VIRTUAL-EXPERIMENT-DRIVEN PROCESS MODEL (VEDPM)

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Chin Aik Lua

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

April 2010

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Virtual-Experiment-Driven Process Model (VEDPM)

By

Chin Aik Lua

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Lua, Chin Aik, Ph.D., Department of Computer Science, College of Science and Mathematics, North Dakota State University, April 2010. Virtual-Experiment-Driven Process Model (VEDPM). Major Professor: Dr. Kendall Nygard.

Computer simulations are the last resort for many complex problems such as swarm applications. However, to the best of the author's knowledge, there is no convincing work in proving "What You Simulate Is What You See" (WYSIWYS). Many models are built on long, subjective code that is prone to abnormalities, which are about corrupted virtual scientific laws rather than software bugs. Thus, the task of validating scientific simulations is very difficult, if not impossible. This dissertation provides a new process methodology for solving the problems above: Virtual-Experiment-Driven Process Model (VEDPM). VEDPM employs simple yet sound virtual experiments for verifying simple, short virtual laws. The proven laws, in turn, are utilized for developing valid models that can achieve real goals. The resulted simulations (or data) from proven models are WYSIWYS. Two complex swarm applications have been developed rigorously and successfully via VEDPM--proving that VEDPM is workable. In addition, the author also provides innovative constructs for developing autonomous unmanned vehicles--swarm software architecture and a modified subsumption control scheme, and their design philosophies. The constructs are used repeatedly to enable unmanned vehicles to switch behaviors autonomously via a simple control signal.

ACKNOWLEDGEMENTS

I would like first to thank my family who supported my journey through NDSU, especially my parents, Lua Keap Meow and Lim Teng Lan; my brothers, Lua Tiong Aik, Lua Toong Aik, and Lua Beng Ex; and my sister, Lua Wan Chin. I also want to thank Dr. Kendall E. Nygard, my academic advisor, for his support all these years, Dr. Karl R. Altenburg for his research advice, and my committee members, Dr. Weiyi (Max) Zhang, Dr. Juan (Jen) Li, and Dr. Chao You.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1. Motivation

The main motivation of this dissertation is to propose a sound process methodology for developing scientific simulations. This methodology utilizes the scientific method in virtual experiments for validating complex simulations such as a swarm of Unmanned Air Vehicles (UAVs) flying in proximity of each other. The verified simulations are realistic--that is, What You Simulate Is What You See (WYSIWYS). The current process models fail to address the unique problems posed by scientific simulations. Climate scientists, for instance, cannot ask the critics to accept their climate simulations due to the following doubts:

- How can the "climate" scientists, who are not programmers, prove that their simulations are reflecting real-world's sciences?
- How can they be sure the simulation code is not corrupted?
- How can they ensure the virtual climate models are sound?

The climate simulations address a serious global crisis and the researchers spent countless hours in perfecting their models. Their effort may not be rewarded if one cannot tell the differences between the models and animations from Pixar studios.

Scientific simulations are important tool and often the last resort for many complex problems. Despite that, there are no rigorous steps in applying the scientific method to the virtual world. The real experiments implementing the scientific method have the following characteristics:

- The experimental variables must be isolated, tested, and measured. For unambiguous result, each variable, if separable, is targeted in a unique laboratory setting demanded by its properties.

- The data obtained from the experiments are related to the variable being examined.
- Unexpected results are the norm of experiments and experimenters must modify their models according to new discoveries.
- The results are repeatable.

A new process methodology, called Virtual-Experiment-Driven Process Model (VEDPM), is proposed by this author to mimic real experiments and utilize the scientific method. VEDPM, depicted in Figure 1, has the following unique characteristics that are not found in current process models:

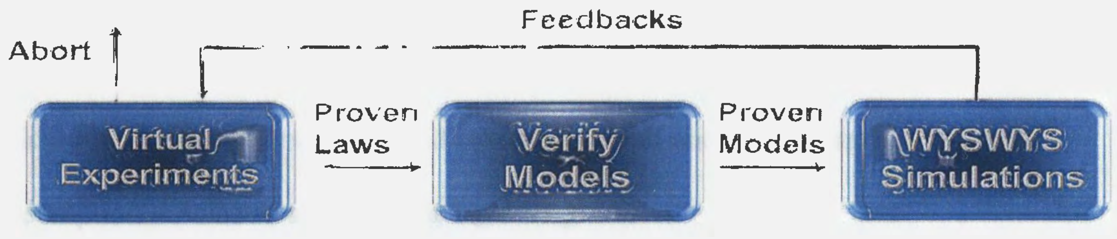


Figure 1. Main Characteristics of VEDPM

1. Virtual experiments--before any virtual model is developed, VEDPM emphasizes the virtual laws (e.g. laws that define forces) that govern a model must be identified and proven in virtual experiments. If the laws or experiments are not identified or constructed soundly, the relevant simulation is then not feasible--that is, the experimenters can abort the simulations in this phase to save costs. Simple and innovative virtual experiments are designed for proving each law. The proven laws should be few in number and short in code, and none of them is discarded or modified.

2. Verify models--they emphasize that the model cannot create its own laws but can utilize all the proven ones. If the model follows them faithfully, it is then proven.
3. WYSIWYS simulations--they highlight that the proven model, in turn, provides high-fidelity simulations and/or data as well as feedbacks to improve the model.

VEDPM is discussed more in depth in Chapter 4. Two complex real swarm applications (Chapter 6 & 7), utilizing the principles of VEDPM, have been developed successfully to support the claims of VEDPM. Swarm behaviors (Chapter 5) proposed by the author, consisting of swarm software architecture and a subsumption control scheme, are also needed for the applications. The software architecture from the left diagram of Figure 2 describes how behavior models are integrated tightly with virtual experiments for validation. The behaviors are refined into hierarchical ones and controlled by the subsumption scheme as shown in the right diagram.

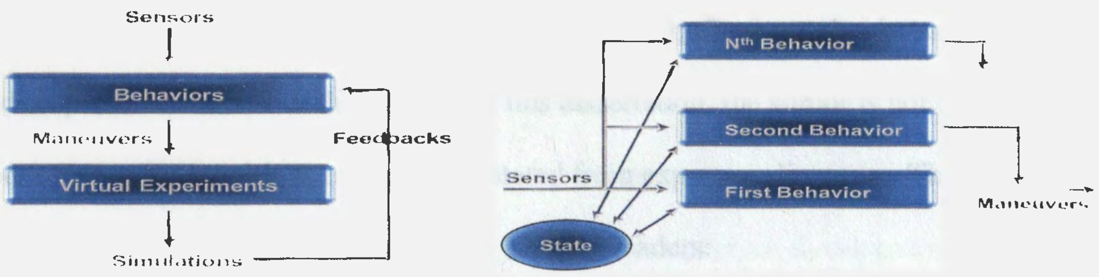


Figure 2. Swarm Software Architecture and Subsumption Control Scheme

The author utilized the constructs above repeatedly to develop the first switchable swarm behaviors as demonstrated in Chapter 6 and robust Tetwalkers' behaviors in Chapter 7. Other contributions are design philosophies of VEDPM and swarm behaviors.

1.2. The Objectives of This Dissertation

The author wants to achieve the following objectives:

1) To show that VEDPM is a powerful tool for proving scientific simulations.
2) To demonstrate that scientific method like virtual experiments can be constructed via VEDPM.
3) To illustrate virtual laws need innovative minds to create.
4) To prove that VEDPM is feasible via two applications.
5) To build WYSIWYS simulations.
6) To provide general software architecture and a modified subsumption control scheme for swarm behavior model.
7) To demonstrate that the autonomous behaviors can be switched via a simple control signal.
8) To show that emergent intelligence is possible at swarm level.
9) To explain that behavior models coupled with VEDPM are powerful tools for developing applications for unmanned vehicles.

1.3. Working Definitions for the Terms Used

Definitions are important to avoid unnecessary arguments and enhance the clarity of ideas presented here. For the purpose of this dissertation, the author is not concerned about other strict definitions expressed or stated from experts or literature. The author defines them “loosely” to convey the main ideas to readers:

1. Virtual experiments--these are the central concept of this dissertation on how to mimic real experiments with scientific method being applied soundly to the virtual environments.
2. Virtual-Experiment-Driven Process Model (VEDPM)--a new process methodology to ensure a realistic simulation.

3. Virtual laws--they are computing version of physical laws with discrete property. They are proven via virtual experiments.
4. Emergent intelligence--the overall group behaviors appear to be "intelligence" though the participants may not aware of it. An individual goose, for example, takes local cues in lining up a flying formation from neighboring geese, and the desired formation seems "intelligently." The author does not believe the swarm intelligence is comparable to human's one.
5. Autonomous behaviors--these robust and independent actions are not preplanned.
6. Local cues--they are hints provided locally and cannot be foreseen by central planners. The geese, for instance, get local hints from neighboring ones in flying formation, which may not be visible to the farthest member geese of the group.
7. Reactive behaviors--they are uncontrolled reflexive actions like hands recoil from hot surfaces. For swarm agents, these can be behaviors induced automatically from sensors or control signals.
8. Software verification and validation (V&V)--if the developed software meets right requirements, then validation is achieved. If the right product is produced, then verification is done. The differences between verification and validation are unimportant for practitioners except to the theorist. For the purpose of this dissertation, the author uses these terms to mean the laws or models have been proven via VEDPM. It serves no purpose to verify the

correctness of source code in VEDPM as the "right" code may not produce the right laws. For this reason, the traditional V&V are not employed in VEDPM. The author, instead, introduces virtual laws and experiments as testing tools for proving the source code.

9. Swarm software agents--software agents have many definitions [1, 2, 3, 4, 5]. The author defines the term loosely for individual software processes that represent individual autonomous UAVs.
10. What You Simulate Is What You See (WYSIWYS)--a realistic scientific simulation governed by proven virtual laws via virtual experiments for scientific research. It is directly opposite to Pixar's animations.
11. Scientific simulations--they are special kind of simulations where sound virtual laws can be mimicked from existing real physical laws.
12. Process models--they are theoretical software development methodologies on how software is developed or tested effectively in fulfilling customers' and users' requirements. So far, the methodologies are arts and not sciences.
13. Unmanned vehicles--these are autonomous machines that can walk (Unmanned Land Vehicles, ULVs), fly (Unmanned Air Vehicles, UAVs), swim (Unmanned Water Vehicles, UWVs), and do space flight (Unmanned Space Vehicles, USVs).

1.4. Summary of Each Chapter

Chapter 2 introduces three main types of traditional software process models: Ad Hoc, Waterfall, and Iterative. Ad Hoc is the most intuitive: one just writes the code, runs the program, and hopes for the best. Waterfall is a top-down process model that plans every development step ahead of actual work. Iterative model will develop core software first, then adds more functions and improves the previous software in next iterative step. The rest of the models such as Prototyping are combinations of Waterfall and Iterative process models. In the chapter, the author argues that none of the process models addresses the unique problems posed by scientific simulations than VEDPM. The model closer to VEDPM is Ad Hoc in its unrestrictive and intuitive approach to problem solving. It, nevertheless, lacks virtual experiments for proving virtual laws, which in turn, prove the models and simulations. Some proponents of Exploratory Model insist that scientific simulations are part of the model. However, like Ad Hoc, Exploratory does not emphasize scientific method, virtual laws, and experiments. Moreover, it is more restrictive than Ad Hoc model. Chapter 2 continues to introduce the traditional V&V method for code testing. The author argues that the abnormalities of virtual laws are the issue and not the code per se. In addition, the traditional reviews, inspections, and walkthroughs are not concerned with the abnormalities. As such, VEDPM excludes traditional V&V since they are ineffective for proving realistic scientific simulations. The last part of Chapter 2 is about the background of swarm simulations and unmanned vehicles, and the problems facing them. In particular, there is no tool for verifying the simulations. VEDPM was inspired by the validating problem facing swarm simulations, which is the topic of Chapter 3.

Chapter 3 explains why two previous works are important to this dissertation. One paper has many weaknesses that the author wants to avoid in the dissertation. The other one has strengths that the author wants to include. The Solar Sail paper is the "prototype" of VEDPM. It is helpful to discuss in great length on how virtual laws are proven rigorously via virtual experiments in that paper. Most importantly, the author shows the creative process in selecting, modifying, and validating the well-accepted physical formulas. The author demonstrates that the product of the paper, the proven gravitational code, can be utilized to determine the significant digits of universal gravitational constant--a problem that is difficult to solve in real experiments. The ability of finding simple, sound equations is an important skill for VEDPM.

Chapter 4 stresses that VEDPM is the main objective of this dissertation. It has three main parts: virtual experiments, verify models, and WYSIWYS simulations. Virtual experiments are difficult idea to convey as some developers may think they are not necessary. The examples of climate and swarm simulations, nonetheless, suggest otherwise. The output of virtual experiments is proven laws, which in turn, are utilized to verify models. The proven models provide WYSIWYS simulations. The WYSIWYS feedbacks are used to improve the law or for adding next law. The experimenters can abort VEDPM if they fail to discover any law. One interesting requirement of VEDPM is that all proven laws must be used! The logic for that is subtle: nature does not hide all facts, but programmers do, intentionally or unintentionally. Section 4.2 explains the importance of critical thinking in designing virtual laws and experiments for Navy Swarms and Tetwalkers. Finally, the section lists the strengths and weaknesses of VEDPM.

