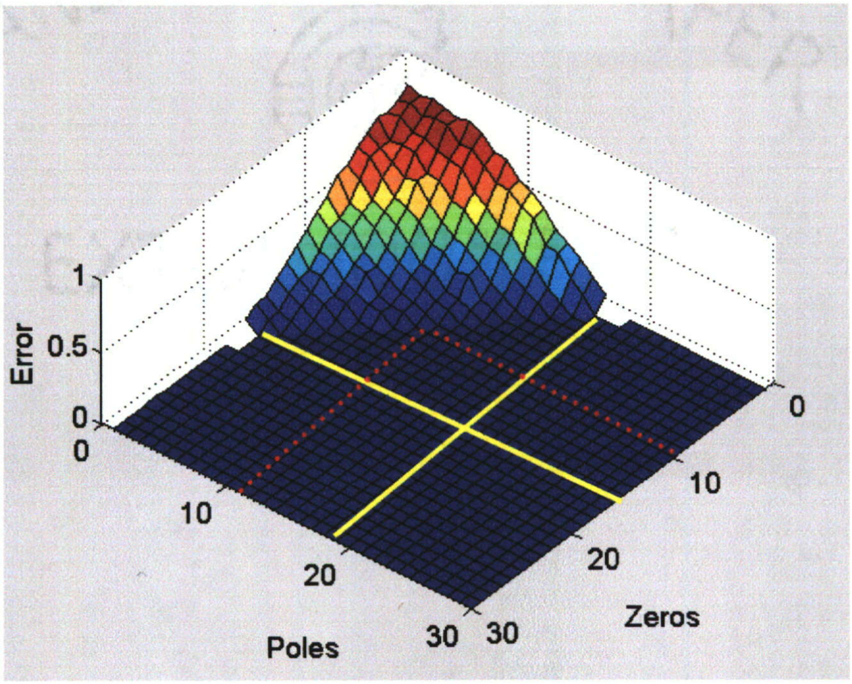


Figure 4.13 – MSE for various filter orders.

Using the Steiglitz-McBride iteration it can be seen that the MSE approaches zero as the filter order, *p*, approaches 19. This means that the given impulse could be approximated almost exactly with an AR filter of order 19. In order to achieve the same performance using a MA filter, the order required is 16. The iterations of the algorithm run during the Steiglitz-McBride iteration were capable of matching the b_n values more closely then the a_n values to their respective true values. The dotted lines in Figures 4.13 represent the values at which the MA and AR filter orders are equal and the MSE is approximately zero. From these results, the required number of coefficients for future testing of similar signals was obtained, order (10, 10).

Since the Steiglitz-McBride iteration has an additional design parameter, the number of iterations, further testing was performed on the filter to assess its performance. The order of the MA and AR filters were varied from 0 to 10. The number of algorithm iterations was varied from 0 to 20. The resulting MSE was calculated. This simulation was also averaged over 1,000 runs. Simulations where the filter exhibited instability were not included in the MSE calculation results. Figure 4.14 shows the MSE simulation results for various filter orders and algorithm iterations.

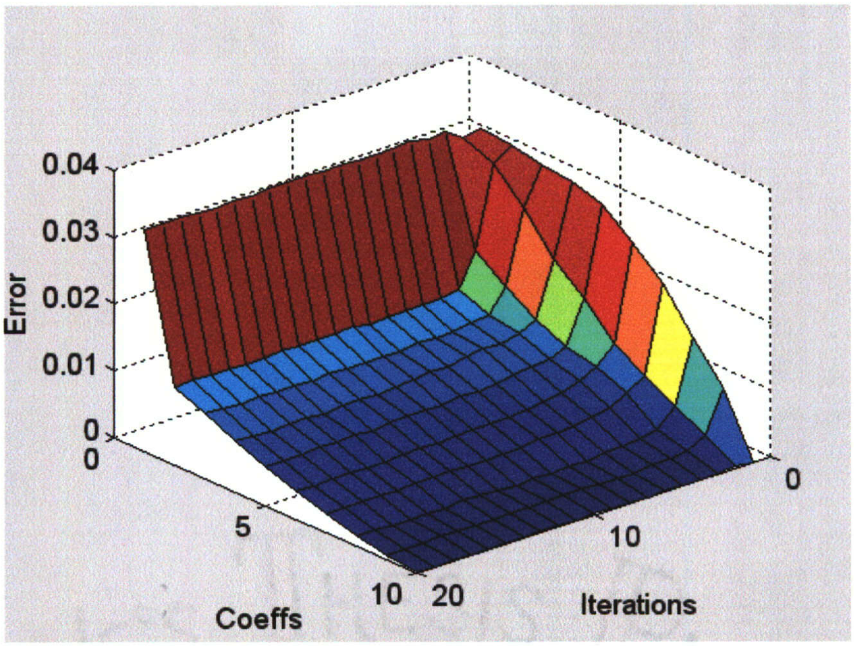


Figure 4.14 – MSE for various filter orders and algorithm iterations.

The figure illustrates that above 5 iterations, increasing the number of iterations does not affect the overall performance of the filter. Therefore, all further ARMA model coefficients were calculated using 5 iterations.

Using the prediction and correction functions 1-dimensional single path evolution can be obtained. Figure 4.15 illustrates one example of this type of evolution. The trial illustrates a 10 sample known output signal. This initial known signal is used for the initial input into the parameter correction calculations. This signal was acquired as a crack growth profile as obtained via the deterministic modeling technique described in Section 4.2.1. The signals obtained from the deterministic method are used when calculating the initial filter coefficients for all of the

ARMA models developed in this section. Using a random noise input into the ARMA model, the model was used to calculate an additional 10 samples, samples 11 to 20.

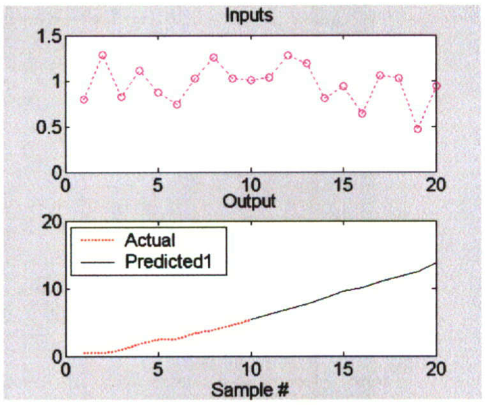
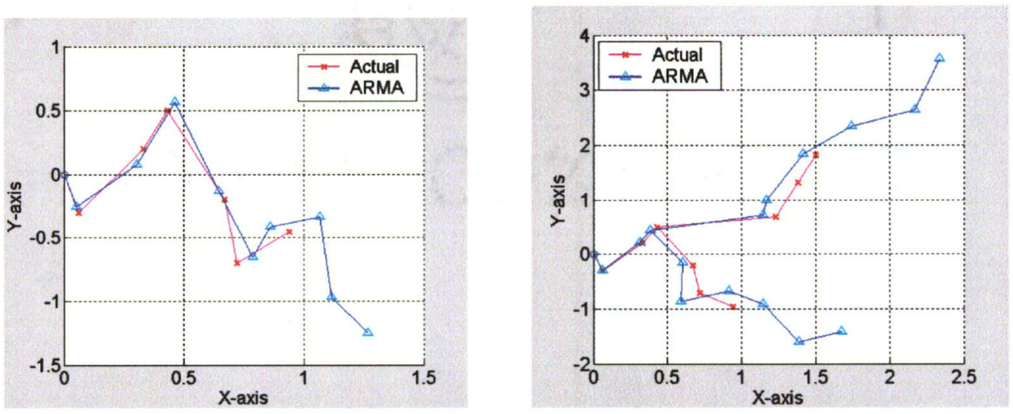


Figure 4.15 – Single-path one-dimensional evolution with ARMA model.

Two-dimensional single-path evolution can be obtained using sets of coefficients that represent each dimension individually. The assumption made for this implementation is that each dimension contains suitable information such that an evolutionary prediction can be obtained through the past information within that dimension. Figure 4.16 illustrates two examples of growth and the corresponding ARMA model predictions.



(a) **(b)**

Figure 4.16 – Single-path two-dimensional evolution with ARMA model.

In Figure 4.16 (a) the red nodes represent 6 known nodes. From these 6 known nodes, the ARMA coefficients were calculated. Using those coefficients, 9 ARMA nodes are calculated; these 9 nodes correspond to 6 nodes to match the known input and 3 evolutionary predictions. Figure 4.16 (b) also represents 3 ARMA predictions per branch. From these simulations, it can be seen that the ARMA model can produce evolutionary predictions based on the information present in each dimension.

The same approach used in 2-dimensional modeling can be applied to 3-dimensional modeling. Figure 4.17 (a) shows a single branch of a 3-dimensional ARMA model with 9 nodes, where 3 of the 9 are evolutionary predictions. Figure 4.17 (b) illustrates a 2 branch growth model also with 3 evolutionary predictions per branch.

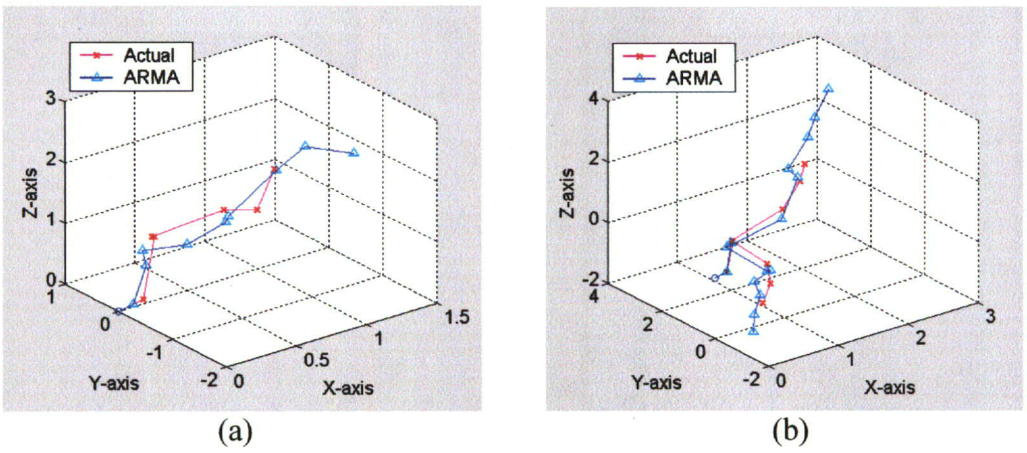


Figure 4.17 – Single-path three-dimensional evolution with ARMA model.

The α input within the high-level single-path algorithm acts directly on the filter output. Thus, the user is able to add a variable level of randomness into the filter output by varying the α parameter.

Using the same implementations of prediction and parameter correction, multi-dimensional evolution can be obtained. The 1-dimensional implementation of fully constrained evolution is depicted in Figure 4.18. The initial value is 6 samples long. The goal input is an additional 4 samples for a total of 10. The initial set of filter coefficients is calculated from the first 6 samples. These coefficients are then used to calculate 10 node values. The node values obtained from the ARMA model are subjected to a 20% variation, either positive or negative; these varied values are output from the prediction function. The level of variation is a pre-set design parameter that allows the user to specify the amount of randomness present in the output signals, just as the α parameter works in single-path evolution.

If the prediction values obtained were within the length tolerance they were accepted, otherwise another set of 10 samples was generated. The same coefficients were used to generate sets of samples until 100 failed attempts were made. After 100 failed attempts, a new set of filter coefficients were obtained using the parameter correction function.

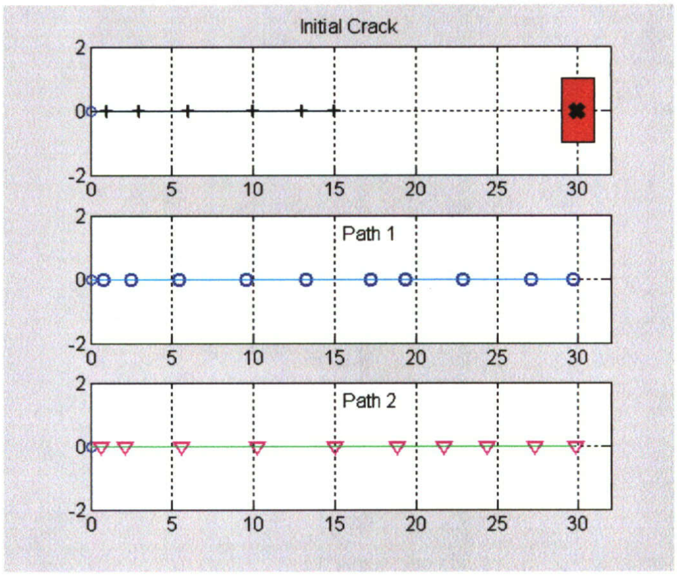


Figure 4.18 – Multi-path one-dimensional evolution with ARMA model.

The figure above shows the initial input and 2 acceptable paths that reached the goal value.

The same procedure for path generation was applied to a 2-dimensional example. Each dimension was evaluated individually in an attempt to find an acceptable path in the fewest number of iterations. Figure 4.19 illustrates an example of multi-path evolution where the basis of the parameter correction is a 6-sample path. The evolution is constrained to 10 samples and a certain target value area.

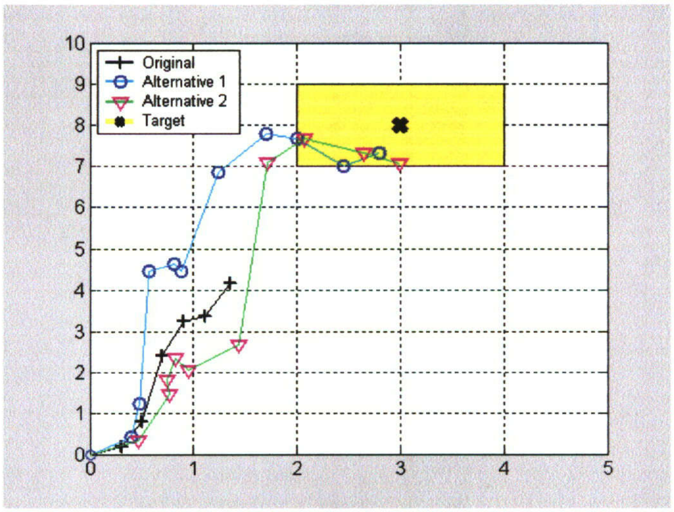


Figure 4.19 – Multi-path two-dimensional evolution with ARMA model.