Chapter 5 starts with swarm software architecture consisting of sensors, behaviors, maneuvers, virtual experiments, and simulation. The virtual experiment layer provides the proven forces to drive the behaviors. For a behavior model, sensory input served as a trigger to activate a behavior. Behaviors layer decides the maneuvers but the proven forces from virtual experiments layer animate the requests. The behaviors layer can be expanded into more specific hierarchical behaviors that are controlled by modified subsumption scheme. The scheme consists of a state and behaviors that has higher priority for higher behavior. The autonomous actions are resulted from the hierarchical behaviors and their priorities. The state, however, can interrupt them according to its goals. For switchable behaviors, shown in Navy Swarms, a UAV can have more than one control schemes that are triggered from a remote signal. The chapter continues to introduce minimalist as the preferred design philosophy in several issues. The behaviors coupled with principles of VEDPM are developed successfully for two real applications described in the next two chapters.

Chapter 6 demonstrates how three virtual forces, defined by the respective virtual laws, are created and proven by virtual experiments via VEDPM. The chapter explains an innovative and sensible virtual experiment is required for proving each force supported by reliable simulations and numerical data. The proven forces are fully defined by 9 Java code statements, which are easy to comprehend and refute by critics. Despite the short, simple code, it can drive complex Navy Swarms' scenarios depicted at the end of the chapter. The WYSIWYS simulations prove that the decentralized behavior model can achieve complicated tasks including switching behaviors for different goals triggered by a remote signal without losing dynamic local maneuvers of UAVs. The simulations conclude that numerous, inexpensive, and autonomous swarm munitions are a threat to Navy carriers.

Chapter 7 describes another successful application via VEDPM--Tetwalkers that are Unmanned Land Vehicles (ULVs). Unlike the UAV's engine, the Tetwalker's strut forces change only its shape. Tetwalker moves zigzagly by shifting its CG toward a direction. Since each strut force may not be in the direction of motion, the design of Tetwalkers' behaviors is more demanding than UAVs'. The chapter describes how virtual strut forces are proven rigorously by virtual experiments supported with reliable simulations and numerical data. Like Navy Swarms, the code for proven forces is short with only 6 Java statements. The chapter explains in detail how the proven forces help to discover an effective and proven behavior model that can achieve goals. The author wants to emphasize that without VEDPM, it is difficult to verify Tetwalkers' gaits. The WYSIWYS simulations demonstrate that Tetwalkers can explore Mars autonomously. Using VEDPM, this feasibility is known today rather than 30 years later!

Chapter 8 concludes this dissertation, which discusses whether the objectives listed in Chapter 1 have been met as well as future work.

CHAPTER 2. BACKGROUD AND RELATED WORK

The first part of the chapter introduces the traditional process models. They are not suited for scientific simulations as there is no methodology to validate the sciences (not code per se) being simulated. There are three main types of process models--Ad-hoc, Waterfall, and Iterative. The rest of the models combine or expand some of the elements of the three models. And none of them deals with rigorous requirements of scientific simulations.

The second part is about how to ensure software correctness and meet users' or customers' requirements. The traditional test methods are verification and validation, which consist mainly of reviews and code testing. These methods do not address misrepresentation of scientific facts in simulations but errors in code or requirements. Scientific simulations are about discovery that does not have predefined requirements.

The third part discusses the swarm literature and unmanned vehicles. The best-known swarm simulator is initiated by Santa Fe Institute (SFI), New Mexico [8]. Though the project was intended for "scientific" investigation in emergent behaviors of swarms, it suffers the same weakness as climate simulator mentioned above--that is, no methodology for validating simulations. Moreover, the software is overly complex and designed for all kinds of swarm applications. The author believes that each scientific simulation, like real-world counterpart, has unique virtual laws and experiments waiting to be discovered. Thus, it makes no sense to have a general scientific simulator. Most swarm simulators are closer to animations than scientific endeavors. To the best of author's knowledge, there is no work in applying the modified subsumption control scheme to unmanned vehicles. In addition, there is no work like VEDPM for validating all scientific simulations.

2.1. Traditional Process Models

There are three main types of process models: Ad-hoc, Waterfall, and Iterative. Each is explained in more detail in the following subsections:

2.1.1. Ad-hoc Model

This model is simply trial and error. It was used by many early software developments. It relies entirely on the skills and experience of the individual staff members performing the work. The feedbacks are mainly from the developers themselves. The main advantage of this model is to provide developers the greatest freedom in finding the solutions to the vague requirements, which are often changed with new results from running the programs. The main disadvantage is that the quality of the program is poor since it changes often with new findings. This model has one element "similar" to VEDPM than others is the need of experimenting which code works and which does not. Unlike VEDPM, it does not support scientific method, virtual laws and experiments.

2.1.2. Waterfall Model

The main principles of Waterfall [6], as shown in Figure 3, are the following:

- The project is divided into sequential phases with directions of feedbacks represented as arrows. Some overlaps between phases are allowed.
- It emphasizes on planning, schedules, and budgets.
- It uses formal reviews and extensive documentations to control the software quality.
- Approvals are needed for ending or initiating each phase.

The main strengths provided by Waterfall:

- The orderly phases and strict controls ensure quality, reliability, and maintainability of the software.

- The development progresses can be measured.
- With well planning, it conserves resources.

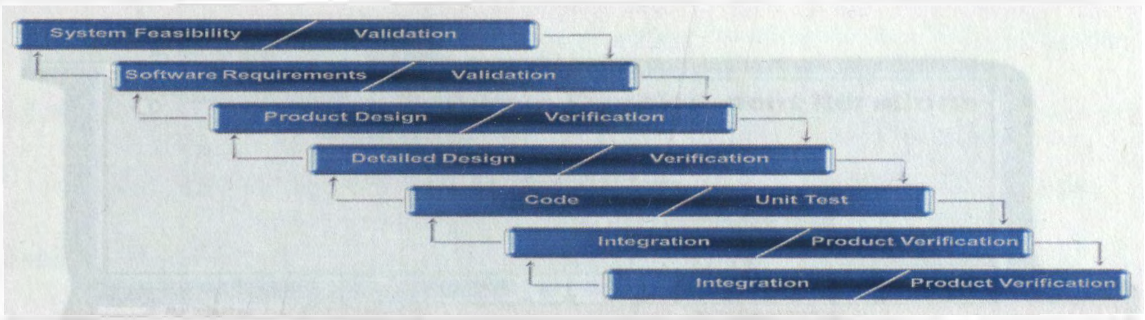


Figure 3. The Self-explanatory Steps of Waterfall Model

The main weaknesses of Waterfall are listed below:

- Real projects rarely follow the sequential flow required by the model.
- Changes that occur late in the life cycle are more expensive.
- The model requires users to identify their requirements early.

Waterfall is most appropriate for the following applications:

- Applications are large, expensive, and complex.
- Applications have clear goals and solutions.
- Applications' requirements are clear.

However, they are not appropriate for the following applications:

- Large applications with constant changing requirements.
- Real-time applications.
- Event-driven applications.

2.1.3. Iterative Model

Iterative Model [6], as shown in Figure 4, addresses the main weaknesses of Waterfall with the following principles:

- Feedbacks are allowed between requirements and iterations.
- It has smaller, incremental releases.
- It can accommodate changes occurred later in development.

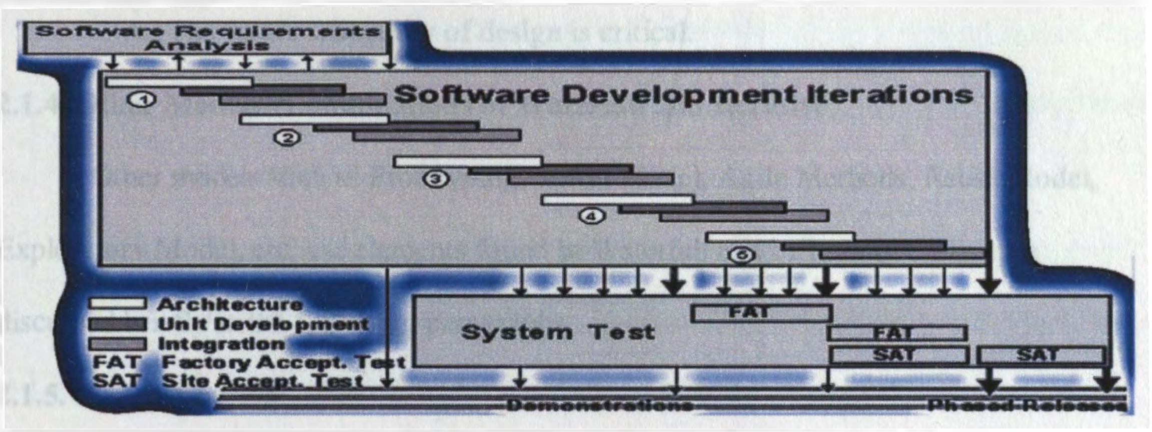


Figure 4. The Self-explanatory Steps of Iterative Model

The main strengths provided by Iterative Model:

- Each release provides faster results with less upfront costs.
- Valuable feedbacks can be obtained early.
- It does not require clear requirements from the start.

The main weaknesses of Iterative Model are the following:

- Users need to be actively involved throughout the project.
- Communication and coordination skills take center stage in project development.
- Informal requests for improvement lead to control and quality issues.

Iterative Model is most appropriate for the following applications:

- Project objectives are unclear.
- Functional requirements may change frequently and significantly.
- Projects that need to be implanted immediately.

However, it is not appropriate for the following applications:

- Project objectives are clear.
- Projects that are mainframe-based or transaction-oriented batch systems.
- The future scalability of design is critical.

2.1.4. Other Models--Combinations of Waterfall and Iterative

Other models such as Prototyping, Spiral Model, Agile Methods, Reuse Model, Exploratory Model, etc. use elements found in Waterfall and/or Iterative. They are discussed briefly in the following paragraphs.

2.1.5. Prototyping

It was developed on the assumption that it is often difficult to know all of the requirements at the beginning of a project. Typically, users know the objectives of the project but the details of data, system features, and capabilities. A throwaway prototype code is often developed for users' feedbacks, and entire new programs will be developed once the requirements are identified. Each release (Iterative) is a small-size Waterfall. Prototyping Models allow development without up-front requirements and developers can build simple version of the system and present it to customers, and the prototype code is often thrown away.

Prototyping has the following steps:

1. Requirements definition--it is similar to the conceptual phase of Waterfall, but not as comprehensive.
2. Design--once the initial requirements are collected, the prototype is rapidly developed.
3. Evaluation--the prototype is presented to the customer for comments and

improvements.

4. Refinement--new requirements collected from customers are studied and the prototype is refined further.

The main problems of Prototyping are the following:

- It can give customers a false impression that prototype is the finished product that has met all requirements.
- Due to rapid development, Prototyping can lead to poorly designed systems.

Other variation of Prototyping is Rapid Application Prototyping, which emphasizes strict time limits on each release and relies heavily on rapid application tools for quick development.

2.1.6. Spiral Model

The main feature of this model is risk assessment for each cycle of release (Iterative) that uses steps in Waterfall. It was designed to include the best features from Waterfall and Prototyping with risk assessment. Similar to Prototyping, an initial system is developed, and then repetitively modified based on input from customers. Unlike Prototyping, each development uses steps similar to Waterfall. Risk assessment is important step to evaluate whether the project should continue. The Spiral has the following steps:

1. Project goals--similar to Waterfall's conceptual phase, the goals or obstacles are identified, and alternatives are determined.
2. Risk assessment--the associated risks and alternatives are identified and evaluated.
3. Development--detailed requirements are determined and code pieces are

developed.

4. Management--customers can analyze the results and feedbacks are given to developers.

Unlike earlier models, the risk assessment component provides a valuable tool in assessing software development risks in software processes. The costs, however, can be more, and risks sometimes cannot be evaluated precisely.

2.1.7. Agile Methods

These methods do not focus on processes, documents, task distribution and development phases but on individuals, working software, customer collaboration and responsiveness to changes according to a plan. They use short iterations and working together with customers to achieve better communication, maneuverability, speed and cost savings. The main problem of the methods is that customers may not want a fully cooperating relationship with the team due to demanding work from them.

2.1.8. Reuse Model

This process reuses existing software components for new projects. It is particular suited for object oriented computing environments, which is popular in today's software development. The reused modules are maintained in software library that can be copied by any projects. It has the following steps:

- Definition of requirements--initial system requirements are collected.
- Definition of objects--the objects are identified.
- Collection of objects--scan the software libraries for potential reused objects.
- Customized objects--if reused objects are not suitable, create new ones.
- Create prototype--a prototype created and/or modified using the necessary

objects.

- Prototype evaluation--the prototype is evaluated to determine if it adequately meet the requirements.
- Requirements refinement--requirements are further refined as a more detailed version of the prototype is created.
- Objects refinement--objects are refined to reflect the changes in the requirements.

The main problems of Reuse model are the following:

- It is limited to object-oriented development environments.
- The reused components are developed in other systems under certain circumstances. As circumstances change beyond the limits of the model, the results from using it are no longer predictable.

2.1.9. The Exploratory Model

In some cases, it is difficult to identify a system's requirements as much of the research is based on guesswork and estimation. Like Ad-hoc model, there are no precise specifications. Exploratory is simple and has the following steps:

1. Initial development--a brief system's requirements are created for a rudimentary starting point.
2. System construction--a system is developed or modified based on available information.
3. System test--the system is tested to see the results and on to improve them.
4. System Implementation--after many iterations, the system is finished if the results are satisfactory.

The main problems of Exploratory Model are listed below:

- It requires high-level programming language such as LISP.
- It is hard to predict the costs and cost effectiveness.
- The design is crude.

The Exploratory model is discussed last as it seems similar to VEDPM. However, the main differences are the following:

- It does not support scientific method, virtual laws and experiments, and WYSIWYS simulations.
- It does not emphasize a short-code layer that mimics virtual laws like the 9 code statements in Navy Swarms or the 6 statements in Tetwalkers.
- It applies to all kinds of simulations but VEDPM is limited to scientific ones.
- In short, VEDPM is not Exploratory model.

2.1.10. Summary of Traditional Process Model

The author could not find a suitable traditional model for addressing the validity of scientific simulations despite claims from Exploratory and Ad Hoc. This is not a surprise since current processes do not separate the underlying virtual laws and subjective science models. The model designers fail to recognize a science knowledge and programming skill are inadequate to address the validity issues.

2.2. Traditional Verification and Validation (V&V)

This subsection explains general concepts of traditional V&V [38, 39, 40, 41]. Traditional V&V is the process ensuring that software being developed will meet right requirements (validation) and right product (verification). The differences between verification and validation are unimportant for practitioners except to the theorist. The practitioners use the term V&V to refer all the activities that ensure all the required functions are satisfied. The two main activities of V&V are reviews (including inspections and walkthroughs) and testing.

2.2.1. Reviews, Inspections, and Walkthroughs

Reviews are conducted during and at the end of each phase of the process life cycle to determine whether the requirements, design concepts, and specifications have been met. Reviews consist of the presentation of material to a review board or panel. They are most effective when conducted by personnel who have not been directly involved in the development of the software being reviewed. Reviews can be formal and informal. Informal reviews are conducted on an as-needed basis. The developer chooses a review panel and provides and/or presents the material to be reviewed. The material may be as informal as a computer listing or hand-written documentation.

Formal reviews are conducted at the end of each life cycle phase. The acquirer of the software appoints the formal review panel or board, who may make or affect a go or no-go decision to proceed to the next step of the life cycle. Formal reviews include Software Requirements Review, Software Preliminary Design Review, Software Critical Design Review, and Software Test Readiness Review.

An inspection or walkthrough is a detailed examination of a product on a step-by-step or line-of-code by line-of-code basis. The purpose of conducting inspections and walkthroughs is to find errors. The group that does an inspection or walkthrough is composed of peers from development, test, and quality assurance.

2.2.2. Testing

Testing is the operation of the software with real or simulated inputs to demonstrate that a product satisfies its requirements and, if it does not, to identify the specific differences between expected and actual results. There are varied levels of software tests, ranging from unit or element testing through integration testing and performance testing, up to software system and acceptance tests.

Testing can be formal or informal. Informal tests are done by the developer to measure the development progress. "Informal" in this case does not mean that the tests are done in a casual manner, just that the acquirer of the software is not formally involved, that witnessing of the testing is not required, and that the prime purpose of the tests is to find errors. Unit, component, and subsystem integration tests are usually informal tests.

Informal testing may be requirements-driven or design-driven. Requirements-driven or black box testing is done by selecting the input data and other parameters based on the software requirements and observing the outputs and reactions of the software. Black box testing can be done at any level of integration. In addition to testing for satisfaction of requirements, some of the objectives of requirements-driven testing are to ascertain:

- Computational correctness.
- Proper handling of boundary conditions, including extreme inputs and conditions that cause extreme outputs.
- State transitioning as expected.
- Proper behavior under stress or high load.
- Adequate error detection, handling, and recovery.

Design-driven or white box testing is the process where the tester examines the internal workings of code. Design-driven testing is done by selecting the input data and other parameters based on the internal logic paths that are to be checked. The goals of design-driven testing include ascertaining correctness of the following:

- All paths through the code--for most software products, this can be feasibly done only at the unit test level.
- Bit-by-bit functioning of interfaces.
- Size and timing of critical elements of code.

Formal testing demonstrates that the software is ready for its intended use. A formal test should include an acquirer-approved test plan and procedures, quality assurance witnesses, a record of all discrepancies, and a test report. Formal testing is always requirements-driven, and its purpose is to demonstrate that the software meets its requirements.

Each software development project should have at least one formal test, the acceptance test that concludes the development activities and demonstrates that the software is ready for operations. In addition to the final acceptance test, other formal testing may be done on a project. For example, if the software is to be developed and delivered in increments or builds, there may be incremental acceptance tests. As a practical matter, any contractually required test is usually considered a formal test; others are "informal."

After acceptance of a software product, all changes to the product should be accepted as a result of a formal test. Post acceptance testing should include regression testing. Regression testing involves rerunning previously used acceptance tests to ensure that the change did not disturb functions that have previously been accepted.

2.2.3. Summary of Traditional V&V

After careful studies in various traditional methods of software verification and validation, the author concluded that they are not well suited for VEDPM, which utilizes virtual laws and experiments as crucial tools for verifying WYSIWYS simulations. Traditional V&V concerns about code errors and incorrect requirements while VEDPM focuses on representing real sciences in virtual worlds. In short, traditional V&V was not designed for validating scientific simulations.

2.3. Backgrounds of Swarm Simulations and Unpiloted Vehicles

Swarm simulations mainly concern on imitating local, emerging, autonomous, reactive group behaviors of social beings like flock of birds, schools of fishes, swarms of insects, etc. Craig Reynolds' work [7] is frequently cited as model example for swarm simulations. It, however, has the following drawbacks:

- It is not a scientific simulation but movie animation.
- There is no behavior design in enabling swarm agents to achieve higher and practical goals.
- There is no mechanism for improving the current swarm behaviors.

Another popular swarm simulator is developed by Santa Fe Institute (SFI), New Mexico [8]. It was developed for "scientific research" for "general" swarms. Although the SFI's simulator is more rigorous than Craig Reynolds' simulation, it suffers similar weaknesses:

- The sciences in the simulator cannot be tested.
- The code is long and complex.
- It is a "general tool" for swarm simulations--i.e., all laws are predicted and changes to the basic architecture is impossible.
- It is an open-source project, which implies everyone can add some code to it.

Though there are many other swarm simulations, they are not as rigorous as SFI's simulator. Moreover, they have similar design flaws as SFI's simulator.

The overall design philosophy of emergent intelligence through local interactions with neighboring individual members is inspired by natural or synthetic swarms such as ants [9], [10], graphical turtles [11], boids [7], and fishes [12] .

Research in the area of autonomous behaviors, motor schema, and force fields as their control mechanism is in its infancy. Gillen and Jacques describe a simulator to evaluate control alternatives for intelligent munitions [13]. Passino et al. [14] explore a reactive biomimicry approach to developing a search map of a battlefield area with UAVs. Three examples of autonomous, multiple, mobile robotics aspires to similar control design goals: achieving a global behavior in a group of distributed robots using only local sensing, minimal communication, and behavior-based control mechanism are given by Fredslund and Mataric [15].

Werger [16] demonstrated a robot soccer-playing team with a minimalist, behavior-based control system. By combining a few basic behaviors, two different group formations of three robots emerged. Mataric [17] showed how a set of simple behaviors, based on local sensing, can be combined so that a global behavior emerges. For example, a global flocking behavior emerges as each robot performs its local. Kube and Zhang [18] demonstrated that only two basic local behaviors, avoidance and goal seeking, are enough for the physical robots to perform a collaborative box-pushing global behavior.

Altenburg, Schlecht and Nygard [19] developed a framework for a simulator that employs a swarm of UAVs with limited sensors and local behaviors to achieve the attack, and is arguably more robust than a deliberative approach.

2.3.1. Summary of Swarm Simulations and Unmanned Vehicles

Despite the works cited above, the author's work is different in the following manners:

- A general software architecture (in Chapter 5), tightly coupled with virtual experiments, were designed for unmanned vehicles such as UAVs, ULVs, UWVs and USVs. The same, common architecture has been applied successfully for UAVs (Chapter 6) and ULVs (Chapter 7), and partially for USV (Chapter 3).
- A switchable subsumption control scheme is created to enable each unmanned vehicle in achieving a higher goal, and a different one by switching to another set of the scheme.
- A validating tool, VEDPM, is utilized to verify the behaviors scientifically. It is proven via successful applications described in Chapter 6 & 7.

The foundation of this dissertation is inspired from author's previous work--NASA Solar Sail project, which is explained in next chapter.

CHAPTER 3. PREVIOUS WORK

3.1. The Overviews of Two Previous, Related Papers

Two related published papers from the author, "Synchronized Multi-Point Attack by Autonomous Reactive Vehicles with Simple Local Communication" [20] and "ANTS with Firefly Communication" [21], are discussed in this section. The author wants to avoid the weaknesses of first paper and employ the strengths of second paper. The first paper presents a model consisting of a swarm of unmanned, autonomous flying munitions to conduct a synchronized multi-point attack on a target. The UAVs lack global communication or extensive battlefield intelligence, instead, relying on passive short-range sensors and simple, inter-agent communication. The multi-point synchronized attack is successfully demonstrated in a simulated battlefield environment. The simulation results indicate that the reactive, synchronized, multi-point attack is effective, robust and scalable. It is especially well suited for numerous, small, inexpensive, and expendable UAVs.

The strengths of first paper are the following:

- The behaviors achieve a complex task.
- The UAVs are able to cooperate with each other for a common goal.
- Simple and minimalist sensors.

The weaknesses are the following:

- The physical laws or mechanics are not sound.
- The unsound laws cause some abnormalities.
- All computational units are based on pixels and thus useless for scientific experiments or research.
- The behaviors are not switchable for different goal(s).

The second papers is concerning Autonomous Nano-Technology Swarm (ANTS) from NASA that employs numerous, autonomous, l-kg solar sails for surveying and studying asteroids in the Asteroid Belt. There is no convincing work on a simulator that validates the solar sail's behaviors and weak propellant system in the extreme space environment. Thus, the author has developed and verified an Environment Agent (EA) that simulates gravity and light force based on well-understood Newtonian and sound light force equations. Sail Agents (SA) simulate swarm behaviors that are able to turn their reflective surface in four orthogonal directions and produce 3-D maneuvers. The simulator is able to model key behaviors of SAs. The author also provides a model that has the simplest swarm behaviors and unorthodox sensors for testing feasibility of ANTS using EA. The communication is done via on-off light patterns, which are similar to fireflies' light signals.

The strengths of second paper are the following:

- The simulations and numerical data are sound.
- The two basic forces are represented universally by mathematical equations.
- The gravity and the simulation errors, which are minimal, can be checked by an independent equation that is not part of the code.

The weaknesses are the following:

- There is no separate and independent virtual experiment software layer. That is, the virtual laws' code is intertwined with model's, which violates the principle of VEDPM.
- The behaviors are not switchable for different goal(s).
- It is the prototype of VEDPM. Thus, full benefits of VEDPM were not

applied.

In spite of drawbacks, the design of virtual laws and their proofs, as explained in details in the following subsections, inspired the author to develop a full-fledged VEDPM

3.2. The Proof Process of Virtual Experiment Layer in Solar Sails

Gravity is a major component of EA and was developed from the bottom up to provide validation in three ways: 1) the universal gravity equations implemented in code are verified to ensure they will provide the intended results, 2) the simulation results meet accuracy standards, and 3) the underlying force-vector model is sound.

The Newtonian gravitational law is represented from equations (1), (2), & (3) stated below. Equation (1) and (2) describe gravity everywhere, but equation (3) describes gravity as perfect circular orbits. Theoretically, with the same initial values, both groups of equations describe the same orbit. Equation (1) and (2) are coded, but not equation (3). The gravity in code is proven if it agrees with the theoretical one with minimum deviation as explained below.

Gravity from the sun acting upon the sail is computed using Newton's gravitational law:

$$Gravity = GMm/r^2$$

$$\boldsymbol{Gravity = GM/r^2 \ldots (1)}$$

$$Where\ G = constant\ of\ gravitation$$

$$= 6.6742 * 10^{-11}\, m^3\, kg^{-1} s^{-2}\ [6]$$

$$M = sun's\ mass = 1.9891 * 10^{30}\, kg\ [5]$$

$$m = sail's\ mass = 1\ kg$$

$$r = distance\ between\ sail\ and\ sun$$

Equation (1) is easy to use since the only variable is r. Once the gravity is known, the acceleration is given by Newton's second law:

$$F = ma$$

Where F = sun's attractive force on sail

m = sail's mass = 1 kg

a = sail's acceleration

Since the sail's mass = 1 kg, the equation can be simplified:

$$\boldsymbol{F = a \ldots (2)}$$

If the acceleration is known, the sail's velocity and position within a given second can be calculated. For example, the sail's acceleration at 1 AU, 149,597,870,000 meters [5], from the sun is computed as:

(2) = (1), by Galileo's Principle of Equivalence

$$\boldsymbol{a = GM/ r^2}$$

$$a = 6.6742 * 10^{-11} m^3 kg^{-1} s^{-2} * 1.9891 * 10^{30} kg \,/\, (149597870000\ m)^2$$

$$a = 5.9321 * 10^{-3}\ m/s^2$$

Since acceleration ***a*** is a vector, one cannot determine the velocity and displacement without its direction. The directions of sail's starting acceleration, velocity, and displacement are depicted in the left diagram of Figure 5. The acceleration $5.9321 * 10^{-3}$ m/s^2 means the change of velocity in 1 second is $5.9321 * 10^{-3}$ m/s (Newton's 2nd Law) in vector ***a***'s direction. Assuming velocity ***v***'s magnitude is 0.018 m/s and 90 degrees from acceleration ***a***, then the net velocity, ***u***, is computed by vector addition shown in the right diagram.

Vector ***u***'s magnitude is the square root of $(5.9321 * 10^{-3}\ m/s)^2 + (0.018\ m/s)^2$, which is $1.8952 * 10^{-2}$ m/s in velocity ***u***'s direction. The new displacement represents the new sail's location at the end of the second. The process is repeated again for the next second. Note that equation (3) is not involved.