Figure 4.19 shows two possible paths of length 10 that terminate within the defined target area. If the target area significantly deviates from the pattern within the initial data, the ARMA model is unable to obtain acceptable paths. The algorithm will generate sets of new coefficients, but none will be capable of reaching such a goal. After 100 failed attempts of attaining an acceptable set of coefficients the algorithm simply terminates.

If the same approach taken for 2-dimensional evolution is applied to 3-dimensional evolution, multiple 3-dimensional paths can be obtained. One such example is illustrated in Figure 4.20.

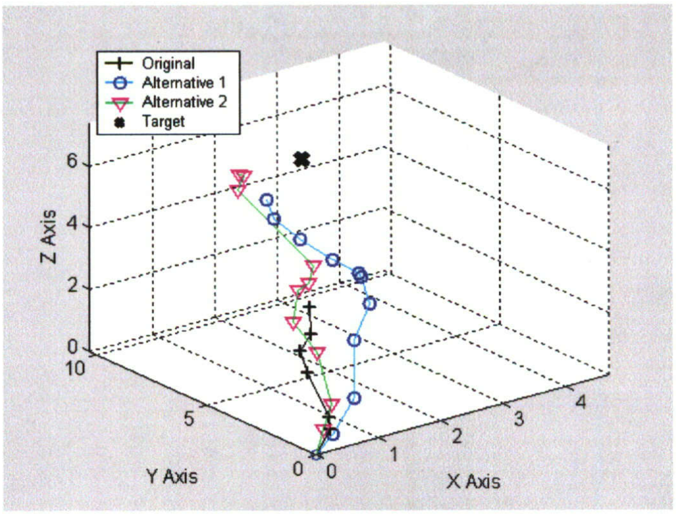


Figure 4.20 – Multi-path three-dimensional evolution with ARMA model.

Again a set of 6 values forms the basis for the initial parameter correction. The coefficients calculated from this function are used to generate sets of 10 nodes. If the nodes terminate within the 3-dimensional target area the paths are accepted. Two such acceptable paths are illustrated in the figure above.

From the 1, 2 and 3-dimensional progression it can be seen that an *N*-dimensional single-path or multi-path evolution could be achieved through the use of *N* sets of coefficients. Also, the user has the ability to set the level of randomness applied to each of the signals. Finally in multi-path evolution, the user has the ability to set termination points such that the algorithm will terminate if acceptable paths are not being reached.

4.2.3 Genetic Algorithms

A genetic algorithm was developed in order to achieve evolutionary predictions. The implementation was based on a typical genetic algorithm structure as described in Chapter 2, Figure 2.7. The genotypes, a direct representation of the evolutionary format, were normalized growth rates between –1 and +1. The prediction function required evolving a population of individuals. The correction function allowed user input controls within the genetic algorithm structure to be altered. The implementation specifics for the genetic algorithm structure are located in Section 3.3.3.3.

A single population of 50 individuals was created. The population evolved over 30 generations with a mutation rate of .01. Each individual has 9 chromosomes, 6 correspond to the input signal and 3 correspond to the user input. The input signal represents known growth rates and was obtained via the deterministic modeling procedure. For the fitness evaluation function, the standard deviation for the first 6 nodes was set to .15 and the mean was set to the input signal value. The standard deviation used for the predicted values was .3 and the mean was user-input at .6. After the population evolved for 30 generations, a representative individual was selected. Figure 4.21 (a) and (b) illustrate two such representative individuals from different populations, with different input signals. In each case, 6 values corresponded to matching the initial path and 3 points represent predictions.

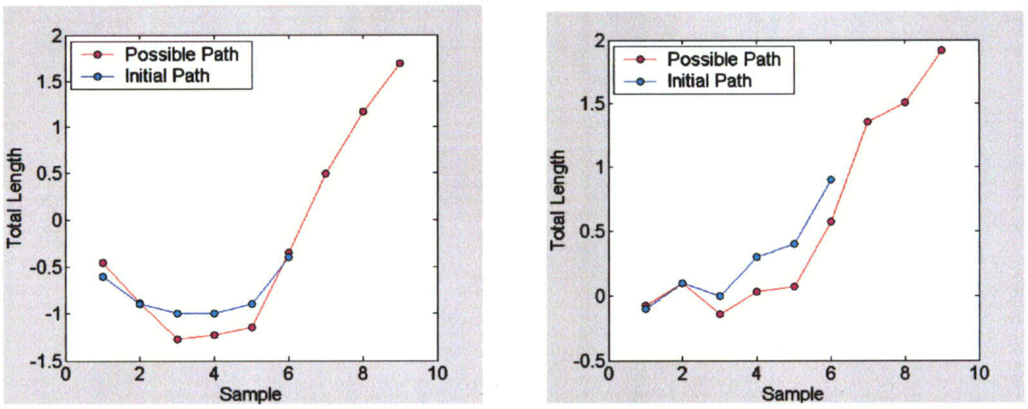


Figure 4.21 – Single-path one-dimensional evolution with GA.

Using the parameters described above it is seen in Figure 4.21 that the individuals evolved towards a fit of the input data. An analysis of the entire population, similar to that performed in Figure 4.22, showed that most individuals in the respective populations were also approaching the same general form as the ones displayed in Figure 4.21. This means that the samples displayed are truly representative of the population's solution. This also implies that the population is evolving towards a single, potentially optimal, solution. Lowering the standard deviation of the Gaussian curves allowed for tighter fits of the predictions to the data. Similarly, if the deviation was increased, the fit of the individuals within the population to the known signal were not as precise.

The same design parameters were used to evolve an input signal, but the population was allowed to evolve over 100 generations. The values of all the individuals were averaged, and tracked through the generations. Figure 4.22 shows the chromosome history for the first 6 values.

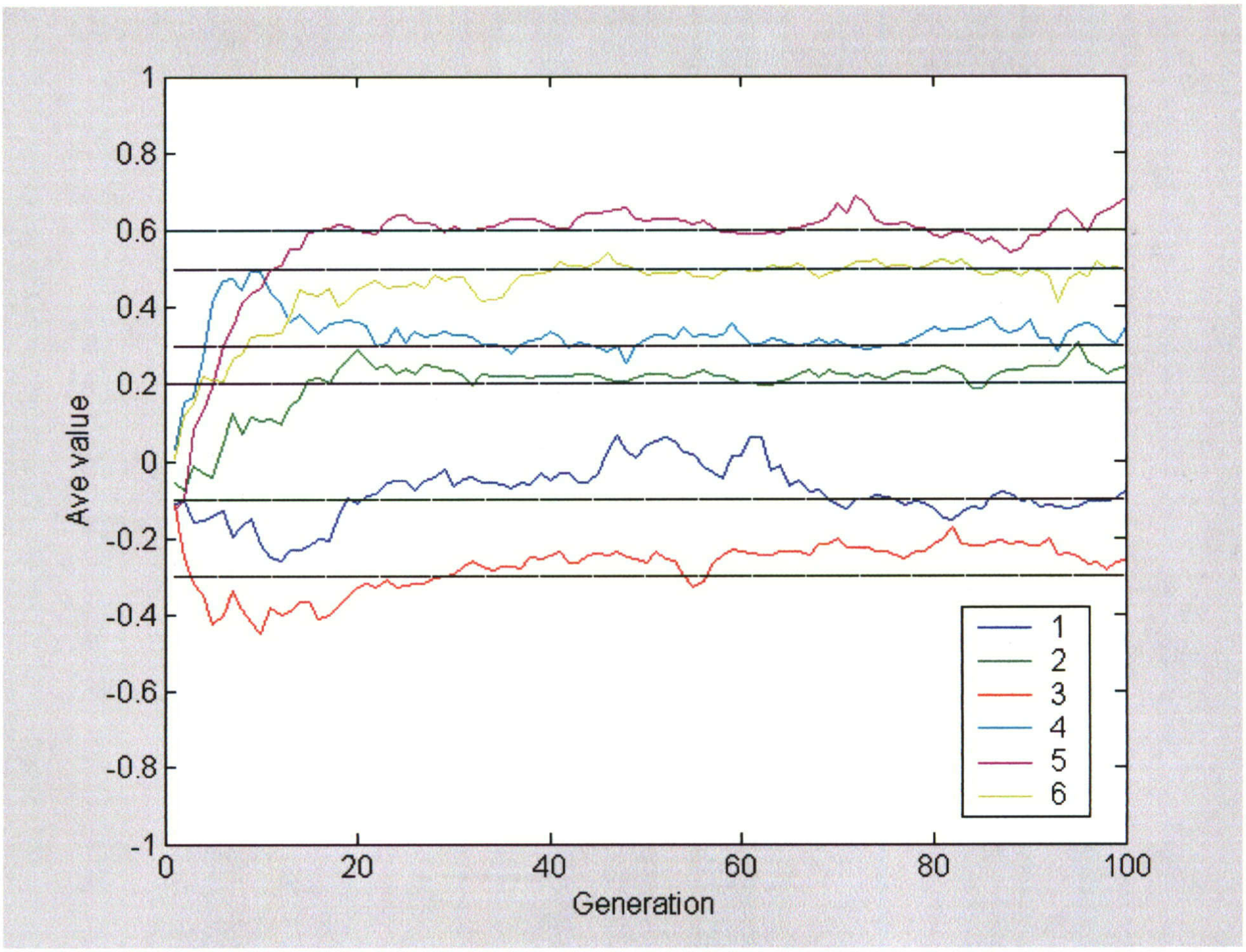


Figure 4.22 – Chromosome evolution.

From this plot it can be seen that as the population evolves, all of the individuals begin to resemble the input signal by converging towards the known values. The population exhibits a great deal of randomness during the early generations and gradually settles to the desired values. Because of the mutation rates, there is still some amount of oscillation around the ideal values. Figure 4.23, shows the phenotype of the average individual across a range of generation values.

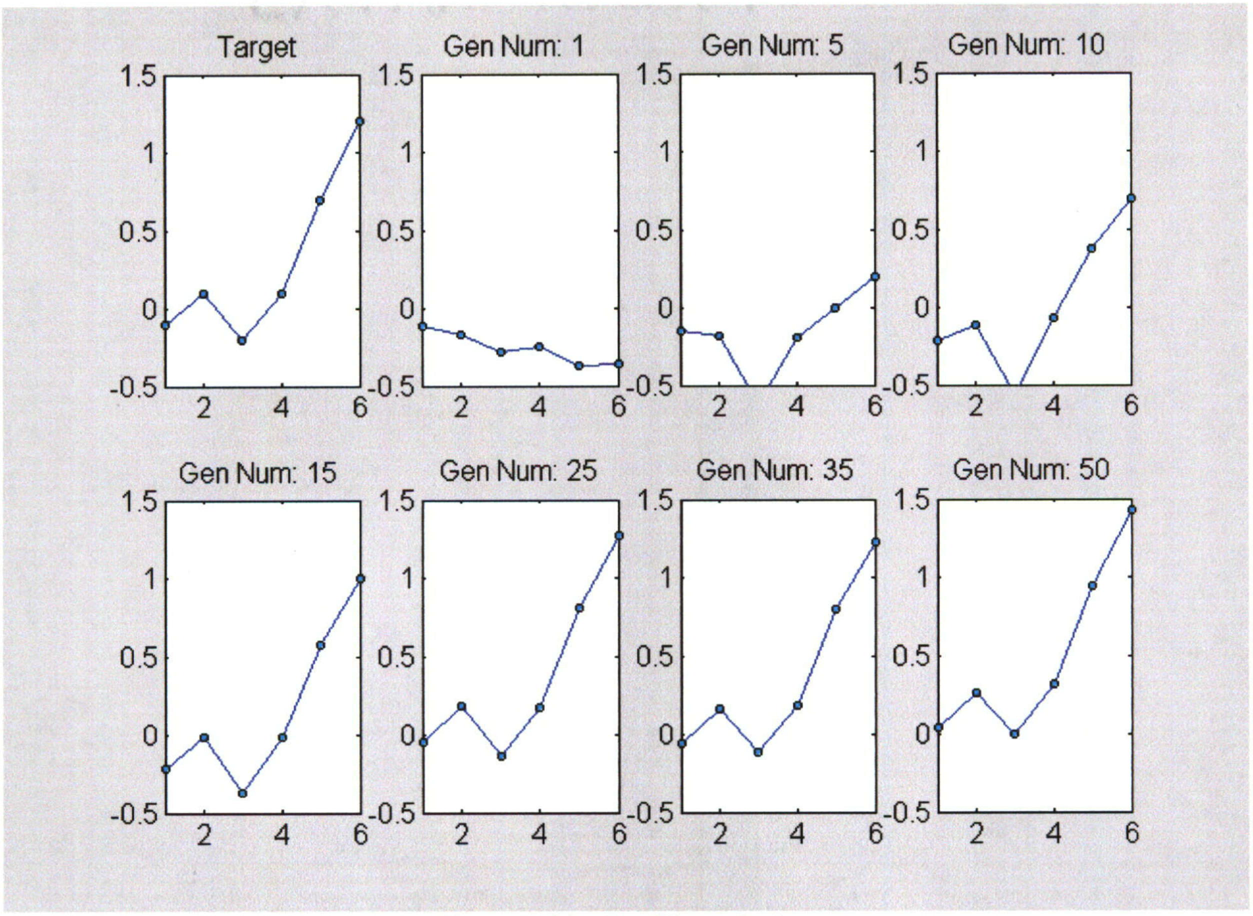


Figure 4.23 – Profile evolution.

The first plot represents the target value; these values are the mean values used in the fitness evaluation function. As the population evolves, the average individual is seen to evolve towards the target value.

Next, 2-dimensional evolution was performed using the same basic structure. The 2 dimensions were allowed to evolve independent of each other. Independent populations were used because the convergence was faster and the fit to the data was more accurate as compared to a single multi-dimensional population. The 2-dimensional structures were evolved over 40 generations for a population size of 80. The mutation rate was increased to .1 to allow for a greater amount of diversity during high generation numbers. Representative individuals from two different growth scenarios are displayed in Figure 4.24.