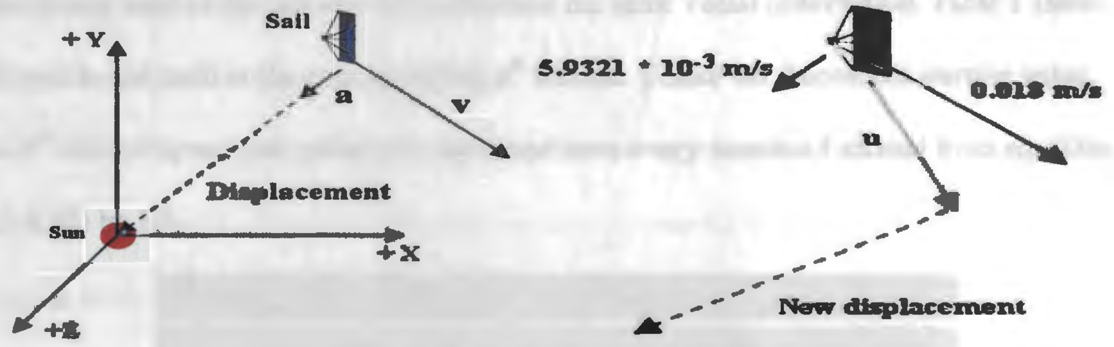


Figure 5. Sail's Vectors in Java3D

Newton's second law can also be applied in circular motion:

$$ma = mv^2/r$$

$$a = v^2/r \ldots (3)$$

Where v is velocity and other factors are defined above. Equation (3) is well tested through experiments. It indicates that for a perfect circular orbit, ***r*** is constant since it is the radius of the perfect circle. If r is constant, then acceleration ***a*** must be constant (since this is the same force at equal distance and thus the same acceleration, equation (1) & (2)), and by the validity of equation (3), the magnitude of velocity v is constant, as well too. The direction of velocity v is a tangent to the circle since any other directions will either increase or decrease the magnitude of the velocity. .

3.3. Empirical Observation and Experiments

The coded gravity was proven correct via experimental simulations and supported by the corresponding numerical data. At the start of the simulation, EA positions the sail at coordinate (149597870000, 0, 0). That is, 1 AU at the X-axis from the origin (0, 0, 0). As

EA required to push the sail around a *near* perfect circular orbit, equation (3) is used to calculate the theoretical starting velocity (i.e., at time = 0 second) at 1 AU for the sail, which is 2.9789 * 10^4 m/s. The author observed the sail moved in a circle. The numerical speeds and radii of the sail at 1 AU confirmed the same visual observation. Table 1 shows 10 speeds and radii at the corresponding nth second. Unlike the theoretical starting value, the nth-second speed and radius are calculated once every simulated second from equation (1) & (2) by EA.

SS = theoretical starting speed of sail = 2.9789 * 10^4 m/s		
SR = theoretical starting radius from sail to sun = 149,597,870,000 m		
N = nth orbiting second		
N (s)	**N speed - SS (m/s)**	**N radius - SR (m)**
3155296	**1.7434 * 10^{-3}**	**-8.7549 * 10^3**
6310592	**2.8209 * 10^{-3}**	**-1.4166 * 10^4**
9465888	**2.8209 * 10^{-3}**	**-1.4166 * 10^4**
12621184	**1.7434 * 10^{-3}**	**-8.7549 * 10^3**
15776480	**-7.3487 * 10^{-9}**	**7.1442 * 10^{-2}**
18931776	**-1.7434 * 10^{-3}**	**8.7550 * 10^3**
22087072	**-2.8209 * 10^{-3}**	**1.4166 * 10^4**
25242368	**-2.8209 * 10^{-3}**	**1.4166 * 10^4**
28397664	**-1.7434 * 10^{-3}**	**8.7550 * 10^3**
31552960	**-5.3878 * 10^{-9}**	**8.4320 * 10^{-2}**

Table 1. Numerical Evidence Verifying Gravity Equation

From Table 1, the nth-second speed and radius are nearly constant throughout the simulation. For example, the difference between the 31,552,960th-second (about 1 year) and theoretical starting speed is -5.3878 * 10^{-9} m/s. Similarly, the difference in radius is 8.4320 * 10^{-2} m. Over a few hours of running the simulation, the differences in speed and radius at the 6.988 * 10^{10}th second (2215 years) are -2.7537 * 10^{-3} m/s and 1.3828 * 10^4 m.

Other numerical data at other locations such as 3.3 AU confirm the same observations of Table 1. Thus, the sail travels in a near perfect circle as predicted. This establishes the validity of gravity in the code.

3.4. Gravity and Light Force in EA

3.4.1. The Model Design

After proving the gravity vector is sound, the next step is to add light-force vectors to the proven model. Blomquist's equations [22] are employed since the light-force equations are in vector forms:

$$drag = 9.12 * 10^{-6} * s\,(f_r \cos^3\theta + 1/2\,(1 - f_r) \cos\theta)$$

$$lift = 9.12 * 10^{-6} * s\,(f_r \cos^2\theta \sin\theta)$$

Where θ = the angle between incident light & sail surface's normal.

s = total sail's surface in square meters.

$9.12 * 10^{-6}$ N/m^2 = light force/m^2 due to normal incident light (i.e., $\theta = 0$) at 1 AU.

lift = a force component along orbit.

drag = a force component away from sun.

f_r = sail material's reflectivity, 1 means all reflected, and 0 means all absorbed.

Graphically, they are depicted in Figure 6:

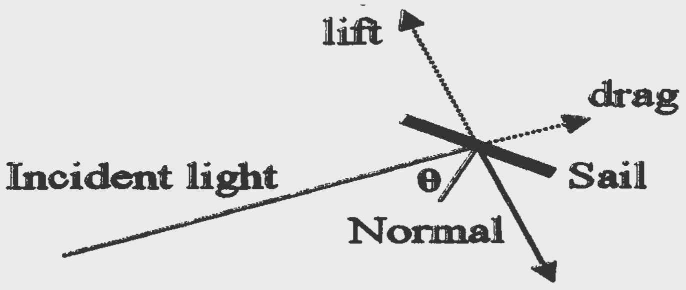


Figure 6. Light Force Components

Drag is the force vector that is in the direction of incident light, where lift is the force vector that is perpendicular to the incident light. The light-force equations are formulated at 1 AU only. The author modified them into universal equations for a sail that is r meters away from the sun with 100 m^2 sail surface as:

$$\boldsymbol{drag = 9.12 * 10^{-4} (AU/r)^2 (f_r \cos^3\theta + 1/2 (1 - f_r) \cos \theta) \ldots (4)}$$

$$\boldsymbol{lift = 9.12 * 10^{-4} (AU/r)^2 (f_r \cos^2 \theta \sin \theta) \ \ldots \ (5)}$$

A few observations about equation (4) & (5) are noted here. First, since the sail's mass is 1 kg, the light-force vectors in the equations are equal to their respective accelerations using equation (2). Second, θ describes how the sail turns its surface. Third, if $\theta = 90^\circ$, both drag and lift becomes 0 and thus has no light force--i.e., the edge of solar sail's surface is facing the sun directly. Fourth, if $\theta = 0^\circ$ & $f_r = 1$, then lift = 0, but drag = $9.12*10^{-4}$ m/s^2 [1, 4]--this implies the sail's entire reflective surface is facing the sun directly. Fifth, if $\theta = 35.3^\circ$ & $f_r = 1$, lift has maximum value of $3.51 * 10^{-4}$ m/s^2. Sixth, for light-force simulations, the author assumes $f_r = 1$ for all θ. Finally, all lifts have drag counterparts, which imply a sail orbits around an asteroid (one of the main goals of ANTS) could be difficult, if not impossible, as none of the drags or lifts are pointing toward the sun.

The light-force calculations are similar to gravity example above except for their directions. Additional observations are the following:

- Acceleration **a** is always orthogonal to lift but parallel and opposite to drag.
- Java 3D computes the net force by vector addition of **a**, lift, and drag.
- The net acceleration will change the sail's velocity v.
- The changed velocity v will change the displacement at the end of the second.
- The process is repeated in the next second.

SA can turn its surface counterclockwise (CC), clockwise (CW), up, and down as shown in Figure 7.

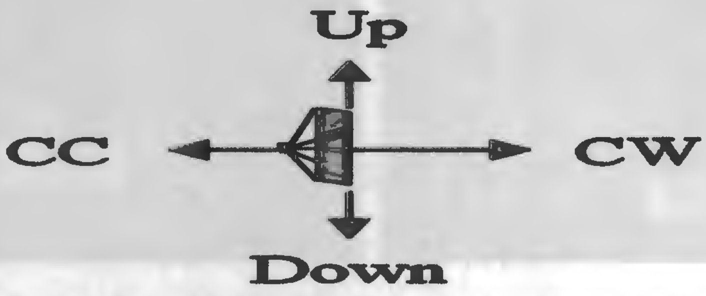


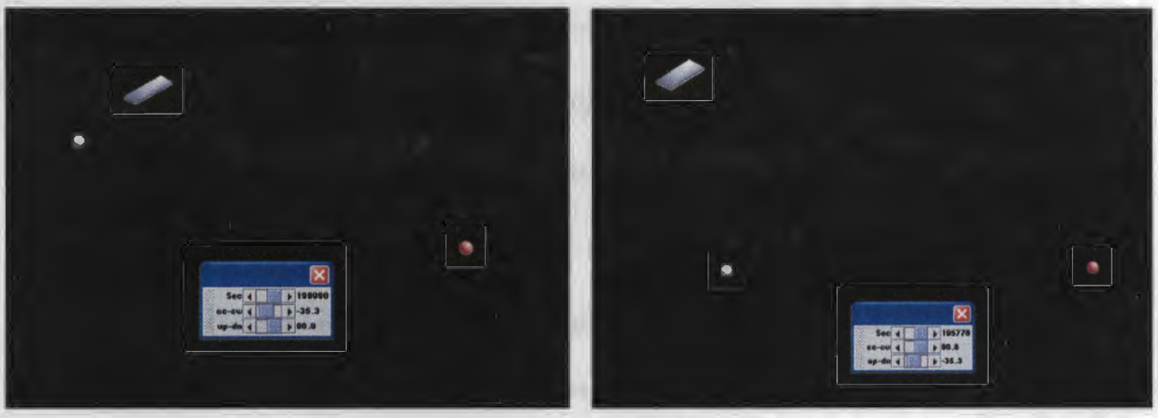
Figure 7. Four Ways in Turning Sail's Surface

CC is turning the cross counterclockwise horizontally with Up-Down arrows fixed. The opposite is CW. Up is turning the cross clockwise vertically with CC-CW arrows fixed. The opposite is Down. SA can only choose one of the four turnings in each second. Without them, the sail cannot move in a 3-D space. Equation (4) & (5) can be applied correctly in each direction as the sail's surface is a perfect square.

Unlike gravity experiments, there is no well-established theoretical light-force equations to prove the validity of equation (4) & (5) in code. However, if the equations are incorrect (not likely), it could potentially be replaced with other formulas as the underlying model in EA is still sound.

3.5. Scientific Simulations of Solar Sails

One way to verify light force without an independent equation is to compare its behaviors against proven ones from gravity. Thus, an asteroid ball (obeys only gravity) and a sail (obeys gravity and light force) were used for experimental simulations as shown in Figure 8. At the same starting position, both objects orbit together with the same velocity if default $\theta = 90^\circ$ (means no light force).



```
Sec    199990
ee-cw  -35.3
up-dn  90.0
```

```
Sec    195776
ee-cw  90.0
up-dn  -35.3
```

Figure 8. Light Force Reduces (Left) or Increase (Right) Sail's Velocity

The left picture shows the sail's speed is slower than the ball by setting θ=-35.3°, where light force acting against the sail's direction of motion. The opposite effect is setting θ=+35.3° that increases the sail's speed as depicted in the right picture. The author has visually confirmed that the sail was able to maneuver in 3-D space using four turnings manually, even at 3.3 AU. The corresponding numerical data agreed with visual simulations. It was, however, difficult to make the sail closer to the ball. Hence, the initial results indicate that orbiting around an asteroid (the main mission of ANTS) is difficult, if not impossible.

In summary, the light force is verified via experimental simulations and by additional reasoning listed below:

- As EA repeats the same proven process within each second, the light-force vector computations are valid by mathematical induction.
- The much weaker light-force vector is added to the proven gravity vector.
- The observed light-force effects in the simulations were consistent and predicted by equation (4) & (5).
- All vector computations are monitored in one Java method. Thus, potential

"misbehaviors" can be traced from there.

- Java 3D methods like vector1.angle (vector2) are used to check the orthogonality of gravity, lift, and drag in each calculation. And the fact that it is 90 degrees all the time proves the vector calculations are correct.

3.6. Testing Universal Gravitational Constant Using Proven Code

Some readers may be confused why the universal gravitational constant is being experimented hereby. The main reason is to convince the readers that the meticulously proven gravity code from Solar Sails is so valuable that it can help to determine the elusive value of universal gravitational constant (denoted G). Thus, the full-fledged VEDPM is an extremely powerful tool to solve some of the toughest problems in sciences. The experiment designed for testing the constant is explained in the following subsection.

Universal gravitational constant appears in many equations of physics such as Newtonian law of universal gravitation, Einstein's theory of general relativity, etc. Presently, most laboratory experiments [42, 43, 44, 45] agree with the first 2 significant digits of G: $6.6 * 10^{-11}\ m^3kg^{-1}s^{-2}$. The value, nevertheless, varies with each experiment due to extraordinary weak force of gravity. To improve the digits significantly, the experimenters, assuming they survive, have to live near a massive black hole.

Despite the difficulty, the author attempted to use the proven code to improve the significant digits of the constant. There are, however, several challenges in that approach:

- G value is determined solely from real-world experiments, and cannot be derived from equations such as those employed in Solar Sails.
- Mathematical equations are supersymmetry--i.e., the values on both sides of the equations are always balanced off, including the incorrect ones. This

implies the gravitational constant cannot be derived by manipulating arithmetic operators on related equations.

- At present, very little is known about the nature of gravity. Thus, it is "impossible" to prove the true value of G through theoretical analysis.

Despite the challenges, the author believes the gravity simulator can help to investigate the constant by utilizing Kepler's Laws:

- Kepler's Laws describe the orbital phenomena that are affected by gravity indirectly.
- Kepler's 2nd and 3rd Laws are already implanted indirectly in Solar Sail gravity simulator. For instance, the sail sweeps through equal circular area in one simulated second (2nd Law), and completes one revolution with period proportional to the radius of the circle (3rd Law).
- The author believes the specific strength of gravitational pull gives orbital properties of Kepler's Laws.

The steps of finding more significant digits of the constant are the following:

1. Incorporating equation (3) above into Kepler's 3rd Law:

$$T = c/v = 2\pi r/v$$

$$T^2 = 4\pi^2 r^2/v^2$$

$$T^2 = 4\pi^2 r^2/(GM/r)$$

$$T^2 = 4\pi^2 r^3/GM$$

$$\boldsymbol{T^2 = (4\pi^2/GM)r^3 \ldots (6)}$$

(T^2 is proportional to r^3, 3rd Law)

Where T = orbital period

c = *circumference of the orbit*

v = *tangential velocity*

2. Substituting an arbitrary G value in the equation to calculate the theoretical T value. For example, put $G = 6.67 * 10^{-11}$ in the equation (6) above:

   $$T^2 = (4\pi^2/(6.67 * 10^{-11} * 1.9891 * 10^{30}))(149{,}597{,}870{,}000)^3$$

   $$T = 31562889.7066 \text{ seconds}$$

3. Running the gravity simulator with initial velocity calculated from equation (3) using $G = 6.67 * 10^{-11}$ at position (149597870000, 0, 0). The time taken for the returning sail to reach its original position after one revolution is recorded. After 31562889 seconds, there is still 21044.0106 meters away from the original starting point, and with one additional second, the sail will go past the starting position. The fractional last second is not calculated by the simulator as it is smaller than one-second time step. In short, the remaining distance produced from the fractional second is a "break" from the rigid equations.
4. Comparing 21044.0106 meters from the simulator to the distance predicted exactly from theoretical last fractional second, which is 0.7066 s * 29780.2624 m/s = 21042.7334 meters.
5. Calculating the last-second "relative error ratio" of the two distances using the example: 21044.0106/21042.7334 = 1.0000606955. Then, compute the "relative error" by subtracting 1.0000606955 from 1.0, which is 0.0000606955; if the number is less than 1.0, then use 1.0 to minus that number. The main idea of this step is to break the theoretical equations;

otherwise, they will provide a "false" gravitational value as demanded from the equations. Thus, the "fuzzy" 21044.0106 meters are from the simulator. Whereas 21042.7334 meters are predicted exactly by Kepler's 3rd Law.

6. The G value that reflects the lowest relative error is "correct" based on the assumption the real G value will demonstrate the Kepler's property of orbital period.

However, it is still difficult to carry out the steps above if the "exact" digits of G are not known. Thus, the author hereby proposes a "correct" formula for $G = \pi/(\sqrt{3}*e) * 10^{-10} = 6.67259496511 60014298... * 10^{-11}$. This formula has several peculiar facts:

- The orbital period obtained from the simulator is 31556751 seconds or 365.240 days when G = 6.67259496511 6014298* 10-11. They are close to one solar year [46, 47] despite the fact that the orbit is a circle.
- All three variables, π, $\sqrt{3}$ and e are irrational numbers.
- The first 6 significant digits are 6.67259, which are standard G value since 1987 [48]. The odd for this to occur coincidentally is 1 in 1,000,000.

Some preliminary results using virtual experiment steps and G value described above are shown in Table 2. The data in Table 2 indeed favor $G=6.67259*10^{-11}$ as it has the lowest relative error among digits from 6.60 to 6.69 (except 6.67). This is just a preliminary experiment. To prove the next digit from 6.670 to 6.679, the steps above are repeated. The process continues until all the desired digits have lowest respective relative ratios.

The table does not include the digits 6.67 as they agree with the first three digits of 6.67259. To avoid other G candidates taking advantage of the last three assumed-correct digits of 6.67259, three 0's are placed at the end of each candidate.

G Value	Fractional-second Distance from Simulator	Fractional-second Distance from Kepler's 3rd Law	Relative Error Ratio	Relative Error
$6.60000*10^{-11}$	5846.16	5846.90	0.999875	$1.25212*10^{-4}$
$6.61000*10^{-11}$	21661.73	21661.60	1.000006	$5.83895*10^{-6}$
$6.62000*10^{-11}$	20439.59	20439.29	1.000014	$1.42691*10^{-5}$
$6.63000*10^{-11}$	25736.77	25736.61	1.000006	$6.44118*10^{-6}$
$6.64000*10^{-11}$	1788.08	1787.14	1.000526	$5.26412*10^{-4}$
$6.65000*10^{-11}$	1955.79	1955.42	1.000187	$1.87123*10^{-4}$
$6.66000*10^{-11}$	20282.06	20282.66	0.999971	$2.93975*10^{-5}$
$6.67259*10^{-11}$	***21523.59***	***21523.51***	***1.000004***	***$3.69460*10^{-6}$***
$6.68000*10^{-11}$	28040.30	28040.86	0.999980	$2.00520*10^{-5}$
$6.69000*10^{-11}$	5532.40	5532.07	1.000060	$5.95391*10^{-5}$

Table 2. Relative Errors Obtained by Breaking Equations

Though the virtual experiment might be able to show that $G = \pi/(\sqrt{3}*e) * 10^{-10}$, it is NOT the goal of this subsection. The objective is to demonstrate the power of proven gravity code developed from Solar Sails. Since proving the constant requires more work, including rewriting the source code, the author will leave the complete proof of the constant to future work.

Without the proven gravity simulator, it is impossible to devise the experimental steps and results shown above. Thus, this justifies the importance of full-fledged VEDPM for tough scientific problems, which is the focus of next chapter.

CHAPTER 4. VIRTUAL-EXPERIMENT DRIVEN PROCESS MODEL (VEDPM)

Although VEDPM concepts are discussed briefly above, this chapter explores them more. The author emphasizes the main objective of this dissertation is introducing VEDPM as a new software development process for scientific simulations. The swarm software architecture, though not a software process, is necessary for constructing successful behavior models via VEDPM. Section 4.1 introduces the basic elements of VEDPM. A simple diagram summarizes the process, and then followed by detailed explanation on what each part does. Section 4.2 explains the critical thinking needed for applying VEDPM on two real-world, complex applications discussed in Chapter 6 and 7. Section 4.3 lists the strengths and weaknesses of VEDPM. In particular, the author acknowledges VEDPM is only valuable for scientific simulations that have virtual laws designed by knowledgeable experimenters who can "think outside the box." It is not for software programmers per se--multifaceted skills are required.

4.1. Virtual-Experiment-Driven Process Model (VEDPM)

VEDPM shown in Figure 9 has three phases: virtual experiments, verify models and WYSIWYS Simulations; four branching decisions: proven laws, abort, add 1 law and feedbacks. The products are proven virtual laws (like forces) and models (like behavior models).

VEDPM has a clear and simple structure for conveying the main concepts. Unlike many process models, VEDPM encourages experimenters to quit the process as early as the first phase if they cannot design the virtual experiments for proving the virtual laws, which implies a realistic simulation model is not feasible. The details of each phase are explained in the following subsections.

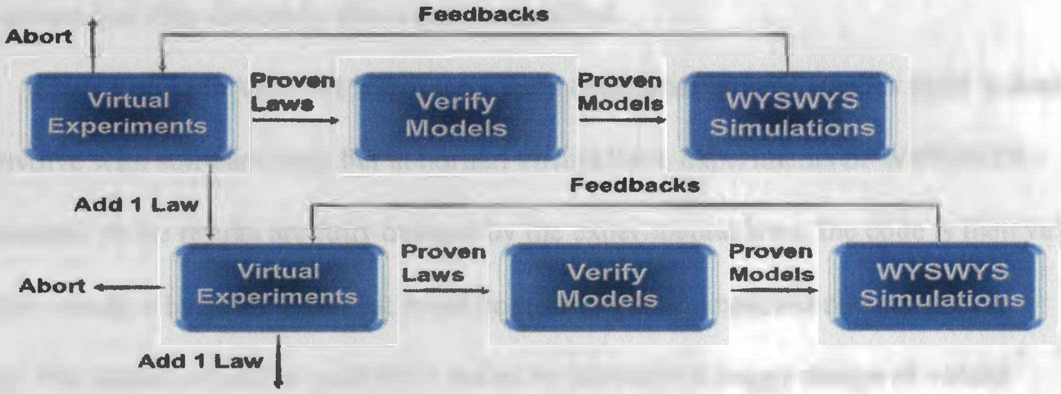


Figure 9. Steps of VEDPM

4.1.1. Phase I: Virtual Experiments

Virtual laws are crucial for the success of scientific simulations, but there is no rigorous process for proving them. This part is striving to imitate real-world experiments where physical laws are identified, isolated, and then experimented. Unlike virtual laws, physical laws are always present in and part of the nature. Since virtual laws are unnaturally created and integrated into simulations, sound design of virtual experiments are required for investigating them. If not, the simulations are unrealistic and practical applications cannot be determined. VEDPM is not for average experimenters, who may not have comprehensive skills for designing sensible virtual laws and experiments.

Despite some immaterial limitation, physical laws can be coded easily into virtual ones. Each coded law, obviously, needed to be proven. The experimenter must think critically for an ingenious virtual experiment that can validate the law clearly. The author, for instance, has constructed a simple turn-left experiment for probing the validity of forward thrust and left torque in Navy Swarms. The well-designed experiment, with help from a reliable equation, is able to prove the forces unambiguously via visual and numerical evidences. This process repeats until all laws are proven. Otherwise, the experimenter has to abort VEDPM. All proven laws (or code) are "natural" part of the simulations and thus cannot be discarded or modified.

Code testing in VEDPM is different from traditional process models since it does not involve with software bugs but abnormal virtual laws, experiments or WYSIWYS simulations. If the results are fully defined by the experimental laws, the code is then valid. In other words, a bug-free code that is not conforming to an experimental variable is buggy. The author offers the guidelines below to minimize a buggy design of virtual experiments:

- All virtual laws in virtual experiments must be identified, isolated, and experimented clearly. This is analogous to real experiments in which the experimental variables under investigation are always identified, isolated, and experimented.
- Simple, well-accepted laws should be preferred over complicated ones so that the experimental results are simply demonstrated without controversy.
- Each law must have a reason to exist. The incorrect law corrupts the whole experiment.
- Each law that has been proven must continue to exist in the simulation. This corresponds to real world that does not discriminate the presence of any rightful natural phenomenon.

As VEDPM is new and unorthodox, the author needs to convince readers that it is the only way to produce realistic scientific simulations. The author, thus, has developed two swarm applications--Navy Swarms and Tetwalkers--successfully via VEDPM. Chapter 6 and 7 explain the detail steps of employing VEDPM for developing the applications. The experimental results there support VEDPM.

4.1.2. Phase II: Verify Models

The experimenters can design any virtual model. However, only those obey the proven laws are chosen. If not, the resulted simulations are not viable. For example, the behavior models that are driven by the proven virtual forces were chosen for Navy Swarms and Tetwalkers. It is futile for model developers to deny as the applications, without the support of virtual laws, are not feasible and incurring significant costs if continue. To avoid manipulations from the models, the coded laws and models are separated. VEDPM does not restrict the length of feasible model code. In this phase, the experimenters should utilize the proven laws for deciding the feasible models.

4.1.3. Phase III: WYSIWYS Simulations

The valid models are now ready to demonstrate realistic simulations--WYSIWYS. The main characteristic of WYSIWYS is that both intended and unintended simulated results will be revealed, which are similar to real experiments. The negative outcomes are the strength of WYSIWYS and evidence of high fidelity simulator. If intended results occur, experimenters can stop VEDPM. Otherwise, the negative feedbacks are guides to further and better refinements. If the final refinement is still not satisfactory, the solution is then not feasible. At the very least, WYSIWYS can confirm the infeasible outcomes.

Some readers may argue that VEDPM has already included in the "broad definition" of traditional process model, and therefore not new. If it is true, there exist only two types of process model--Waterfall and Iterative--the rest, such as Agile Methods, are combinations of the two. VEDPM has new elements that are not emphasized by traditional processes for scientific simulations:

- Virtual experiments are constructed innovatively and instrumental for proving the relevant virtual laws.

- A simple, separated, innovative, unique, short code (less than a page, preferably) is invented to simulate proven virtual laws at every simulated time step.
- VEDPM implements scientific principles and method FAITHFULLY in virtual world.

4.2. A Simple Example Showing How VEDPM Works in Three Phrases

Despite many words in explaining the three phases, many readers are still confused why it is needed. As such, one naive example, which has no practical applications, is developed quickly via VEDPM. It serves as a shortcut for those readers who wish to skip Chapter 6 & 7 where practical applications are developed via VEDPM principles.