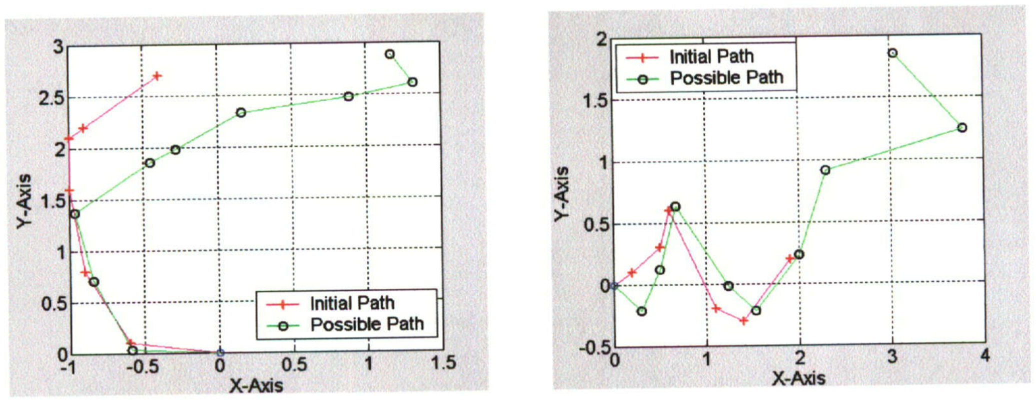


Figure 4.24 – Single-path two-dimensional evolution with GA.

Finally, 3-dimensional single path evolution was obtained in a manner similar to that of 2-dimensional evolution. The three population sizes were set to 70 and the generation number to 50. The resulting evolutionary paths are illustrated in Figure 4.25

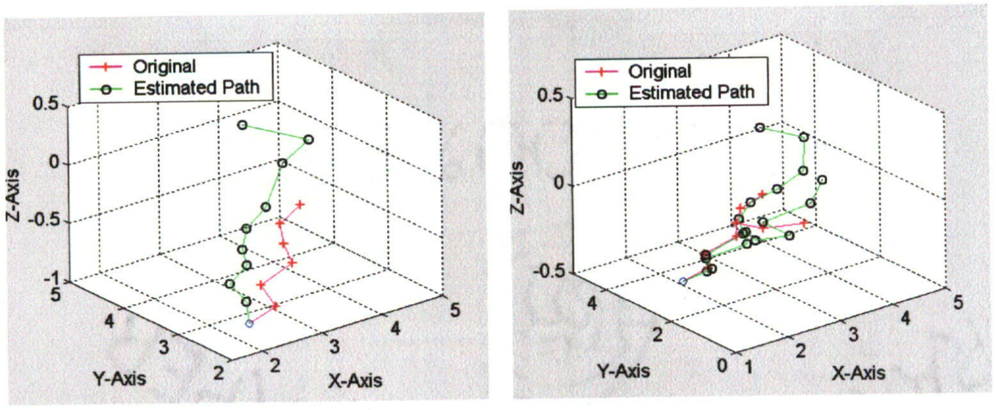


Figure 4.25 – Single-path three-dimensional evolution with GA.

Figure 4.25 (b) shows an example of a branched path growth. Each individual in this population therefore consisted of matrix nodes, instead of single valued nodes. The overall fitness of the individual is based on the combined fitness for both branches. The branches are not evaluated independently.

Using the same prediction and parameter correction functions as single-path evolution, multi-path evolution can be obtained. In multi-path evolution an entire population is evaluated in the same manner as single-path evolution. An output path, a single individual, is then chosen from the population. If that path satisfies the goal, then it is stored. If the path does not satisfy the constraints then a new individual is chosen. The path selection is repeated until either a valid path is selected or there have been 20 failed attempts to obtain an individual. When a suitable path is not attained, a new population is generated and evolved in an attempt to generate more suitable individuals. Multi-path, 1-dimensional evolution was achieved using a population of size 30, evolved over 20 generations, mutation rate of .01, a prediction standard deviation of .3, and a user standard deviation of .6. The results are displayed in Figure 4.26

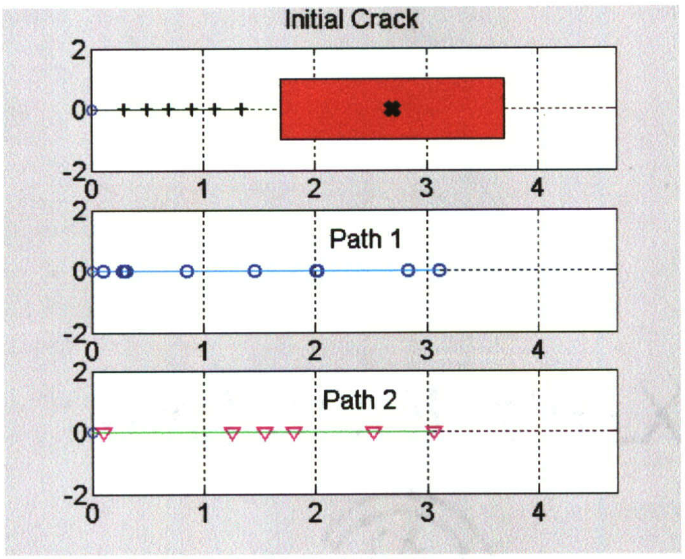


Figure 4.26 – Multi-path one-dimensional evolution with GA.

Next, a 2-dimensional goal was set. The same evolutionary parameters used in 1-dimensional evolution were used for the 2-dimensional evolution. The original input, and two possible paths from the population are illustrated in Figure 4.23. The initial signal consists of 6 points, the target signal consists of 12. The evolution is fully-constrained as both a target value area and a desired number of points are specified.

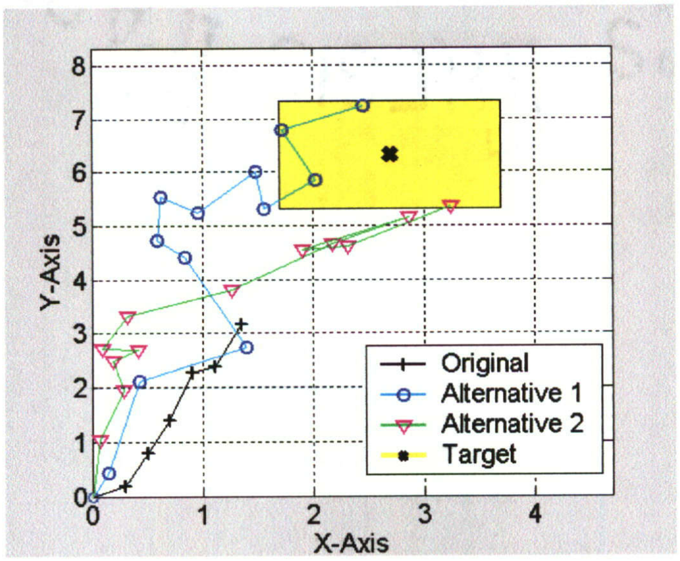


Figure 4.27 – Multi-path two-dimensional evolution with GA.

Finally, 3-dimensional evolution was achieved by adding a third population that represented the z-axis. Again the same evolutionary parameter set was utilized to achieve 3-dimensional evolution. The result is shown in Figure 4.28

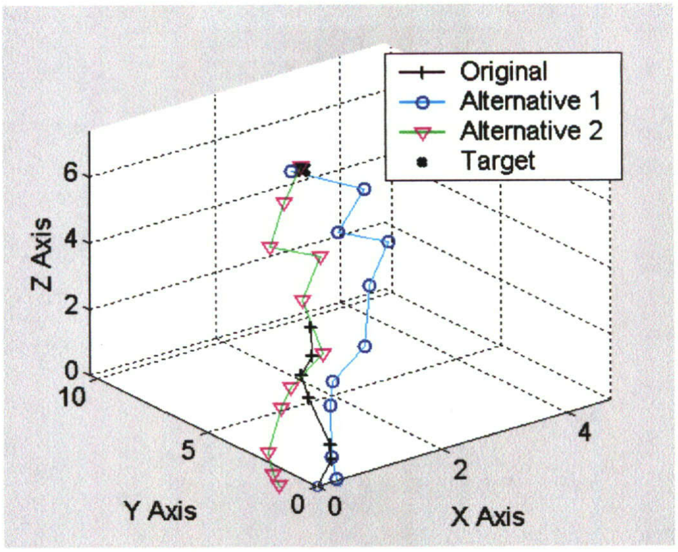


Figure 4.28 – Multi-path three-dimensional evolution with GA.

From the 1, 2, and 3-dimensional implementations of both single and multi-path evolution it can be seen that N-dimensional evolution would be realizable. Each dimension could have a population from which representative individuals could be selected. Multiple, single dimensioned populations, have a number of advantages over a single multi-dimensional population. Each population is much smaller in size and requires a lower number of generations to converge to an acceptable answer. Also during the output selection process if one-dimension does not meet a certain criteria that single dimension can be reselected instead of the entire growth profile. This implies that for N, 1-dimensional populations of size p there would be p^N individuals of N-dimensions. The final benefit is that separate populations are more conducive to parallel or distributed processing implementations then a single larger population.

4.2.4 Hidden Markov Models

The last of the evolutionary algorithms that was implemented was a hidden Markov model. The HMM focused on providing transitional and visible state probabilities such that a representative growth profile could be obtained. Because there was not an overwhelming amount of data to train with, the algorithm implementation assumes some a priori knowledge about the phenomena under investigation. Experiments can be performed in order to achieve a suitable amount of reliable data to allow for a physically accurate probabilistic model [48]. Assuming this information is known, the HMM structure and state transition probabilities can be set prior to running the algorithm. The structure used in these experiments consists of 3 hidden states, and 3 possible visible states emitted from each hidden state. The prediction function therefore consisted of evaluating the HMM for a given number of iterations. For each iteration, the hidden state is transitioned and a visible state is observed. The correction function required updating the visible state transition probabilities. Section 3.3.3.4 explains the implementation specifics for the HMM.

In order to achieve 1-dimensional growth, a HMM was developed with 3 hidden states and 3 visible states. The states were fully connected; there was a finite, non-zero probability that a given hidden state could transition into any 1 of the 3 hidden states. Initial probabilities were based off the characteristics of the initial signal obtained via deterministic modeling. For signals with large growth rates the hidden state transitional probabilities favor hidden state 1. When the growth rates are small, the transitional probabilities favor hidden state 3. The hidden state transitional probabilities were fixed; the visible state probabilities were adjusted through user input and the parameter correction function. The visible state values are given below.

Table 4.1 Visible state values

$v_{11} = .7$	$v_{12} = .8$	$v_{13} = .9$
$v_{21} = .4$	$v_{22} = .5$	$v_{23} = .6$
$v_{31} = .1$	$v_{32} = .2$	$v_{33} = .3$

Using this structure, a 1-dimensional single path evolution was obtained. Figure 4.29 shows two single-path evolutions for a HMM growth profile.

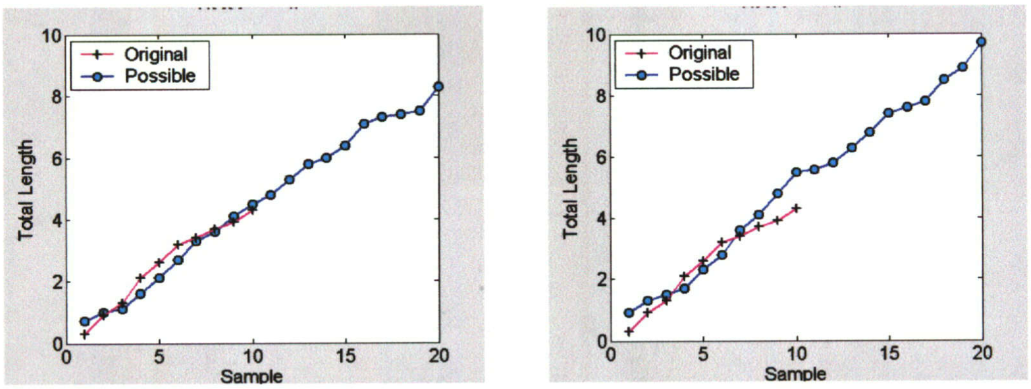


Figure 4.29 – Single-path one-dimensional evolution with HMM.

The first set of 10 points represents the initial signal obtained from deterministic modeling. The characteristics within this signal determined the hidden state transitional probabilities. The HMM was then evaluated over 20 runs. The first 10 nodes correspond to the initial signal and the second 10 nodes correspond to evolutionary predictions.

Similar to 1-dimensional, 2-dimensional growth was realized. The overall structure of the model was not changed. In order to follow the same trend as the other evolutionary modules, a second structure was designed for growth in the second dimension. The overall structure of the HMM was the same; however the hidden state transitional probabilities were altered to allow a fit to the given 2-dimensional data. Figure 4.30 illustrates the realization of this HMM structure.

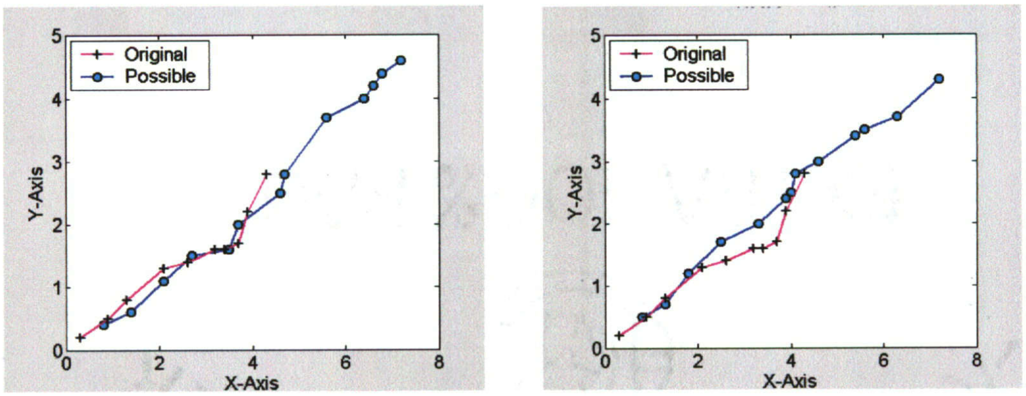


Figure 4.30 – Single-path two-dimensional evolution with HMM.

Finally, single-path 3-dimensional growth was realized by adding a third model to simulate the evolution in 3 dimensions. Figure 4.31 illustrates the output from this model. From the dimensionality progression, an *N*-dimensional HMM can be realized through the use of additional independent hidden states. Although not implemented, the model could have shared hidden states and exhibited dimension-independent visible states.