Assuming the UAV in Figure 10 has the following physical flight mechanics and initial conditions:

- The UAV has forward thrust represented by velocity **v** m/s due north initially.
- It has right torque represented by **a** m/s^2 due East initially, and **a** is perpendicular to **v**.
- It has no left torque.
- And that are all the forces it has. The magnitudes of **a** and **v** are large enough for useful rightward maneuvers in the virtual environment.

Furthermore, three readers of this dissertation are volunteering themselves to design a "scientific" simulation that illustrates the true trajectory of UAV within its abilities. Reader A or C is proficient in software design and coding--in the simulations, each person demonstrates that the UAV indeed flies trajectory A or C as indicated in Figure 10.

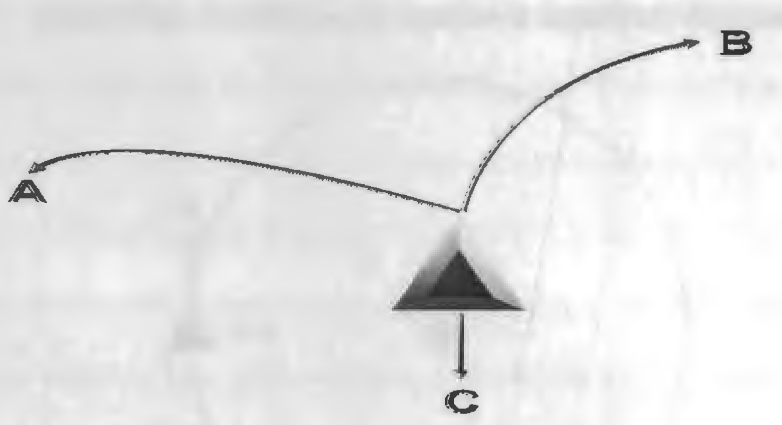


Figure 10. A Naive UAV with Forward Thrust and Right Torque

Reader B, nevertheless, used VEDPM to develop a simulation that shows trajectory B. Instead of arguing the proficiency of software skills and bug-free code, reader B employed scientific method for the simulation. To persuade reader A and C, reader B demonstrated how the virtual experiment was conducted.

The virtual experiment is set up in Figure 11. Reader B chose feasible and realistic $\mathbf{v_0}$ and $\mathbf{a_0}$ defined by a well-accepted vector circular equation $\mathbf{a} = \mathbf{v}^2/\mathbf{r}$. The virtual equation derived from the same physical one computes the same forces discretely (non-continuously), and thus some immaterial errors exist between the two equations. The radius is $\mathbf{r_0}$ meters ($\mathbf{r_0} = (\mathbf{v_0})^2/\mathbf{a_0}$) from the object. Figure 11 is an innovative virtual experiment design for verifying virtual forces of **a** and **v**:

- The expected experimental result is based on a well-tested vector equation of perfect circular motion with directions of $\mathbf{v_0}$ and $\mathbf{a_0}$ represented by the arrows.
- Radius $\mathbf{r_0}$, the predicted constant value, can be used to verify the experiment easily and unambiguously at any simulated moment. The virtual radius will not be equal to $\mathbf{r_0}$ exactly since computers are discrete machines.
- The virtual experiment does not involve any statistical analysis since each

value of the variables in the equation is completely deterministic.

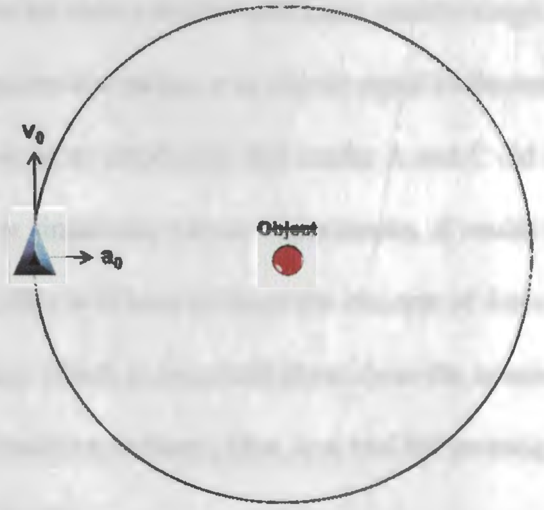


Figure 11. Virtual Experiment Conducted by Reader B

- If the experiment shows both virtual radius and r_0 are close and within acceptable error range, the virtual thrust and right torque are then verified.
- Thus, virtual experiment in VEDPM concerns about abnormal results or behaviors rather than software skills or bugs.

To prove VEDPM is workable, reader B wrote experimental code that set the UAV r_0 meters away from the object with constant forward velocity v_0 m/s and rightward acceleration a_0 m/s^2. Reader B knew that if the code could not enable the UAV in traversing a near perfect circular orbit clockwise in the simulation, as demanded by the equation, the virtual experiment would have failed. By trials and errors, the expected circular motion occurred with virtual radius within an acceptable error range, which validates the virtual forces. It also verifies that the chosen time step, one simulated second, is small enough to describe virtual forces accurately and realistically. In contrast, one can think about the "real world" is being simulated from an "imaginary continuous machine" with infinite speed!

The size of the error range depends on the chosen time step. A simulated millisecond is more precise than a second and has a smaller range. Phase I was completed for reader B when experimental radius was almost equal to theoretical one at the end of each second. It is important to emphasize that reader A and C did not have code for verifying basic forces or simulating virtual experiments. If reader A and C utilize the principles of VEDPM, they will have to learn the concept of force vectors for designing their virtual experiments, which in turn, will show them the reverse thrust and left torque are not feasible. The virtual experiment, thus, is a tool for investigating each force objectively and scientifically.

In phase II, reader B must write model code that proves trajectory B is the actual path and obeys the following VEDPM's principles:

- The model cannot have its own force code or change the proven ones.
- It only provides input values to the proven forces.
- It must apply all proven forces.

These principles ensure the model cannot ignore or alter the forces, which are tool for validating the model code. By experimenting different models, reader B finally wrote model B that simulated a path "close" to trajectory B--an error if it is exactly equal to trajectory B as the machine cannot compute the curve continuously. Reader B discovered that it was impossible to simulate trajectory A and C without violating the proven forces. Like phase I, reader B did not concern about software bugs if the simulated path is CLOSE to expected trajectory. Reader A and C, on the other hand, had no strategies for proving the underlying forces in their code.

Reader B relied on WYSIWYS simulations in phase III for visual and numerical confirmations. The graphics code that draws the results from the proven model must obey the following VEDPM's principles:

- The graphics code cannot change the model code.
- It draws objects defined by the proven model.
- It must represent the vectors and data defined by the proven model accurately and realistically.

Reader B utilized reliable Java3D, a third-party graphics API, for drawing WYSIWYS scenes accurately. Reader B ensured the model and graphical codes were separated--Java3D had no access to the model. Since the principles were strictly followed, reader B was confident that the simulations were WYSIWYS, and phase III was completed. Reader A and C did not bother the concepts of WYSIWYS.

By now, the readers should notice that VEDPM is anything but easy as evidenced by this "simple" example. Reader A and C finished their projects fast without VEDPM. Their work, however, produces no feasible maneuvers for the UAV. Reader B's work, in contrast, enables the real UAV to traverse a path close to trajectory C. Some readers may argue that it is obvious the reverse thrust and left torque are not feasible using common senses. The author, nevertheless, argues that VEDPM should be used to prove the "obvious" infeasible solutions scientifically instead of common-sense reasoning. In addition, the infeasible ones may not be obvious in complex applications.

The simple example demonstrates one crucial aspect of VEDPM--the well-thought-out design of the virtual experiment. Reader B's circle in the virtual experiment is an effective, powerful and inspired method for verifying the underlying forces of UAV. The readers should recognize by now that virtual experiments require creative thinking from experimenters, and it is not suited for someone who thinks linearly.

4.3. Critical Thinking in Developing Navy Swarms and Tetwalkers via VEDPM

The naive example has no practical applications and fails to demonstrate the true power of VEDPM. As a result, two complicated yet powerful applications, Navy Swarms and Tetwalkers, have been developed via VEDPM principles. The virtual law or experiment design requires critical thinking, and some general strategies are discussed in the following subsections.

4.3.1. Virtual Laws and Experiments in Navy Swarms

The objectives of Navy Swarms' simulations are determining whether numerous UAVs can achieve effective avoidance in close space and switch behaviors autonomously and adaptively when simple commands such as 0 or 1 are broadcast from a remote location.

The creative design of virtual laws for Navy Swarms was time consuming but indispensable. The virtual laws are forces that propel a UAV. The first obvious approach is applying difficult aerodynamic physical laws in virtual experiments, but there are several disadvantages:

- Aerodynamic laws are difficult to isolate, measure, and comprehend.
- They are complicated to design and code.
- The experimental results may be inconclusive due to multifaceted causes implied from the laws.

The author, as an alternative, designed three simple virtual vector forces--forward thrust, left torque, and right torque. They have the following advantages:

- The vectors can define the magnitudes and orientations of the three forces accurately.

- The resultant forces from vector additions are straightforward.
- The experimental results and abnormalities are unambiguous and easy to recognize.

Some critics may disagree with the *simplistic* laws. The author insists that the *simplicity* is essential for *simple* design of relevant virtual experiments. The forces alone are not enough for the experiments. A well-accepted circular motion equation provides two circles as expected goals: one circle is for forward thrust and left torque, and the other one is for forward thrust and right torque. The advantages of perfect circles are described in the simple UAV example discussed above. The behavior model of Navy Swarms does not interfere with the forces. It conveys the preferred actions but depends entirely on the proven forces for final simulated results. And WYSIWYS simulations provide realistic results of forces in the experiments.

4.3.2. Virtual Laws and Experiments in Tetwalkers

Compared to Navy Swarms, Tetwalkers have only two virtual forces--motor expansion and contraction--to be tested by virtual experiments. Again, simple vectors are employed for defining the virtual forces:

- A vector with its tail at the strut's center and head pointing to a node defines the magnitude and orientation of the expansive force.
- A vector with its tail at a node and head pointing to the strut's center defines the magnitude and orientation of the contractive force.
- The expansive and contractive forces have equal magnitudes but different orientations.

After the forces are defined, creative experiments are needed for proving them. The circular motion equation is not applicable to Tetwalkers due to their unique struts' structure. The symmetry of regular tetrahedral shape provides a unique method for testing strut forces in virtual experiments:

- 1-Tet has 4 nodes and 6 struts.
- All 6 struts have equal length.
- If equal force is applied on each strut, the regular shape is maintained, provided the initial shape is regular.

The objective of the virtual experiments is to confirm 1-Tet is maintaining the regular shape when equal expansive or contractive force is applied on each strut. Chapter 7 presents evidences that 1-Tet indeed can maintain its regular shape by expanding or shrinking equilaterally. Like Navy Swarms, the behavior model of Tetwalkers does not interfere with the forces. And WYSIWYS simulations provide realistic results of forces in the experiments.

4.4. Strengths and Weaknesses of VEDPM

The strengths of using VEDPM for developing applications are as the following:

- Virtual experiments are a tool for proving laws.
- The proven laws, in turn, are employed for designing or verifying model code.
- If the laws cannot be proven via experiments, the simulations are then not feasible.
- The unfeasible simulations are terminated in phase 1 if the forces cannot be proven.
- Last, at least two real and complex applications, Navy Swarms and

Tetwalkers, have been developed successfully via VEDPM.

There are, of course, weaknesses in applying VEDPM as listed in the following:

- The initial investment in virtual experiments could be high when reasonable virtual laws are not discovered after a long search.
- Resulted from extreme difficulty in simulating "virtual" experiments from mimicking the real ones, the experimenters must be creative and unorthodox--that is, to "think outside the box," a quality that not every researcher has despite the rhetoric!
- VEDPM is not suitable for researchers who expect predictable results or patterns.
- It is sometimes difficult to separate the code between the laws and models, and between models and graphics.

Despite the disadvantages, VEDPM is the first process model that, to the best of author's knowledge, requires the experimenters to apply scientific method rigorously.

CHAPTER 5. SWARM BEHAVIOR DESIGN

There are several reasons for choosing the swarm behavior applications as example software developments for demonstrating VEDPM principles and power:

- The author has experience in similar swarm applications for U.S. Navy/Air force and NASA. The work in [20], for example, is pioneering work and cited by researchers in Navy, Air force, European defense company, etc., as evidenced by Google Scholar search [49]. The application mentioned in the paper, nonetheless, is not developed through VEDPM. A more sophisticated version resulted in Navy Swarm application was developed via VEDPM principles.
- The swarm applications are challenging and hence ideal for demonstrating VEDPM.
- They are scientific simulations and thus virtual experiments are possible.
- The numerous, autonomous and reactive swarm agents with complex group behaviors and emergent intelligence are best demonstrated in simulations.
- Behavior models are able to achieve real goals adaptively and intelligently without sacrificing local, autonomous and reactive behaviors.
- Last, they cannot be developed convincingly using existing process models except VEDPM.

Swarm behavior design consists of two parts: swarm software architecture and modified subsumption control scheme. The first describes how VEDPM is integrated tightly with swarm behavior model for verification purposes. The latter depicts how autonomous behaviors are implemented and subsumed. The following subsections describe each part.

5.1. General Swarm Software Architecture

The swarm software architecture, proposed by author in conformance with VEDPM, is depicted in Figure 12. It has the following components:

- A behavior model that reads information from sensors at each simulated time step.
- The model suggests maneuvers to virtual experiment layer.
- Virtual experiment layer enforces maneuvers that obey sound physics and movement mechanics.
- The WYSIWYS simulations, supported by proven forces, provide realistic feedbacks for further actions.

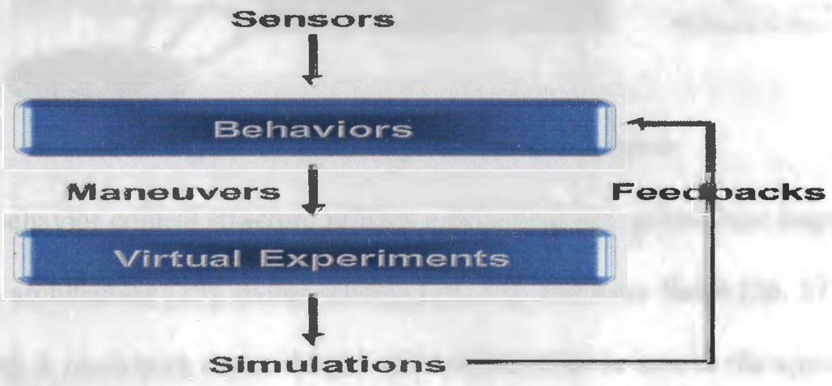


Figure 12. Swarm Software Architecture

The experimenters need more time to integrate sound physical laws into virtual experiment layer as virtual forces. One example from the author's previous work [21] in NASA's solar sails reveals that:

- The author needed to modify two universal light force equations from existing ones.
- The modification occupied most of project time as the researcher in light

forces [22] did not modify them into universal forces.

- If the author could not modify the light forces, the virtual experiments for solar sails would have failed.

VEDPM is a tool for validating all scientific virtual entities and not just virtual forces encountered in many swarm applications.

5.2. Modified Swarm Behavior Control Scheme

Swarm behavior models could be complex and uncontrollable. A control scheme, thus, is needed for managing them as outlined in Figure 13.

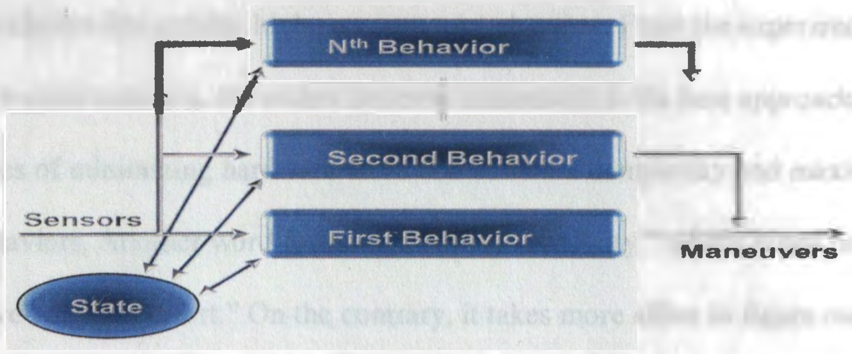


Figure 13. Subsumption Control Scheme

The behavior control structure utilizes a decentralized architecture inspired by the subsumption architecture [23], motor schema [24, 25], and force fields [26, 27, 28]. At each time step, it reads both sensors' input and internal state to invoke the appropriate behavior that suggests a maneuver. If State does not issue a control signal, the higher behavior layer (higher rectangle) will inhabit and subsume the lower layer automatically, as required by the subsumption scheme. The author has modified the original one by substituting a state that can achieve realistic goals. The original one does not concern with goal setting. The author also allows behaviors to be any order depending on the missions. Brooks [23], however, believed a natural, unalterable order exists, and it does not allow subsets of behaviors. But the author believes any combinations of behaviors are possible if they are proven by VEDPM. This is another reason why VEDPM is important, as it is able to settle the theoretical barriers imposed by a famous architecture. The tight coupling between the sensors, state and behaviors is responsible for vehicles reactive maneuvers. The structure can have N behavior layers as needed to accomplish a goal. A different goal has the same structure but different layers. The modified subsumption architecture, thus, can be applied to many other applications.

5.3. Minimalist Swarm Design Philosophy

Swarm software is usually designed for inexpensive, expandable and numerous unmanned vehicles that exhibit high autonomous behaviors. From the experience of developing swarm software, the author believes minimalist is the best approach to achieve the objectives of minimizing hardware costs and software complexity and maximizing reactive behaviors. Another word for minimalist is "simplicity," which is not necessary means "naïve" or "less effort." On the contrary, it takes more effort to figure out minimalist solutions. Simplicity has its root in "Occam's razor" since it means the simplest solution should be chosen. And it takes more time and effort to think creatively which solution is simple!

5.3.1. Minimalist for Hardware

The minimalist approach for hardware has the following goals:

- The hardware sensors and mechanics should be simple and thus are less expensive and easy to operate.
- The coupling software is less complex due to undemanding hardware.
- The unmanned vehicles are more fault tolerance due to straightforward software and hardware.

- Last, though not proven, the minimalist requirements enable the unmanned vehicles behave more autonomously.

5.3.2. Minimalist for Software

The minimalist approach for swarm software requires the simplest and minimal behavior models that achieve the same tasks should be utilized. In traditional reactive design, for instance, behaviors such as avoidance, aggregation and dispersion are automatically applied in similar applications. From author's experience, however, only avoidance is necessary for local and autonomous actions in many situations. Moreover, the legacy avoidance code can still be simplified further to minimize software complexity, side effects and bugs.

5.3.3. Minimalist for Heuristics

Swarm behaviors should not use complex heuristics such as fuzzy logics or neural networks unless it is necessary. The rationale for this is quite simple--do geese use fuzzy logics for figuring out flight formations? In addition, they add convolution to software code.

5.3.4. Minimalist for Virtual Laws

A better virtual experiment design should have less virtual laws. For each additional law, the experimenter has to spend more time and effort for proving its validity. Tetwalkers, for example, have only two virtual forces derived from the laws for their elaborated behaviors.

5.3.5. Minimalist for Goals

Navy Swarms are the first swarm behaviors that can incorporate high-level goals without losing local, autonomous behaviors. Nonetheless, more high-level goals are not necessary a better design. For each additional goal, more time and effort are needed in coding that may lead to code complexity.

Minimalist is difficult to practice. It requires one to crystallize and simplify one's designs diligently and intelligently. By comparison, the complicated solutions are easy to identify and quick to implement initially. In the end, however, the author believes the minimalist solutions are best suited for autonomous swarm applications.

5.4. Swarm Software for Real-World Applications

Simulations based on Swarm software architecture and modify subsumption scheme have no value if one cannot prove whether they are realistic, practical or feasible. However, Navy Swarms and Tetwalkers developed via VEDPM prove that the architecture and scheme can achieve real goals. The WYSIWYS simulations of Navy Swarms showed that UAVs achieved the following real-world goals:

- UAVs could avoid close objects and no collisions occurred during simulations with certain settings.
- The attackers could follow their leaders.
- The attackers could follow the targets.
- The behaviors of following leaders or targets were switched when a control signal was received.
- The switch did not affect the autonomous behaviors of attacking UAVs.

The WYSIWYS simulations of Tetwalkers demonstrated that the following real-world goals have been achieved:

- Tetwalkers were able to avoid close objects.
- The gaits were successful as envisioned by NASA.

- They were able to follow a trait.
- The behaviors were autonomous.
- The tumbling was successful.
- They could reach the target despite themselves as obstacles.
- Collisions did not occur during simulations with certain settings.

Judging from the visual images and numerical data from the WYSIWYS simulations, the author concludes that the said swarm software architecture and modified subsumption control scheme are effective. They are applied successfully to UAVs (Navy Swarms, via full VEDPM), ULVs (Tetwalkers, via full VEDPM) and USV (Solar Sail, via partial VEDPM). The details of two successful swarm applications employing swarm behavior design via VEDPM are explained in the next two chapters.

CHAPTER 6. VEDPM APPLICATION I--NAVY SWARMS

6.1. Introduction to Navy Swarms

The author describes a software development via VEDPM for realistic and credible swarm behaviors that can achieve global goals adaptively and intelligently without sacrificing local, autonomous, reactive behaviors. It has two main parts: the swarm behavior model and virtual experiments. In every simulated time step, the model suggests an action, which must be executed by proven virtual forces consisted of only 9 Java statements, toward a goal. The objective of virtual experiments is to ensure the virtual forces obey sound physical laws and movement mechanics of Unmanned Air Vehicles (UAVs). The software design is bottom-up and following a general architecture. The WYSIWYS simulations generated from proven model, demonstrate that numerous, inexpensive, expandable, maneuverable UAVs could sink an aircraft carrier. Navy Swarms have the following scenarios:

- Large bombers or modified cargo planes transport the UAVs. At certain altitude, they are dropped off from a safe distance (several hundred miles or more) from enemy munitions.
- Once entering the target area, the attacking UAVs (AUAVs) will follow (or search) enemy aircraft carriers or follow Communication UAVs (CUAVs) via a simple switch signal broadcast from a remote location, and another signal for switching back.
- Two swarm groups, CUAVs and AUAVs, are cooperating for common goals.
- The larger, faster, fewer, high flying and less maneuverable CUAVs pass control signals from remote command centers to AUAVs via sophisticate

communication sensors and equipment.

- The smaller, slower, numerous, low flying and more maneuverable AUAVs are high explosive munitions that follow either CUAVs or carriers.
- Though remote military officers cannot control low-level local behaviors, they can change high-level goals via remote signals.

WYSIWYS simulations are important and powerful tool for studying the effectiveness of large heterogeneous swarms in attacking enemy targets:

- The unmanned vehicles are expendable; the aircraft carriers, war ships and their crews are not.
- It is very demanding for enemy ships to shoot down all small and high maneuverable AUAVs. AUAVs are much cheaper that fighter jets; thus, some or all of them can be employed as decoys for depleting enemy munitions.
- The unmanned vehicles cruise at an altitude that may not be reachable by some enemy munitions.
- Some expensive munitions can reach UAVs--in that case, UAVs are decoys.
- The manned aircrafts are high value targets for AUAVs.

To prove the scenarios, it is imperative that the swarm simulations have the following requirements:

- The complex scenarios should be powered by a few simple forces.
- The forces must be proven via virtual experiments. The validated forces prove the behavior model; the verified model proves WYSIWYS simulations, and in that order.

- The source code of proven forces must not be changed or modified, as real-world forces are always present.

The current swarm simulators like the one developed by Santa Fe [8] do not address the requirements due to the following weaknesses:

- There is no attempt to prove virtual forces rigorously and scientifically.
- There is no attempt to prove that the mathematical equations used are coupled tightly to the forces.
- There are no separations of force, model and code.
- The source code of forces is changed or modified to suit the solution.

Some "general" simulators claim to represent all real-world forces in many situations. This assumption is flaw as each force is unique that requires unique coding. A scientific simulation, at the very least, must be proven via virtual experiments.

6.2. Navy Swarms' Design Strategies

The following subsections are Navy Swarms' design issues in virtual experiments, virtual forces, autonomous behaviors, and minimalist. The main concepts though have been discussed hereinbefore and hereinafter, it is helpful to reiterate them again, as they are neither obvious nor easy to understand.

6.2.1. Virtual Experiments

Virtual experiments are the most important issue in this dissertation. To the best of this author's knowledge, there is no rigorous work that emphasizes on experimenting virtual entities. Some forces, like Newtonian gravity, are so renowned that many software experts think they fully grasp it. The strategies for experiment design are the following:

- Only a few simple forces are considered; hence, air frictions and other minor forces are ignored at this early stage of project.

- The forces are just vector arrows; thus, vector computations are done via reliable vector methods from Java3D instead of complicated aerodynamic equations.
- The force code validated by the experiment is preserved.
- If virtual experiments for verifying forces cannot be found, the Navy Swarms' scenarios are not feasible.

6.2.2. Virtual Forces

The strategies for force design include the following:

- Only three simple forces are considered for initial investigation of Navy Swarms--forward thrust, left and right torque.
- A well-accepted circular equation can be applied to two virtual experiments--one is for experimenting forward thrust and left torque (anticlockwise circular motion), and the other for forward thrust and right torque (clockwise circular motion).
- If the virtual forces performed as intended, the virtual and theoretical radii should be close.

6.2.3. Autonomous Behaviors

Autonomous behaviors are desirable design due to reasons below:

- They enable UAVs to execute maneuvers with minimum human supervision.
- They allow numerous UAVs to cooperate smoothly as a group.
- They minimize operation costs as each UAV is not controlled by a remote pilot but via high-level goals from an officer--that is, local actions such as

avoidance are automatic.

- They are often governed by simple heuristics. The avoid behavior, for example, enables UAVs to apply close range rule in moving themselves to opposite direction spontaneously when their sensors detect close objects.

6.2.4. Minimalist

Minimalist design has the following objectives:

- For initial development, only two turnings for air maneuvers are necessary instead of standard six. This minimize hardware costs and software complexity
- To streamline swarm behaviors, only three virtual forces are introduced.
- Only three or less behaviors are required for Navy Swarms' scenarios.

6.3. The Virtual Experiment Setup for Navy Swarms

In the following subsections, the author describes how virtual experiments are set up for proving three virtual forces of UAVs--forward thrust, left and right torques (both orthogonal to thrust), as depicted in Figure 14. At each simulated second, the virtual forces perform the following:

- The forward thrust maintains a constant velocity.
- The left torque enables UAV to turn left at constant speed.
- The right torque enables UAV to turn right at constant speed.

The setup does not attempt the following issues:

- To employ complex aerodynamic equations.
- To include air frictions, winds, rains, snows and other weather conditions.
- To simulate movements of UAVs' mechanical parts.