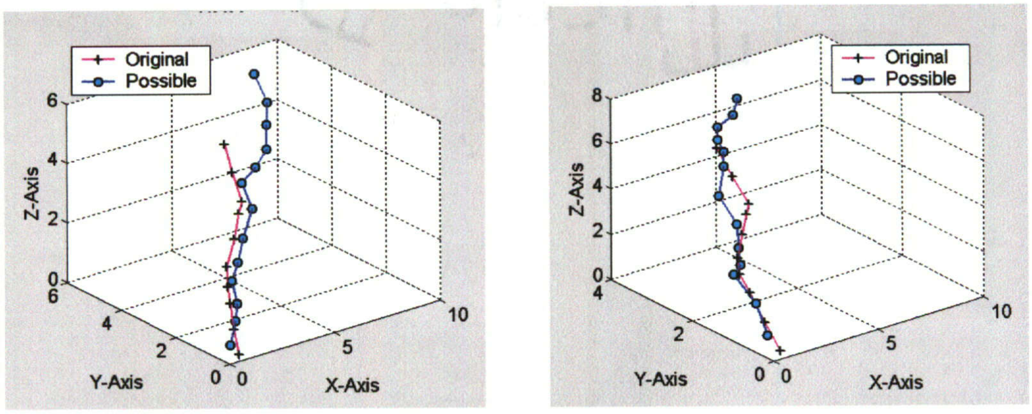


Figure 4.31 – Single-path three-dimensional evolution with HMM.

Next, the prediction and correction functions were used to create multi-path growth. Under the given structure, the model was evaluated until an acceptable number of paths were reached. If the goal was not realizable, the visible state probabilities were slightly altered. One-dimensional multi-path evolution is described in Figure 4.32 below.

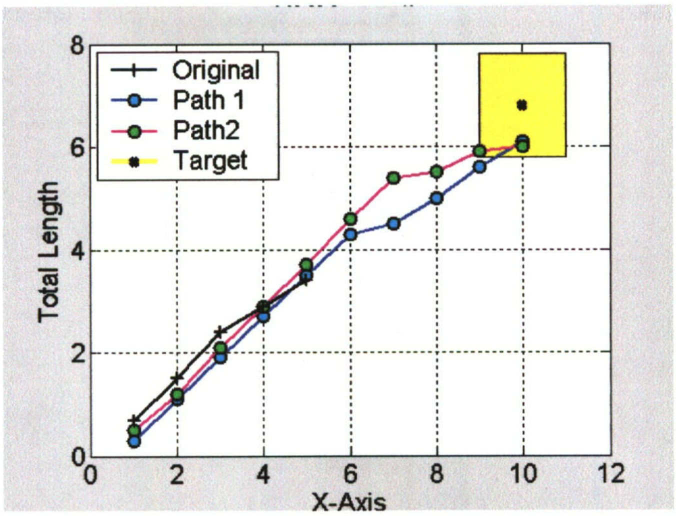


Figure 4.32 – Multi-path one-dimensional evolution with HMM.

Just as single path evolution used an addition structure to realize 2 and 3-dimensional evolution, multi-path evolution could be achieved in the same manner. From the known HMM the evaluation of the model resulted in 2 and 3-dimensional growth profiles. The evolution was fully-constrained, both a target value area and the number of nodes was specified. The results from typical runs are shown in Figures 4.33 and 4.34.

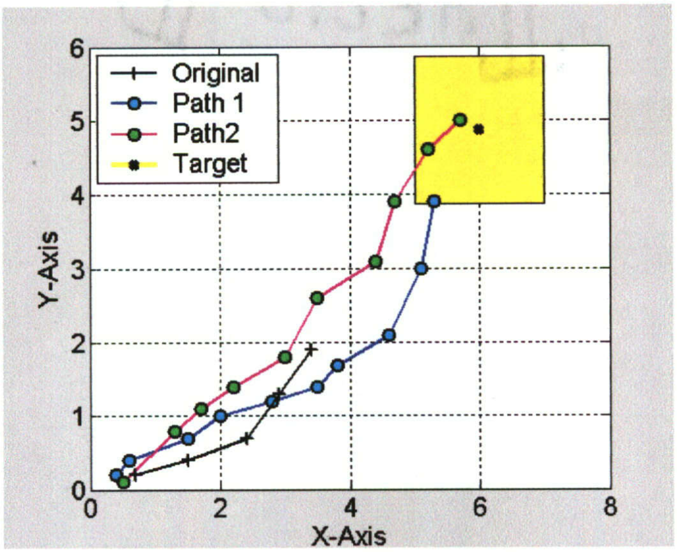


Figure 4.33 – Multi-path two-dimensional evolution with HMM.

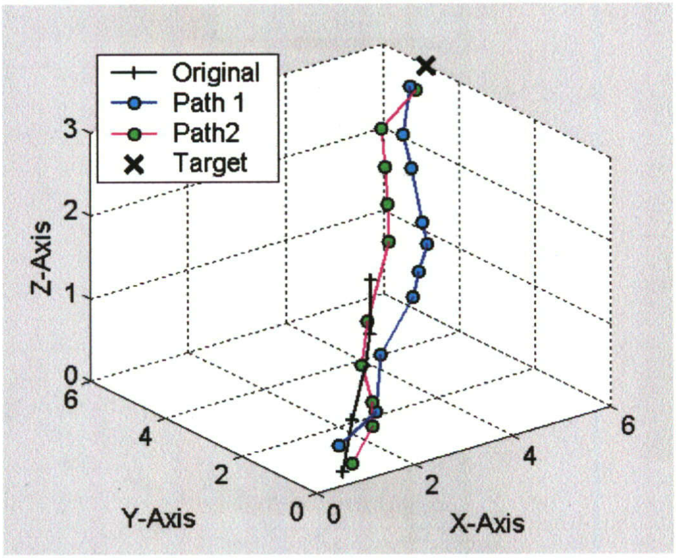


Figure 4.34 – Multi-path three-dimensional evolution with HMM.

Using a set of HMMs *N*-dimensional single and multi-path evolutions can be realized. There are a number of methods in which the models can be structured in order to achieve this realization. The exact number of dimensions and the resolution of each dimension would both have an impact on the ideal structure. Since the HMMs implemented were discrete, the resolution of the models was a function of the visible state outputs. A series of visible states would allow for finer resolution to be obtained at the cost of computational complexity. Similarly, altering the desired resolution may require changing the structure of the hidden states. The hidden states are transparent to the end VR user, but are available to the evolutionary designer.

4.3 Evolutionary Visualizations

The evolutionary visualizations took place within the same world as the multi-sensor data fusion described in Section 4.1. Within this world there were two coordinate systems, one navigable and one static. The static coordinate system was used in order to communicate the evolutionary information with the user. The graphical, functional, and measurement data were represented in the navigable coordinate system. In addition to an output meter, a graphical representation of crack growth was displayed.

Crack growth can be mapped from the evolutionary parameter format in a number of ways. In a direct mapping approach, a 3-dimensinal evolutionary variable would be used. Each of the three dimensions would map directly to a point on the crack in x, y, and z dimensions. If the crack is assumed to be a surface crack, then a 2-dimensional evolutionary variable can be employed. The first dimension would correspond to the propagation of the crack along the main axis of the pipe. The second dimension would correspond to the crack propagation along the circumference of the pipe. In addition to these two spatial approaches, time can also be predicted. If time is not a dimension of the evolution, then it is assumed that there is a constant time interval between the evolutionary predictions. However, time can be incorporated into the model such that an evolutionary prediction would represent the total time elapsed since the previously predicted data point.

Figure 4.35 illustrates a virtual environment set-up for the prediction of a crack on the surface of a pipeline. The pipe section forms the graphical basis of the world. Within the pipe section, a crack has been physically modeled. In addition to the physical world, the user inputs and outputs can also be seen.

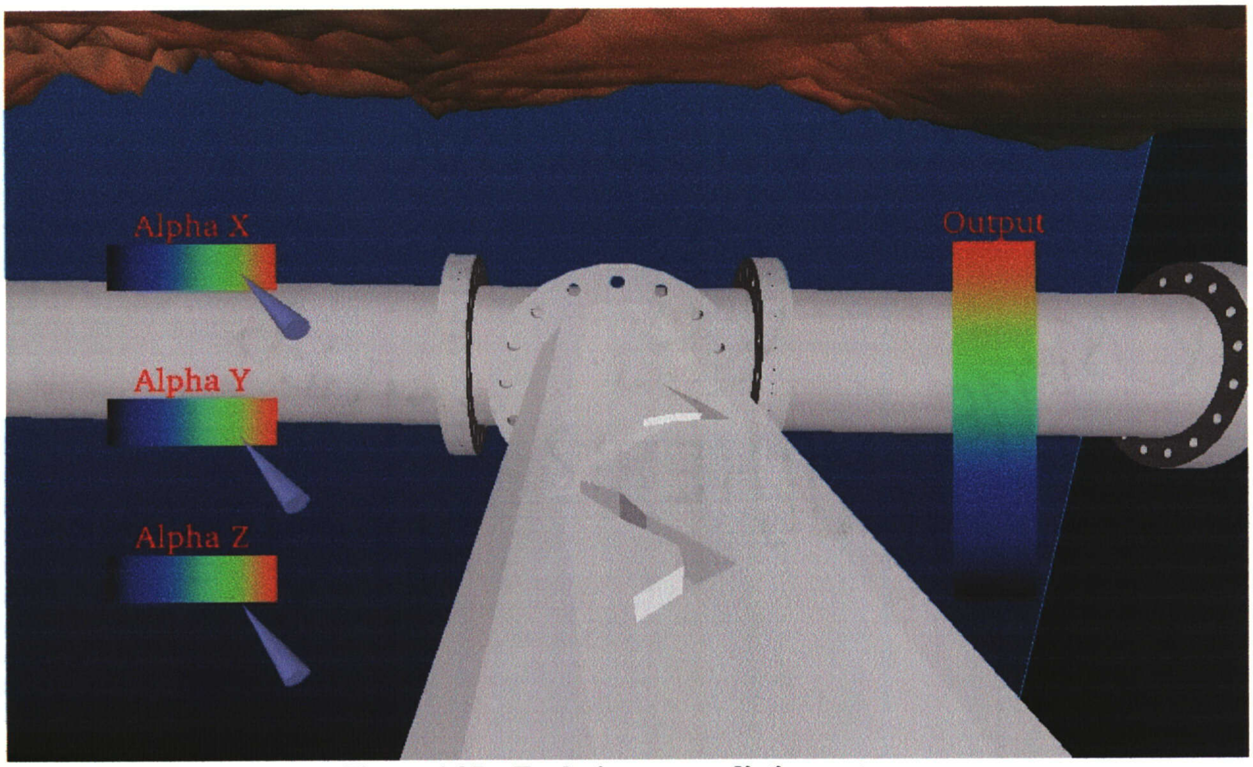


Figure 4.35 – Evolutionary predictions set-up.

Once the user sets the various evolutionary inputs, any one of the evolutionary algorithms could be used to predict the growth of the crack. For this particular implementation only part of the crack was input into the algorithm and the remainder of the crack was predicted. Additionally, the propagation of the crack beyond the known geometry was also predicted. This example can be seen in Figure 4.36. After (or during) the evolution of the crack, the VR user would have the ability to look at the area surrounding the pipeline, Figures 4.5 and 4.6 illustrate this concept. From the information in the surrounding area and the information contained within the evolutionary prediction, the user can draw informed conclusions.

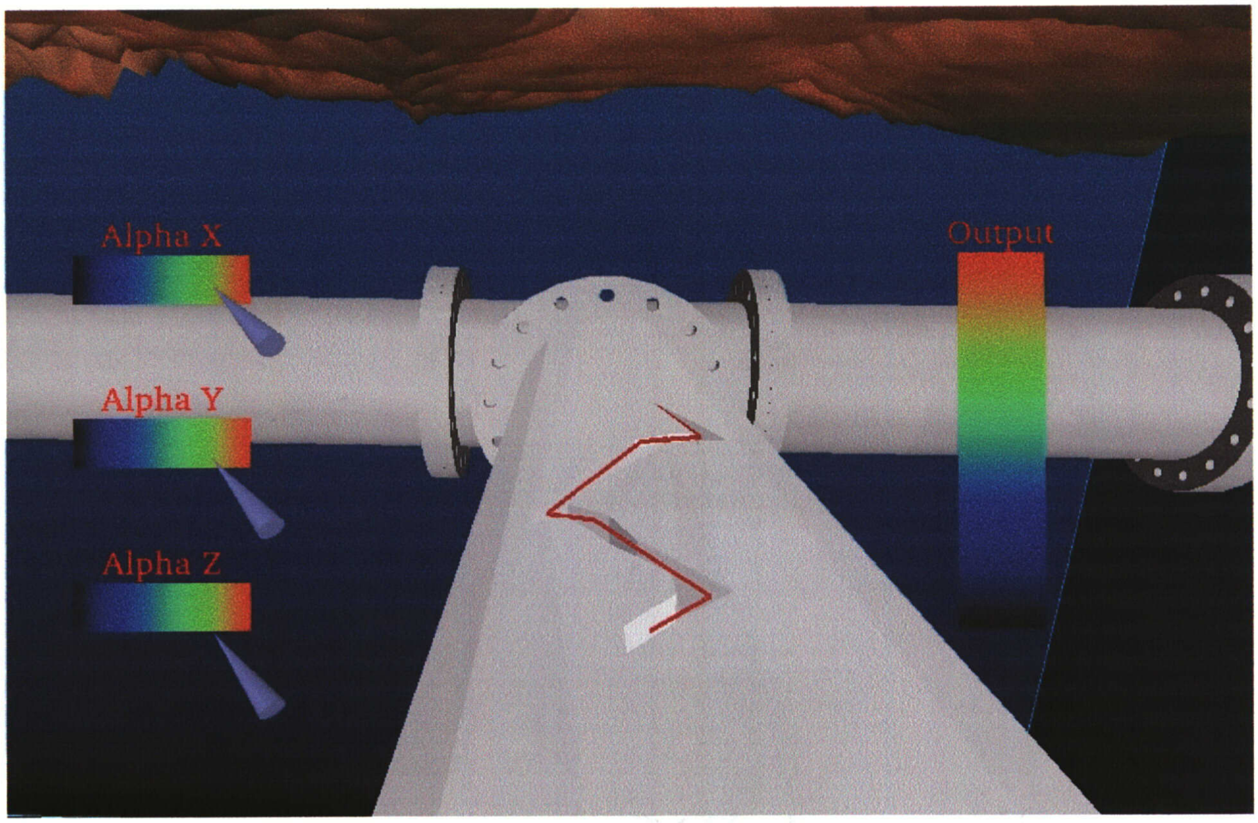


Figure 4.36 – Possible evolutionary path.

4.4 Evaluation Criteria

4.4.1 Rating Scale

The rating scale for each of the evaluation criteria features is from 0 to 10. The scale is shown in Figure 4.37. A rating of 0 represents an unacceptably low result. A 0 rating would necessitate change for the algorithm to be considered further. Ratings ranging from 1 to 3 represent below average ratings. Ratings of 4 to 6 represent an average rating. If an algorithm's performance is above average it receives a rating between 7 and 9. Finally, a score of 10 implies the algorithm is ideal for the described feature.

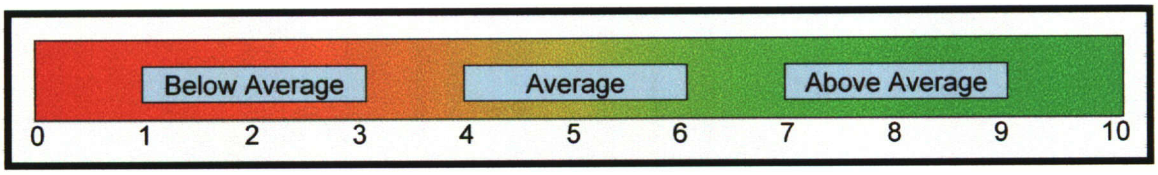


Figure 4.37 – Evaluation rating scale.

One of the common threads to many of the evaluation criteria was the computational complexity of the algorithm. For example, both implementation characteristics take into consideration the simulations required to test the given implementation. Similarly, the speed criterion is specifically concerned with the performance of the algorithms' given functions. The remainder of this section is devoted to a breakdown of computational cost.

For ARMA modeling, the two main functions are prediction and correction. The prediction function consists of evaluating a filter, and the correction requires calculating filter coefficients via the Steiglitz-McBride iteration. The table below breaks down the given computational costs.

Table 4.2 – Computational expense for ARMA modeling

Function	***Operation***	***Number of operations***	***Iterations***
Prediction	Filter	2p+2q-1	1
Correction	Prefilter	2q-3	2 * its
	Convolution Matrix	(p+1)M	its
		(q+1)M	its
	Concatenate	1	its
	Least fit	p+q+1	its

p: Number of poles	M: Size of input
q: Number of zeros	its: Number of iterations

The genetic algorithm structure is intricate. Both the prediction and correction functions evolve a population over a certain number of generations. The table below breaks down the evolution of a population into the various operations and the computational costs for each.

Table 4.3 – Computational expense for genetic algorithm

Function	***Operation***	***Number of operations***	***Iterations***
Evolution	Initialize population	M * N	1
	Evaluation	2 * M * N - N	genNum
	Calculate probability	N	genNum
	Select mates	4	(N / 2) * genNum
	Mate	3	(N / 2) * genNum
	Mutate	M * N	(N / 2) * genNum

M: Length of chromosome	genNum: Number of generations
N: Population size	

The computational costs of evolution with a HMM are broken down in Table 4.4 below. The learning function was not used in the previous sections because sufficient data was not available. However, the learning method for the transitional probabilities is taken into consideration for the evaluation of a HMM over the range of criteria.

Table 4.4 Computational expense for HMM

Function	***Operation***	***Number of operations***	***Iterations***
Prediction	Generate random #	1	M
	Evaluate next state	1	M
	Generate random #	1	M
	Evaluate visible state	1	M
Learning	Forward-Backward	$\sim10^{5\text{-}6}$	1

M: length of 1 set of visible states
~Will vary as a function of data complexity, but the value given represents a typical range

4.4.2 Versatility

Versatility reflects the implementation requirements for the input of the evolutionary parameter. The rating also reflects the ability of the algorithm to adjust to changes in the format of that variable. Since versatility reflects implementation considerations it was broken into two separate elements, man hours and computer hours. Assuming a typical work day, an engineer would account for 8 hours of man hours, and the remaining 16 hours would be optimally used for simulations, computer hours. Therefore to optimize daily productivity man hours should receive a 2:1 weight over computer hours. Using the same ratio, the versatility score of 10 can be broken into sections, 6.66 for man hours and 3.33 for computer hours.

Typically, man hours for a minor change in an algorithm such as these would require approximately 2 hours - 1 for analyzing simulations and 1 hour for performing the necessary changes. The simulations performed would consist of approximately 500 million computations. Using these 2 values as base values, the scores for all 4 algorithms were calculated by dividing the algorithm's score by the base value target score. The chart below illustrates each of the four algorithms versatility performance rating.

Table 4.5 – Versatility ratings

		Target	*Deterministic*	*ARMA*	*GA*	*HMM*
Man Hours	Value [hours]	2	6	2	6	5
	Score	0-1	0.333333333	1	0.333333333	0.4
	Weighted Score	6.66	2.22	6.66	2.22	2.664
Computer Hours	Value [computations]	500,000,000	1,000,000,000	936,100,000	2,789,400,000	500,000,000
	Score	0-1	0.5	0.534130969	0.179250018	1
	Weighted Score	3.33	1.665	1.778656126	0.59690256	3.33
Versatility	**Total Score**	**10**	**3.89**	**8.44**	**2.82**	**5.99**

4.4.2.1 Deterministic

In setting the deterministic parameters directly, it was assumed that a direct mapping can be made between the deterministic analysis output and the VR evolutionary variable. Often times, the deterministic output will be a direct and specific measure of the physical state; for such cases a direct mapping to the visualization model may be difficult or prohibitively costly. In order to perform the mapping, an additional interface is required to ensure that the output is in the proper format. An additional interface means that the algorithm is less versatile. In order to code, and test the interface a total of 4 man hours is needed. An additional 2 hours is needed to analyze the simulation results and make the necessary adjustments. Performing the required simulations for the interface and resulting model would take approximately 10^9 computations. These values result in a score of 3.89 for deterministic modeling.

4.4.2.2 ARMA

ARMA models are quite versatile to changes within the input parameters. The ARMA coefficient calculations are based on the filter inputs and corresponding outputs. The actual format of the data and the corresponding meaning of this format are unimportant to the filter. However, some simulations should be performed in an attempt to optimize the filter parameters. It will take less then 2 man hours to analyze the simulations and make the necessary adjustments to the functions. The simulations being performed require optimizing the number of poles, zeros and iterations of the Steiglitz-McBride function. Assuming the poles and zeros are both varied from 1 to 100 and the iterations are varied from 1 to 10, simulating all possible combinations will require 9.36×10^8 computations. Using the man hours and computer hours required to perform the necessary changes the ARMA model receives a score of 8.44.

4.4.2.3 Genetic Algorithms

The genetic algorithm is seen to have below-average implementation versatility. The algorithm has a number of parameters that must be set according to the format of the evolutionary variable. If the evolutionary variable represents a phenotype, an associated genotype must be defined explicitly for use in a genetic algorithm. Additionally, the cost function must be implemented such that the proper calculations are performed on the genotype.

A change in the evolutionary variable format would require an analysis of the cost function being used. In addition to analyzing a variety of cost functions, different normalization strategies of the fitness score also need to be addressed. To perform a reasonable analysis, at least 5 different cost functions and normalization strategies should be analyzed. For the simulations in Section 4.2.3, both Gaussian and Euclidean distance functions were evaluated. The Gaussian function was evaluated over 3 different normalization strategies, and the Euclidean distance was evaluated over 2. Assuming 1 hour for implementation and testing of each strategy and 1 hour for interpreting all of the simulation results, approximately 6 man hours would be required. For each simulation the number of individuals in the population should be varied from 20 to 50, and the number of generations should be varied from 20 to 60. Performing these two simulations will assure that the proper genetic algorithm structure is being used in conjunction with each of the 5 implementations. At 5.57×10^8 computations per implementation, the overall simulations will require approximately 2.78×10^9 computations. Combing the man hours and computer hours, the genetic algorithm structure receives a score of 2.82.

4.4.2.4 HMM

Hidden Markov models exhibit average versatility. The concept of a HMM is that for a given hidden state, a visible state is observed where the visible state is governed by the evolutionary parameter format. The overall setup of a HMM is therefore driven by the format of the evolutionary parameter. A change in the format would therefore necessitate a change in the model. However, the model is designed to have both hidden and visible states specifically for this reason. Therefore the modifications of the evolutionary parameter are intuitive in nature. In order to implement, train, and test new a HMM structure approximately 1 man hour is needed. The implementation of approximately 3 structures would be needed to compare and find an optimal structure for the new evolutionary variable format. Adding in 2 hours for analyzing the simulation results, a HMM requires 5 man hours. The computational cost for the simulations would be on the order of 10^8; for this reason, HHM received full marks for computer hours. Combining the weighted score results in a score of 5.99 for HMM.

4.4.3 Model I/O Framework

The model I/O framework is an evaluation of implementation specifics with respect to the algorithms' ability to interact within a larger environment, and provide for multiple levels of control for the designer and the end VR user. Each algorithm will again be analyzed with respect to man hours and computer hours required to perform the given implementation. A typical or target value for the man hours is considered to be 5 hours. Two hours each for implementing and testing of both the input and the output, along with 1 hour for analyzing the simulations. The simulations required are approximately 5×10^7 calculations. The analysis of each of the four algorithms is described below.

Table 4.6 – Model I/O ratings

		Target	*Deterministic*	*ARMA*	*GA*	*HMM*
Man Hours	Value [hours]	5	7	10	10	5
	Score	0-1	0.714285714	0.5	0.5	1
	Weighted Score	6.66	4.757142857	3.33	3.33	6.66
Computer Hours	Value [computations]	50,000,000	2,000,000,000	85,370,000	4,463,000,000	500,000,000
	Score	0-1	0.025	0.585685838	0.011203227	0.1
	Weighted Score	3.33	0.08325	1.950333841	0.037306744	0.333
Model I/O	**Total Score**	**10**	**4.84**	**5.28**	**3.37**	**6.99**

4.4.3.1 Deterministic

The implementation of deterministic models is fairly robust in terms of communication of variables within the algorithm. However, using the current implementation there was not a multi-level structure with which the designer could set up; the controls to the algorithm were predefined. Building in multiple levels of control into a deterministic model would require approximately 3 hours defining the functions. Testing the model would take approximately 2 man hours, and the analysis of the simulation would also require 2 man hours. In order to assure that the multi-layer control was function properly many simulations would need to be performed. This would require 2 x 10^9 computations. Using these two variables deterministic modeling received a 4.84 for model I/O framework.

4.4.3.2 ARMA

The ARMA model implementation only requires setting a small number of inputs into the algorithm. The benefit of such an implementation is that the input structure could be used for a variety of applications with few changes needed. However, the model coefficients and the evolutionary variable are the only outputs from an ARMA model. It is seen that the simple structure of the model is both strength and a weakness. In order to set up a multi-layer I/O structure an external implementation is needed. At least 2 different structures should be explored and tested. Taking 3 hours to build and 1 to test, each implementation requires 4 man hours. After both structures are implemented it takes 2 man hours to analyze the simulations and finalize the structure. The total time required to build in a robust implementation for function I/O is 10 man hours. Assuming the number of poles and zeros is varied from 1 to 100, it would require 9.36 x 10^8 computations to perform the needed simulations of the I/O structure. The resulting score of ARMA models is 5.28

4.4.3.3 GA

The genetic algorithms are extremely complicated with respect to the implementation of the model I/O. The cost and mutation functions require a great deal of care when being implemented. The selection of sub-optimal parameters can result in the algorithm running prohibitively slowly or not providing meaningful results. To this end, the genetic algorithms are only as effective as the implementation parameters allow. This gives the designer a large amount of control, but the designer is required to have extensive knowledge of the functions being used. In order to properly incorporate the user and other I/O variables into the implementation at least 5 evaluation functions need to be analyzed. Similarly at least 3 different mutation schemes should be explored. At 1 hour each for development and testing, the various structures require 8 man hours. Additionally, 2 hours are required for analyzing the simulations. This is a total of 10 man hours. Assuming a population sweep from 20 to 50 and a generation number sweep from 20 to 60 the 8 different implementations require 4.46 x 10^9 computations. Using these two factors GA receive a model I/O framework score of 3.37.

4.4.3.4 HMM

Hidden Markov Models exhibit high model I/O framework. Unlike genetic algorithms, the set-up procedure requires little knowledge about the overall structure of the model, but rather knowledge of the phenomena being modeled. The visible and hidden state structure allows the designer to implement various control mechanisms for use by the user. Also, since the model is state-based it is conducive to conveying information to other algorithms about the current state or settings of the model. Since the process of incorporating the I/O into a HMM is intuitive it would not take more the 5 man hours to complete the required implementations. However, in order to assure the structure is suitable for user interaction and tolerant to poor input values, many simulations need to be performed. Since the model is probabilistic in nature, a high number of runs of the model will give a very good estimate of its performance. Assuming the test structure is subjected to 100 test runs, approximately 5 x 10^8 computations would be required to assure proper performance. These characteristics calculate to a 6.99 score for model I/O.

4.4.4 Speed

Since the speed and run-time considerations are hardware dependent features, the precise timing values are not as important as the overall computational complexities of each algorithm. Various implementation concerns and the ability to optimize the parallel distribution of the computations would affect a timing simulation. Therefore the speed ratings are in terms of computations as seen in Table 4.7 below.

Table 4.7 – Speed ratings

		Target	*Deterministic*	*ARMA*	*GA*	*HMM*
Prediction	Value [computations]	150,000	200,000	167,540	233,100	150,000
	Score	0-1	0.75	0.895308583	0.643500644	1
	Weighted Score	5	3.75	4.476542915	3.217503218	5
Correction	Value [computations]	7,000	20,000	16,754	233,100	20,000
	Score	0-1	0.35	0.417810672	0.03003003	0.35
	Weighted Score	5	1.75	2.08905336	0.15015015	1.75
Speed	**Total Score**	**10**	**5.50**	**6.57**	**3.37**	**6.75**

4.4.4.1 Deterministic

The deterministic algorithm shows average speed potential and receives an average score. The deterministic modeling simulations are instantaneous in the implementations described in Section 4.2.1. Since the rates are modeled as known growth rates there are no calculations that need to be performed. However in traditional deterministic modeling implementations there are specific functions that need to be evaluated. The speed of the evaluation will depend heavily on the numerical methods that are employed to solve the model. But typically a deterministic model would require 200,000 computations for a prediction function. Correction would only take about 20,000 computations. These two values result in a 5.5 rating for deterministic modeling.

4.4.4.2 ARMA

ARMA models exhibit above average speed considerations; even for high filter orders the computational burden is low. The majority of the computational time resides in the correction function. For filter orders up to 100 and 5 iterations the number of computations required for a correction is 16,754. The prediction step requires very few computations and is extremely time efficient. However, since the stability of the filter is not guaranteed, multiple iterations are often required to achieve desirable results. These factors become more pronounced during the iterations needed in performing multi-path evolution. A typical scenario assumes 10 regenerations of the filter coefficients, and therefore the prediction computational burden is 167,540 operations. Using these two factors ARMA models receive a 6.57.

4.4.4.3 GA

The genetic algorithm receives a below average score for speed. As mentioned earlier, the complexity of genetic algorithms is a benefit as well as a drawback. In order to achieve parameter correction a number of generations need to be calculated. As the number of generations calculated goes up, the time efficiency goes down. Similarly, a population exhibits more diversity when that population has a large number of individuals. However when there are more individuals it takes longer to evaluate the population and generate a new population. Assuming a population of 30 individuals evolved over 30 generations a GA requires 233,100 computations. This results in a speed rating of 3.37.

4.4.4.4 HMM

Hidden Markov models exhibit above average speed. The prediction calculation stage is time efficient as it only involves generating a hidden state and visible state transition. The prediction stage will therefore exhibit no more then 150,000 computations. When performing a correction function, there are a number of checks that need to be performed to ensure that the probabilities are balanced. Similarly, if the correction function requires learning, the computational costs will be high. On average a HMM would exhibit 20,000 computations. The resulting rating for HMM speed is 6.75.

4.4.5 Memory

Memory is the second evaluation criterion that is concerned with the resources required to perform the evolutionary computations. Memory usage is evaluated on both active and passive variables. Active variables include all the storage necessary for the actual evolutionary data, whereas the passive variables are the pointers, or counters required by the algorithm for translating the data. Since the active variables are storing the actual data they are given a 4:1 weight advantage over the passive variables. It is assumed that a 1 x 1 double size array requires 8 bytes of storage. Assuming 15 predictions, the target active memory requirement would therefore be 120 bytes; assuming one pointer variable the target passive memory requirement is 8 bytes. The table below illustrates how each algorithm performed with respect to the memory requirements.

Table 4.8 – Memory ratings

		Target	*Deterministic*	*ARMA*	*GA*	*HMM*
Active	Value [bytes]	120	200	160	3,600	144
	Score	0-1	0.6	0.75	0.033333333	0.833333333
	Weighted Score	8	4.8	6	0.266666667	6.666666667
Passive	Value [bytes]	8	40	8	8	16
	Score	0-1	0.2	1	1	0.5
	Weighted Score	2	0.4	2	2	1
Memory	**Total Score**	**10**	**5.20**	**8.00**	**2.27**	**7.67**

4.4.5.1 Deterministic

Deterministic modeling approaches receive an average rating for memory usage. Typically the models require a certain number of coefficients and variables that need to be set for each prediction. For complicated models the number of variables may be high, but on average there is a 5:3 ratio for variables to predictions and a 1:3 ratio for pointers to predictions. Fifteen predictions would therefore require 200 bytes of active memory, and 40 bytes of passive memory. These values calculate to a score of 5.2.

4.4.5.2 ARMA

The ARMA models exhibit very high memory performance. The only storage required for the ARMA model is the storage of the coefficients. The models required filter orders of 10, and therefore the active memory requirement is 20 variables or 160 bytes. A single counter is needed to keep track of the current prediction and therefore ARMA models require 8 bytes of passive memory. These two memory requirements total a score of 8.0.

4.4.5.3 GA

As mentioned earlier, one of the drawbacks of genetic algorithms is the amount of computations required. In order to achieve a single path of evolution, an entire population needs to be created. Since only one of those paths is going to be selected there is a lot of memory overhead that is required. Assuming 15 predictions and a population size of 30 there are 450 1x1 arrays needed. This translates to 3600 bytes of active information. The only passive requirement is a pointer to a given individual within the population. The memory score for genetic algorithms calculates to 2.27.

4.4.5.4 HMM

Hidden Markov models require low amounts of storage. The storage requirements of the algorithm are the hidden and visible state transitional probabilities. Assuming 3 hidden states and 3 visible states, 18 probabilities are needed. These probabilities require 144 bytes of active memory. A pointer is required to indicate both the hidden state and the visible state; therefore 16 bytes of passive memory are needed. Since the model requires little overhead and simplistic storage, the HMM receives a 7.67.

4.4.6 Interaction with User

The interaction with the user is defined as the ability to interact with the algorithm and the overall effectiveness of that interaction. An interaction score of 0 means that the algorithm has little or no ability to accept input from the user, and the limited input that is accepted has little effect on the overall process. A score of 5 means the algorithm accepts meaningful and intuitive input. The input exhibits adequate of control over the algorithm. A score of 10 means that the algorithm has ideal control inputs and that the inputs offer a high degree of control over the computations of the algorithm. The graph below illustrates the performance of each of the four algorithms.

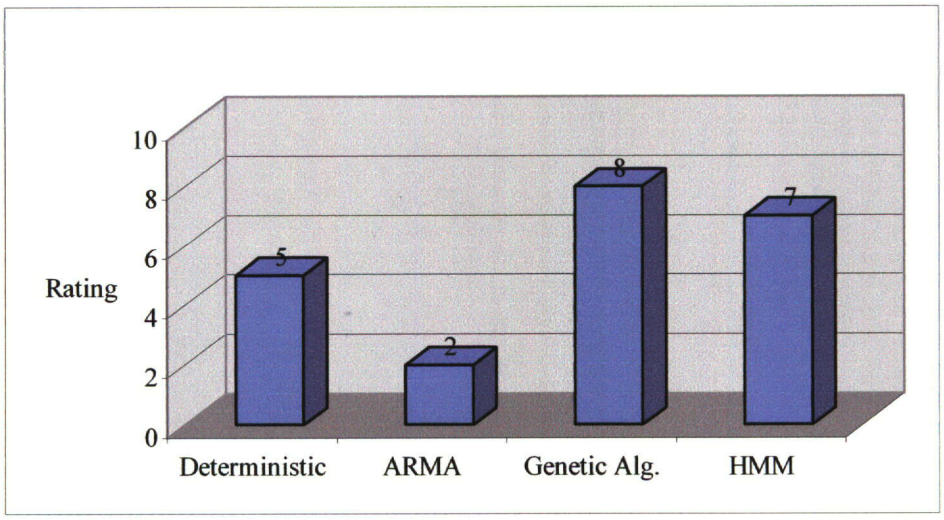


Figure 4.38 – User interaction ratings.

4.4.6.1 Deterministic

The deterministic algorithm exhibits average interaction. Deterministic models in general are models for specific phenomena. Certain environmental assumptions and outside input factors will be present within the model. By controlling the inputs factors to the deterministic model, the model will reflect the output accordingly. The drawback to the interaction with deterministic models is that the model will follow a certain pre-described set of rules. The controls however over the model are meaningful and therefore deterministic algorithms receive an average rating of 5.

4.4.6.2 ARMA

The user interaction with ARMA models is below average. Although the ARMA model was a combination of input and output samples, the input does not allow the user to have a great deal of control over the output for a given sample. The control is very indirect; the only variable that the user has control over is the input variable. There is no mechanism in traditional ARMA modeling for the user to have control over the filter itself. Since the user had only limited control and the extent of that control is also limited ARMA models received a rating of 2.

4.4.6.3 Genetic Algorithms

The overall interaction with the user for genetic algorithms is above average. Using a genetic algorithm model the user has a great deal of control over the evolutionary process. Through the use of a cost function the user can vary the functional evaluation of the population over time. Similarly, the user can control the mutation rates such that the user can alter the probability of dramatic jumps within a population. Each of these factors empowers the user with meaningful tools in the evolutionary process. Information hiding allows the programmer to build variables into the algorithm such that the end-user will have inputs that correspond to these variables within the main computations of the algorithm. Since the user has a wide array of control inputs that produce effective and powerful control over the evolutionary process genetic algorithms receive an 8.

4.4.6.4 HMM

Hidden Markov models exhibited an above average amount of control. The overall structure (number of hidden nodes, visible states, transitions) is preset before the evolutionary process begins. Adaptive nodal structures can be implemented, but they provide an added level of complexity. However, through the transitional probabilities the user can control the probability of certain phenomena occurring. Through the use of variable transitional probabilities the user is given a high degree of control. The control is not an immediate force to a certain state, but rather the probability of it being in that state. Although the user has some amount of control, the precise model structure will still be an influence on the output. HMMs receive a score of 7.

4.4.7 Software Specific

The software specific criteria deal with the ties an algorithm may have to a software platform. If a model has strong ties to specific software such that the model cannot be implemented or incorporated into any other environment it receives a score of 0. A score of 5 means that the model has some ties to a software platform but implementation of the model into a different platform would be possible. A score of 10 reflects an algorithm that would perform identically on any system regardless of the environment that it was operating in. The score for each of the models is illustrated in the graph below.

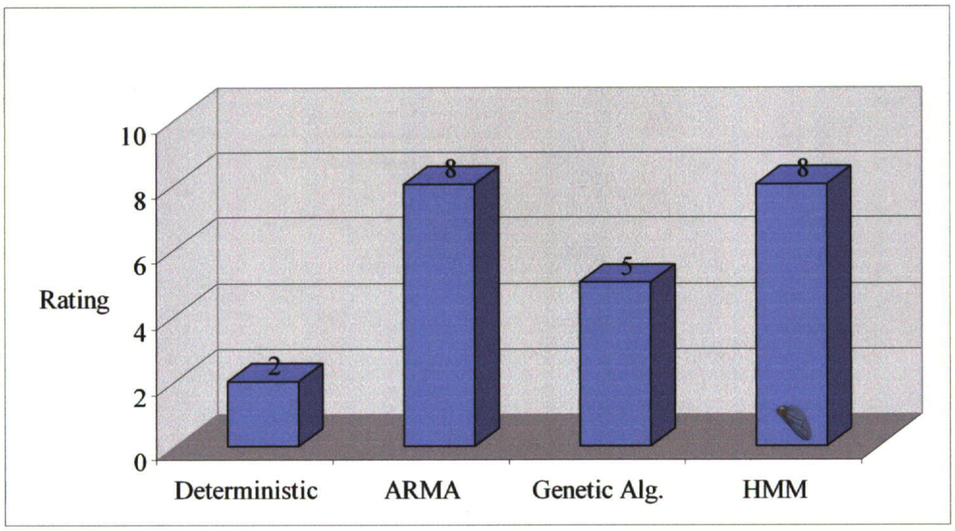


Figure 4.39 – Software specific ratings.

4.4.7.1 Deterministic

In general, deterministic algorithms are relatively software dependent. The deterministic implementation performed in Section 4.2.1 does not capture the complete nature of deterministic modeling. Therefore, the implementation results from the deterministic modeling cannot be used to estimate the software specific nature of the algorithm. In order to assess the software specific nature of the algorithms, specific deterministic model implementations were investigated. The models developed by Southwest Research Institute (SwRi) NESSUS, SAFE, and DARWIN are representative examples [49]. When implementing these models a proprietary program was developed. In addition to the program, specific functions were also proprietary. Integrating such a program into a VR environment would require transporting the whole program and adding an interface. The algorithm's ties to the program are extremely high. Since a significant number of deterministic models have proprietary functions, the deterministic models received a rating of 2.

4.4.7.2 ARMA

ARMA models are above average for software specific implementations. In order to implement the ARMA models, built-in MatLab functions were used. The built in MatLab functions provide an accurate and time-efficient method for computing the required parameters. Even if taken out of MatLab the specialized functions would not be difficult to encode. Since a filter is relatively easy to code on any platform ARMA models received a rating of 8.

4.4.7.3 Genetic Algorithms

The genetic algorithm implementation is not software specific. The implementation could be imported into a variety of other simulation tools in order to accomplish the evolution. However, the implementation in MatLab takes advantage of the parallel processing capabilities of the software. If imported into an environment where this is not possible or prudent, then the efficiency of the code would drop significantly. For this reason, the genetic algorithm implementation received a software specific rating of 5.

4.4.7.4 HMM

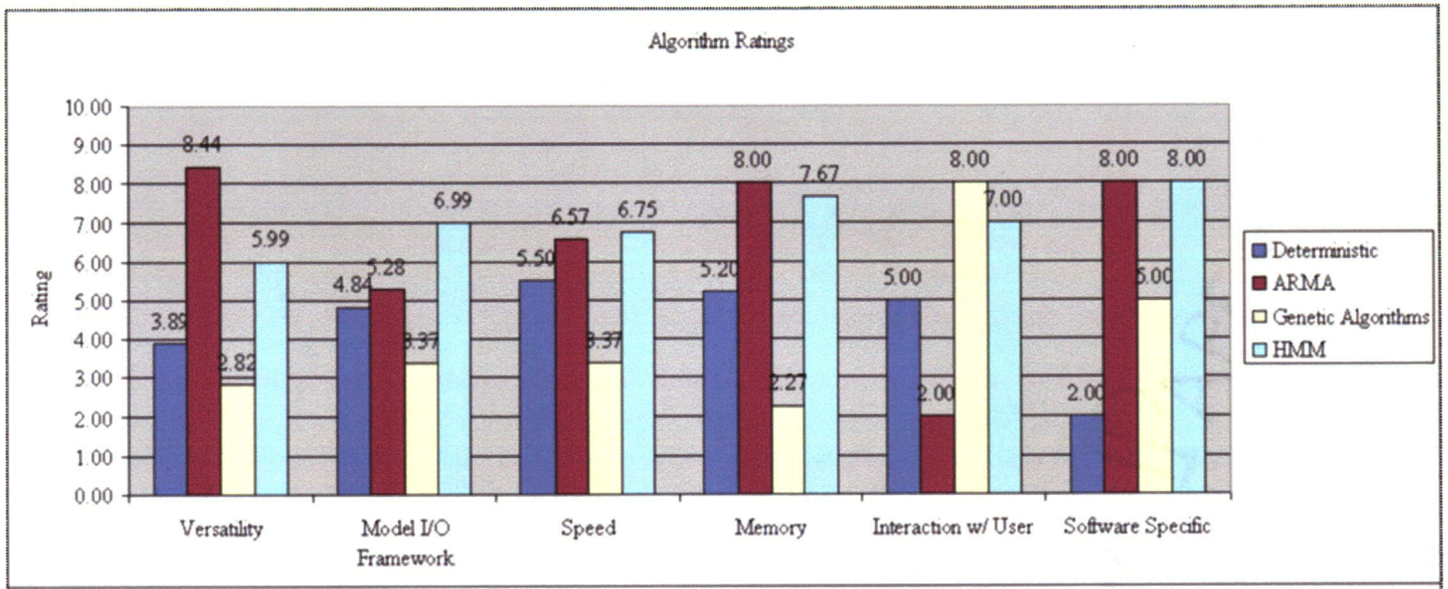
The HMM implementation has above average software specific characteristics. There are no proprietary functions that are required for the algorithm to be implemented. The algorithm runs from simple commands and the generation of random numbers. Random number generation can be performed on any platform and therefore the HMM received an above average rating for software specific characteristic of 8.

4.5 Evaluation

Based on the descriptions above, a chart is compiled from the ranking of each algorithm for the various evaluation criteria. The rankings of each algorithm according to the various criteria are plotted in Figure 4.40 (a).

The chart shows that the deterministic algorithm did not exhibit a strong performance in any given category. For most of the criteria it exhibited an average to below average performance. The ARMA model received a high rating for versatility, speed, memory, and software specific, but a low rating for user interaction. Genetic algorithms were below average for a number of criteria but performed highly with respect to user interaction. Finally, HMMs performed average to above average across the entire range of evaluation measures.

(a)



(b)

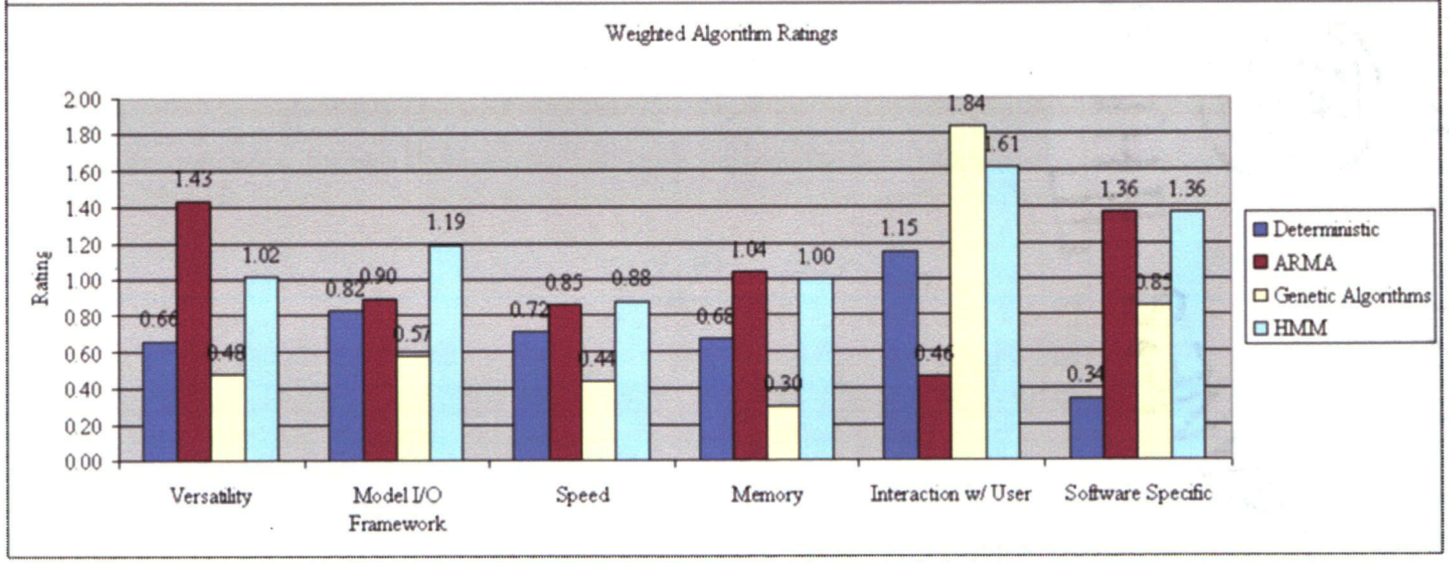
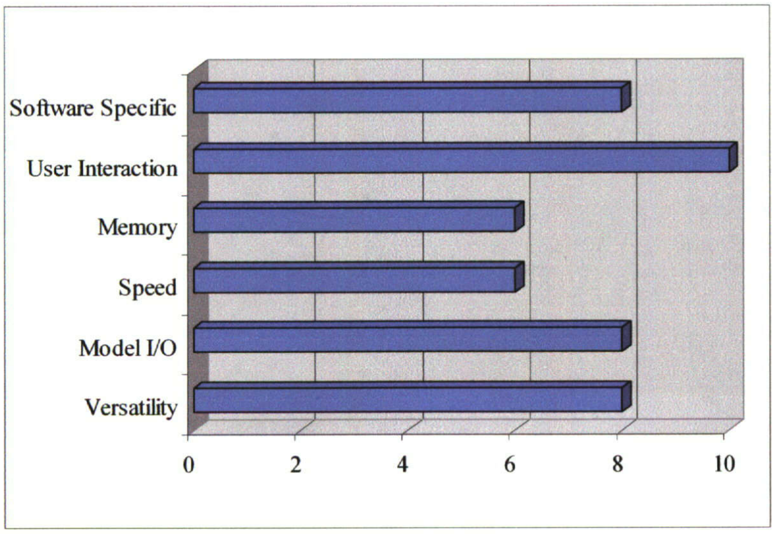
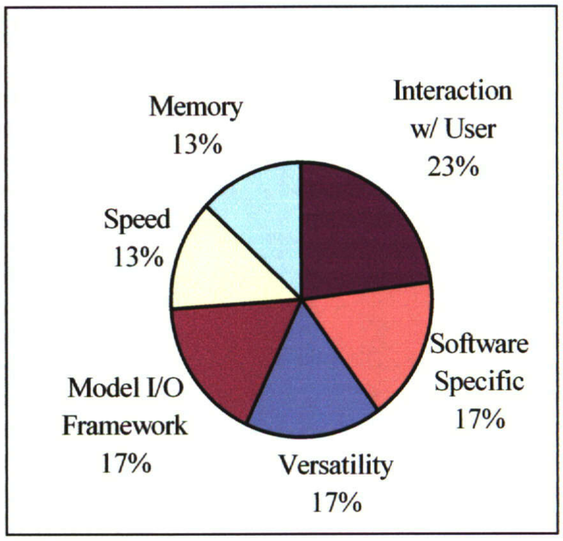


Figure 4.40 – Algorithm Ratings.

Next, the evaluation factors are weighted according to importance. The weighting assignments are seen in Figure 4.41. The weight assignments are based on the importance of the criteria to VR implementations. Figure 3.7 illustrates the relationship between the evaluation criteria and the generalized algorithms goals. Combined, memory and speed form the largest weight because together they will influence the feasibility and practicality of implementing the given algorithm in VR. User interaction is ranked the highest because the user interaction with the algorithm is an essential part of the entire process. Without user interaction, the analysis would suffer in a VR implementation. Finally, model I/O, versatility, and software specific needs are all given equal weights because they each have equal but different impacts on the VR implementation.





(a) **(b)**

Figure 4.41 – Weight assignments.

According to the seven tools of management the rankings in Figure 4.40 (a) can be scaled according to the weights in Figure 4.41 to assure a fair and thorough comparative analysis. The resulting scores are displayed in Figure 4.40 (b). For each algorithm, the evaluation factors are normalized, summed, and displayed in Figure 4.42.

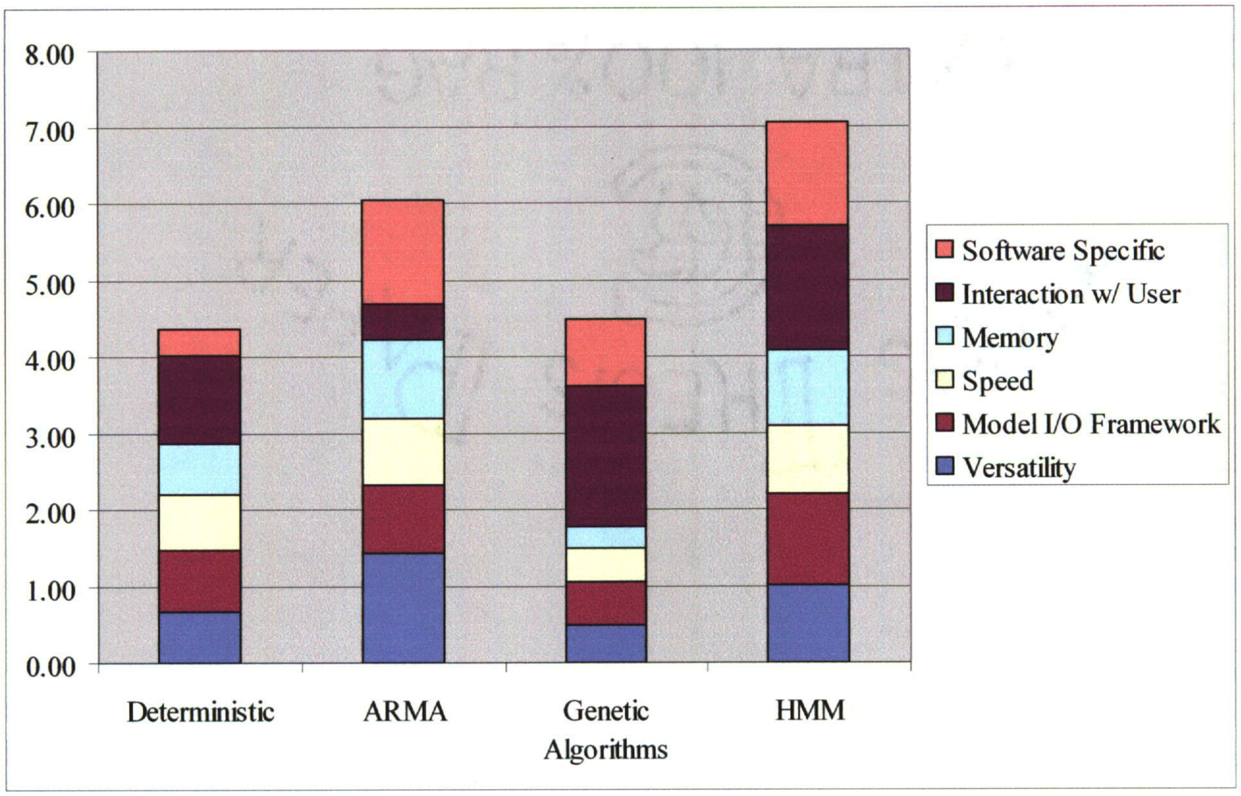


Figure 4.42 – Weighted comparison.

From the weighted comparison it can be seen that hidden Markov models rank highest for performing evolutionary computations in VR, followed by ARMA, genetic algorithms and finally deterministic algorithms. The margin between the algorithms is not sufficient to conclude that any one algorithm is superior to the rest for all VR evolutionary applications. There are tradeoffs that exist between the algorithms, but overall HMMs are efficient across the range of evaluation criteria. For each algorithm, the total score is plotted on a percentage scale to determine which criteria influenced the overall score the greatest. This analysis plot is illustrated in Figure 4.43 below.

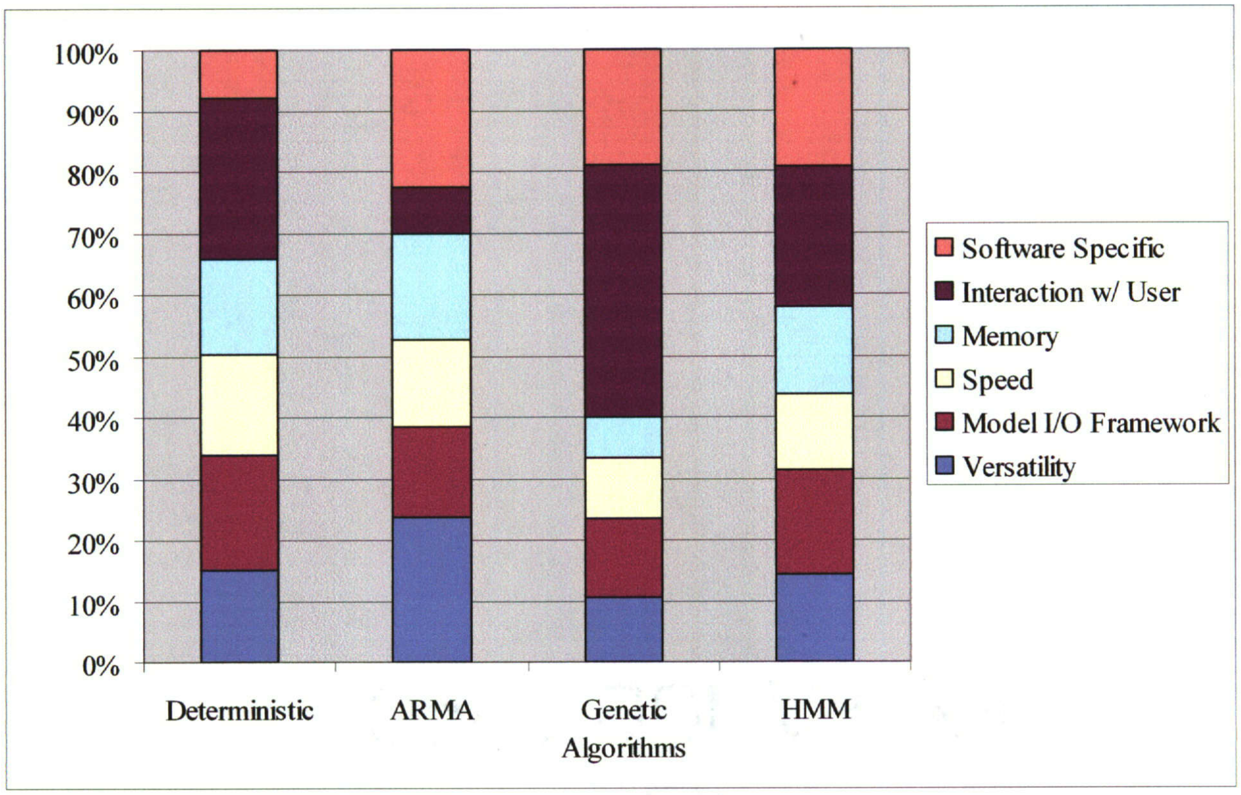


Figure 4.43 – Contributions to weighted score.

From Figure 4.43 is can be seen that deterministic modeling has relatively average performance across the range of evaluation criteria. Deterministic modeling techniques are often software specific and perform a very exact analysis. Often for VR risk analysis a probabilistic approach is more suitable. Versatility, memory and software specific needs are seen to contribute over half of the ARMA model's score. ARMA models are simple to implement, and computationally efficient. The drawback to such models is the lack of mechanisms for the algorithm to interact in the overall virtual environment. Genetic algorithms are seen to exhibit the opposite behavior of ARMA models. Genetic algorithms afford a vast array of possibilities for interactions, but the computational cost of those interactions is relatively high. Finally, HMMs exhibit average to above average performance in all categories. The problem of VR evolutionary programming evolves a large array of variable conditions. The purpose of this analysis is not to conclude that one algorithm is superior to all others, but illustrate the various tradeoffs among the different algorithms when a VR platform is implemented.

CHAPTER 5 – CONCLUSIONS

Virtual reality has emerged as a powerful visualization tool for design, simulation and analysis in modern complex industrial systems. As the hardware and software capabilities inside VR systems mature, it is essential that the design, simulation and analysis techniques that are deployed in VR environments keep pace. The primary motivation for this thesis is to develop a framework for the effective use of VR in design-simulation-analysis cycles. Of particular interest are situations involving large, complex, multidimensional data-sets. The framework developed in this thesis is intended to support not only the integration of such data for visual, interactive, and immersive displays, but also provides a method for performing risk analysis. The principal objectives of this thesis are revisited below:

1. *To develop a framework for multi-sensor data visualization in virtual reality.*
2. *To develop a method for implementing time-evolutionary models for virtual reality.*
3. *To compare the suitability of the various algorithms for virtual reality scenario development.*

The methods developed for meeting these objectives were demonstrated using the following environment – the in-line, nondestructive evaluation of gas transmission pipelines. This environment allowed for incorporating graphical, functional and measurement data. Additionally, the environment had an evolutionary component capable of accepting user input and varying over time.

5.1 Summary of Accomplishments

This thesis has attempted to address the objectives described in the previous section as follows:

1. Development of a framework for multi-sensor data visualization in VR. An immersive data fusion environment was defined and created, consisting of (a) graphical data, (b) measurement data, and (c) functional data. Incorporating user-defined inputs allowed the user to *interact* and *navigate* in the environment. The framework was demonstrated using multi-sensor data obtained during the in-line inspection of a section of a gas transmission pipeline. Graphical models of the pipeline components were displayed in addition to MFL / UT inspection signals, neural network predictions of pipeline condition, and the geographic location of the pipeline network.
2. Development of time-evolutionary models for 1-D, 2-D, 3-D, and *N*-D VR evolutionary simulations. Four candidate algorithms were implemented – deterministic modeling, auto-regressive moving average modeling, genetic algorithm modeling, and hidden Markov modeling. The results of these evolutionary techniques were demonstrated using the growth of a stress-corrosion crack on a pipeline section.
3. Comparison of the evolutionary algorithms for scenario development. The following evaluation criteria were identified – versatility, model I/O, speed, memory, interaction with the user, and software specific features. A prioritization matrix technique was used to rate the candidate algorithms and identify strengths, weaknesses, and trade-offs.

5.2 Conclusions

Virtual reality environments show considerable promise for the integration of multi-sensor data. The system allows a user to rapidly sift through large and complex data sets to isolate features of interest. Additionally, the VR environment has the ability to evolve as a function of both system data and user input. The use of data integration and evolution empowers the user to evaluate scenarios to make informed decisions.

Four algorithms were implemented and analyzed as candidates for single and multi-path evolution. Deterministic modeling for evolution on a virtual platform showed that the technique lacked certain key features to allow it to stand out as the prime candidate for evolution. ARMA modeling is a versatile and computationally efficient method for providing signal prediction. However, ARMA models did not exhibit a high level of user interaction, which is a major drawback of this technique. Conversely, genetic algorithm modeling showed great promise with respect to user interaction. However, the technique has large computational costs. Finally, hidden Markov models exhibited strong characteristics in all of the evaluation categories.

5.3 Recommendations for Future Work

As follow-on to this work, an implementation of these algorithms could be performed to further address the design-simulation-analysis cycle for virtual prototyping. The algorithms could be implemented in conjunction with one another to capitalize on their strengths and downplay their weaknesses. For example, a HMM could be used to generate multiple solutions to a problem. The user could evaluate a small number of these solutions as input into a genetic algorithm. Such an implementation would allow for VR to become an even more powerful tool in the development cycle.

With respect to the in-line inspection of natural gas pipelines, additional algorithms could be developed to further integrate heterogeneous data types. Information logs about past activity and past performance could be used to aid in the risk assessment procedure. Also, functional models could simulate multiple levels of evolution. The interaction of a number of evolutionary models would allow for complex evolutions to take place across a range of phenomena within the virtual environment.

In addition to improving the VR design environments, software platforms that allow for real-time evolutionary parameter computation and display must be investigated.

References

[1] M. Slater, A. Steed, Y. Chrysanthou, *Computer Graphics and Virtual Environment*, Addison Wesley, 2002.

[2] A. van Dam, A. Forsberg, D. Laidlaw, J. La Viola and R. Simpson, "Immersive VR for Scientific Visualization: A Progress Report," *IEEE Computer Graphics and Applications*, pp. 26-52, November-December 2000.

[3] A. van Dam, D. H Laidlaw, R. M. Simpson, "Experiments in Immersive Virtual Reality for Scientific Visualization," *Computers & Graphics*, Vol. 26. pp. 535-555, 2002.

[4] T. Hong, *A Virtual Environment for Displaying Gas Transmission Pipeline NDE Data*, M.S. Thesis, Iowa State University, 1997.

[5] M. J. Schuemie, P. Van Der Straaten, M. Krijn and C. A.P.G Van Der Mast, "Research on Presence in Virtual Reality: A Survey," *CyberPsychology & Behavior*, Vol. 4, No. 2, pp. 183-201, 2001.

[6] L. G. Shapiro, G. C. Stockman, *Computer Vision*, Prentice Hall, Upper Saddle River, NJ, 2001.

[7] Y. Waern, *Cognitive Aspects of Computer Supported Tasks*, John Wikey & Sons, Chichester, 1989.

[8] M. J. Pierce, *Scientific visualization of gas transmission pipeline NDE data*, M.S. Thesis, Iowa State University, 1999.

[9] Kraus, and Fleisch, *Electromagnetic with Applications*, Fifth Edition, McGraw-Hill, Boston, 1999.

[10] R. Beavon, *Chemistry Solutions: F.A. Kekule von Stradonitz*, October 2003.

[11] Encyclopdeia.com. *Human History*. Accessed March 2004.

[12] J. Cremer, J. Kearney, and H. Ko, "Simulation and Scenario Support for Virtual Environments," *Computer & Graphics*, Vol. 20, No. 2, pp 199-206, 1996.

[13] R. Gonzalez, R. Woods, *Digital Image Processing*, Second Edition, Prentice Hall, Upper Saddle River, NJ, 2002.

[14] S. Pinker, *How the Mind Works*, W. W. Norton & Company, New York, 1997.

[15] Ed., M. L. McLaughlin, J. P. Hespanha, G. S. Sukhatme, *Touch in Virtual Environments*, Prentice Hall, Upper Saddle River, NJ, 2002.

[16] D. E. Bray, R. K. Stanley, *Nondestructive Evaluation, A Tool in Design, Manufacturing, and Service,* Revised Edition, CRC Press, Boca Raton, 1999.

[17] R. Miller, *Nondestructive Testing Handbook: Acoustic Emission Testing*, American Society for Nondestructive Testing, Inc. Vol. 5, 1987.

[18] A. S. Birks, *Nondestructive Testing Handbook: Ultrasonic Testing*, American Society for Nondestructive Testing, Inc. Vol. 7, 1991.

[19] D. L. Hall, J. Llinas, *Handbook of Multisensor Data Fusion*, CRC Press, Boca Raton, 2000.

[20] S. Mandayam, L. Udpa, S. S. Udpa and W. Lord, "Signal processing for in-line inspection of gas transmission pipelines," *Research in Nondestructive Evaluation*, Vol. 8, No. 4, pp. 233-247, 1996.

[21] C. Michael, *An invariance transformation algorithm for defect characterization of ultrasonic signals for the nondestructive evaluation of concrete*, M.S. Thesis, Rowan University, 2002.

[22] G. J. Posakony, and VJ Hill , "Assuring the integrity of natural gas transmission pipelines," *Topical Report*, Gas Research Institute, November 1992.

[23] VRCO, Inc., *vGeo Reference Manual*, Version 2.0, January 2003.

[24] W. Kenong, "3-D shape approximation using parametric geons," *Image & Vision Computing*, Vol. 15, pp. 143-158, 1997.

[25] Peiya Liu, "The Virtual Reality Modeling Language Explained," *IEEE Multimedia*, pp. 84-93, July/September 1998.

[26] Nahm, S. Lee, J. H. Park, K. S. Park, "Reality and human performance in a virtual world," *International Journal of Industrial Ergonomics*, Vol. 18, pp. 187-191, 1996

[27] C. Ware, and G. Franck, "Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions," *TOG Info Net Vis.*, pp. 1-21, April 20, 2000.

[28] C. Luo, "Multi-sensor Fusion and Integration: Approaches, Applications, and Future Research Directions." *IEEE Sensors Journal*, Vol. 2, No 2, pp 107-119, April 2002.

[29] J. Vora, S. Nari, A. K. Gramopadhye, A. T. Duchowski, B. J. Melloy, B. Kanki, "Using virtual reality technology for aircraft visual inspection training: presence and comparison studies," *Applied Ergonomics*, Vol. 33, pp. 559-570, 2003.

[30] H. Anton, *Calculus*, 6th Edition, Wiley, New York, 1999.

[31] E. Kreyszig, *Advanced Engineering Mathematics*, 8th Edition, Wiley, New York, 1999.

[32] B. Porat, *A Course in Digital Signal Processing*, John Wiley and Sons Inc., USA, 1997.

[33] User's Guide, *Signal Processing Toolbox for MatLab,* Version 5, Parametric Modeling, 2000.

[34] Ed., Lawrence Davis, *Handbook of Genetic Algorithms*, Thomson Computer Press, USA, 1996.

[35] T. Collins, "Applying software visualization technology to support the use of evolutionary algorithms," *Journal of Visual Languages and Computing*, Vol. 14, pp 123-150, 2003.

[36] H. Pohlheim, "Visualization of Evolutionary Algorithms – Real-World Application of Standard Techniques And Multidimensional Visualization," *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA., 1999.

[37] H. Pohlheim, "Visualization of Evolutionary Algorithms – Set of Standard Techniques and Multidimensional Visualization," *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA., 1999.

[38] I. Kushchu, "Genetic Programming and Evolutionary Generalization," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 5, pp. 431-442, October 2002.

[39] D. Whitley, "An overview of evolutionary algorithms: practical issues and common pitfalls," *Information and Software Technology*, Vol. 43, pp. 817-831, 2001.

[40] R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification*, Second Edition. Wiley-Inter-science, 2001.

[41] J. J. Ford, J. B. Moore, "Adaptive Estimation of HMM Transition Probabilities," *IEEE Transactions on Signal Processing*, Vol. 46, No. 5, pp 1374-1385, May 1998.

[42] The American Heritage Dictionary of the English Language, Fourth Edition, Houghton Mifflin Company, 2000.

[43] J. V. Michaels, *Technical Risk Management*, Prentice Hall, Upper Saddle River, NJ, 1996.

[44] J. D. Andrews, and T. R. Moss, *Reliability and Risk Assessment*, Second Edition, ASME Press, New York, 2002.

[45] T. Bedford, R. Cooke, *Probabilistic Risk Analysis; Foundations and Methods*, Cambridge University Press, 2001.

[46] D. H. Besterfield, *Quality Control*, Fifth Edition, Prentice Hall, Upper Saddle River, NJ, 1998.

[47] V. E. Sower, M. J. Savoie, and Stephen Renick, *An Introduction to Quality Management and Engineering*, Prentice Hall, Upper Saddle River, NJ, 1999.

[48] W. T. Riddell, "Probabilistic Growth of Complex Fatigue Crack Shapes: Toward Risk Based Inspection Intervals for Railroad Tank Cars," *Journal of Mechanical Design*, Vol. 123, December 2001.

[49] SwRI. www.nessus.swri.org. NESSUS accessed January 2004.

[50] S. Haykin, *Adaptive Filter Theory*, Fourth Edition, Prentice Hall, 2002.

[51] J. M. Carroll, *Making Use: scenario-based design of human-computer interactions*, MIT Press, Cambridge Massachusetts, 2000.

[52] L. Rosenblum, and M. Macedonia, "Projects in VR: Integrating VR and CAD," *IEEE Computer Graphics and Applications*, September/October 1999.

[53] Y. Yang, X. Wang, and J. X. Chen, "Rendering Avatars in Virtual Reality: Integrating a 3D Model with 2D Images," *Computing in Science & Engineering*, pp. 87-91, January/February 2002.

[54] J. Campos, K. Hornsby, M. J. Egenhofer, "A model for exploring virtual reality environments," *Journal of Visual Languages and Computing*, Vol. 14, pp. 471-494, 2003.

[55] L. Udpa, S. Mandayam, S. Udpa, Y Sun, and W. Lord, "Developments in Gas Pipeline Inspection Technology," *Materials Evaluation*, pp. 467-472, April 1996.

[56] VRCO, Inc. *vGeo User's Guide*, Version 2.0. January 2003.

[57] D. Bowman, *Interaction Techniques for Common Tasks in Immersive Virtual Environments: Design, Evaluation and Application.* Georgia Institute of Technology. Thesis, June 1999.

[58] F. P. Brooks, Jr., "What's Real About Virtual Reality." *IEEE Computer Graphics and Applications*, pp. 16-27, November/December 1999.

[59] J. G. Proakis, D. G. Manolakis, *Digital Signal Processing, Principles, Algorithms, and Applications*, Third Edition, Prentice Hall, Upper Saddle River, NJ, 1996.