- To claim that the forces work for other types of UAVs.
- To add new forces by discarding or modifying the old ones.

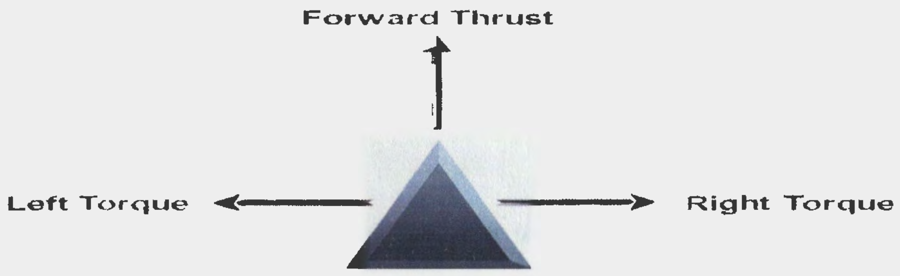


Figure 14. Three Virtual Forces of UAV

Figure 15 shows the forward thrust coupled with left torque enabling a UAV to fly counterclockwisely with the following actions:

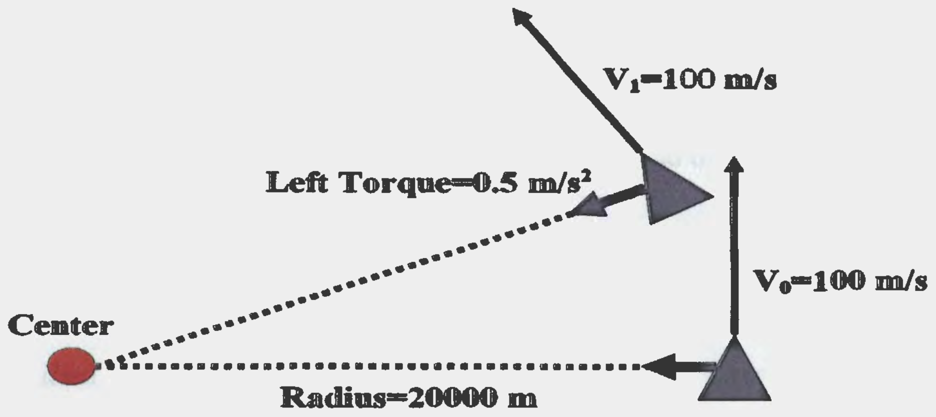


Figure 15. Counterclockwise Circling with Left Torque and Forward Thrust

- A UAV is circling around a center point with a radius of 20000 meters.
- The starting position is at coordinate (20000, 0, 0), which is 20000 meters away from the center.
- At time t=0 second, it travels at 100 m/s (vector $\mathbf{V_0}$).
- At time t=1 second, the velocity is still 100 m/s but the direction has changed as indicated by $\mathbf{V_1}$'s arrowhead.
- The position of UAV at t=1 is computed by vector addition of displacements

from $\mathbf{V}_1$ and left torque.

- The process repeats again for next second.

The vector values in Figure 15 are not random. The relationship between acceleration (i.e. force), velocity and radius is predicted by well-accepted circular motion formula below:

$$\boldsymbol{a} = \boldsymbol{v}^2/\boldsymbol{r}$$

Where a = acceleration in m/s²

v = velocity in m/s

r = radius in m

By plugging in the values from Figure 15, acceleration **a** can be calculated:

$$a = 100*100/20{,}000 = 0.5\ m/s^2$$

The vector **a** is left torque with magnitude 0.5 m/s^2 and orientation toward the center. Thus, if the UAV traverses a perfect circle, the virtual forces, forward thrust and left torque, are then proven. The experimental variable, in this case, is radius r=20000 m, which can be identified and measured easily by experimental code. There are, however, a few complications in this experiment:

- The circular equation describes continuous motion but the computer that performs the virtual experiment is a discrete machine--that is, it cannot compute a continuous curve like Calculus.
- The left torque must push the UAV continuously. However, in Figure 15, it is assumed to exert the force at exactly t=0 second. Thus, the UAV's position at exactly t=1 second IS NOT on the perfect circle, which violates the equation--that is, the virtual radius (or experimental variable), is not equal to 20000

meters, the chosen theoretical radius (or control variable). The forward thrust, nonetheless, does not violate the equation since it cannot alter the UAV's orientation.

- The error in two radii is accumulated after each simulated second.

Despite the difficulties, the two forces can still be proven if the accumulated errors relative to theoretical radius are insignificant. The following paragraphs proceed to conduct the virtual experiment and prove that the accumulated errors are immaterial as evidenced by both visual and numerical confirmations that the UAV indeed is negotiating a NEAR perfect circle in Figure 16 and Table 3, respectively.

The Figure 16 provides visual confirmation that a UAV starts flying from the right and orbits counterclockwise in a near circle as demanded by the circular equation. The circle is preserved during long simulations.

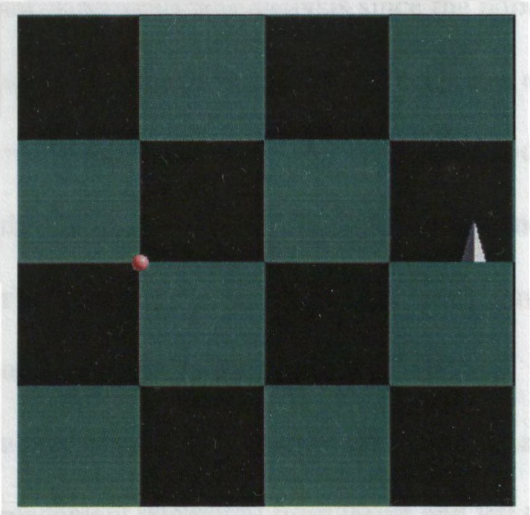


Figure 16. Visual Confirmation of Near Circle Motion

The numerical data in Table 3 has four columns:

1. The first column lists Nth simulated seconds. Each row is 3600 seconds apart.
2. The corresponding virtual radius at Nth second is in second column.

3. The third column shows the difference between the virtual and theoretical radii.
4. Error ratios are defined as the errors obtained in column 3 divided by 20000 m. The largest error is 49.84/20000= 0.002492, which is acceptable.

N^th Sec.	Virtual Radius ; VR (m)	Err=20000 –VR (m)	Err / 20000
3600	20037.44	37.44	0.001870
7200	20049.84	49.84	0.002490
10800	20028.56	28.56	0.001430
14400	19988.01	-11.99	-0.0005995
18000	19955.71	-44.29	-0.002215
21600	19953.69	-46.31	-0.002316
25200	19983.32	-16.68	-0.0008340
28800	20024.40	24.40	0.001220
32400	20049.02	49.02	0.002451
36000	20040.52	40.52	0.002026

Table 3. Numerical Data Confirming Acceptable Accumulated Errors in Radius

The author does not apply statistical analysis since the population variables from the equation are fully determined. A threshold like the acceptable error ratio should not be more than 0.005 is more meaningful and useful. Moreover, the code can notify the experimenters when the threshold is breached. Due to cyclic symmetry, the errors stay within 51 meters in either sign during the experiments. Finally, based on the visual and numeric evidences, the author concluded that:

1. The accumulated errors of simulation are small and acceptable.
2. The net displacement computed from thrust and left torque is proven; otherwise, the UAV will not fly in a near perfect circle.
3. All other factors, including graphics, vector class methods, experiment setting, etc. are all verified. If not, the circular path will not appear.

The forward thrust coupled with right torque have also been proven by similar virtual experiments with the exceptions the other UAV starts at coordinate (-20000, 0, 0) and circles clockwise. The numerical data, due to symmetry, are the same as Table 3. The same conclusions are thus applied as well.

The source code that enforces all three proven forces (forward, left and right) in virtual experiments has only 9 Java statements, as listed below:

```
1. if(modelVector.length()!=0.0){
       if (modelVector.angle(leftTorque)<=Math.PI/2.0)
         modelVector.set(leftTorque);
2.     else modelVector.set(rightTorque);
3.     modelVector.normalize();
4.     modelVector.scale(0.5);}
5. modelVector.add(velocityOfUAV[i]);
6. modelVector.normalize();
7. modelVector.scale(velocityOfUAV[i].length());
8. velocityOfUAV[i].set(modelVector);
9. positionOfUAV[i].add(velocityOfUAV[i]);
```

If statement 1 has zero value for model vector, the control is passed to statement 5. Otherwise, statement 1 checks whether the model vector is close to left torque. If true, it is set to left torque; or else, statement 2 set it to right torque. Statement 3 & 4 ensure both turnings have maximum torque (0.5 m/s^2). Statement 5 performs vector addition of current UAV's and model vectors velocities. Statement 6 & 7 make sure the resultant velocity has constant value (100 m/s). Statement 8 & 9 determine the net displacement. And the same process repeats again for next second. One can view the 9-statement virtual forces as a basis to accept or reject the feasibility of Navy Swarms' scenarios. If the readers reject the scenarios, they have to find errors in the 9-statement code.

6.4. Switchable Swarm Behavior Model Development via Proven Virtual Forces

The following subsections introduce the specific swarm software architecture and subsumption control scheme for behavior model that controls CUAVs or AUAVs. The three proven forces have validated the model, which in turn, provides WYSIWYS simulations. The fact that Navy Swarms' scenarios are generated by the model proves the intended goals of the scenarios have been achieved, in turn, prove the underlying virtual forces, software architecture and control scheme are properly design and valid.

The software architecture for each CUAV or AUAV is illustrated in Figure 17:

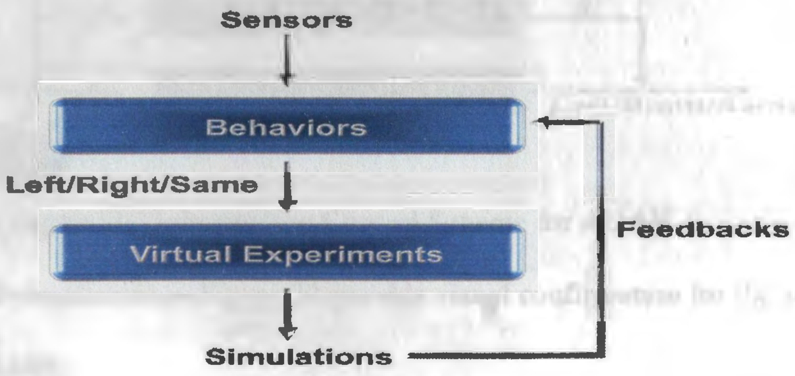


Figure 17. Swarm Architecture

The behaviors layer for CUAV can be expanded further into two behaviors that are controlled by subsumption scheme as shown in Figure 18. The avoid behavior has the highest priority due to self-preservation and thus placed higher in the scheme. If there are no close objects, the control, nevertheless, is passed automatically to search target behavior.

The subsumption control scheme in Figure 19 depicts AUAV, which has the following behaviors:

- To avoid close objects.
- To follow targets or CUAVs as threats are over.
- The state will take control and switch behaviors when high-level goal signal is broadcast.

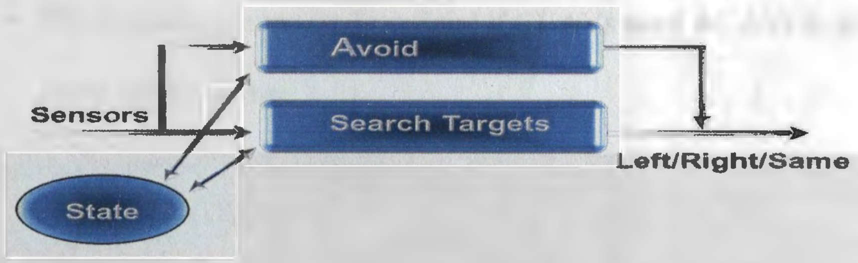


Figure 18. Subsumption Control Scheme for CUAV

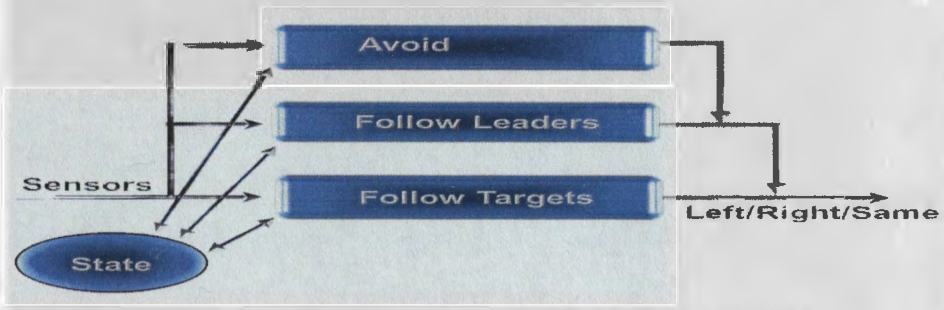


Figure 19. Subsumption Control Scheme for AUAV

WYSIWYS simulations in Figure 20 provide visual confirmation for the activities of CUAVs and AUAVs:

- The balls represent enemy aircraft carriers sail slowly at sea, which is the squares of checkerboard. The simulated aircraft carriers are not WYSIWYS since their forces, which are immaterial and irrelevant, are not proven.
- The high-flying CUAVs are watching over the carriers within an area of interest. If CUAVs fly outside the area, they will reverse course.
- By default, each AUAV follows the closest CUAV. If CUAVs are distant enough, three distinct AUAVs' groups formed beneath them. Both AUAVs and CUAVs will avoid close objects impulsively. Once the threats are over,

AUAVs follow the closest leader again. Two groups of AUAVs will swap their members dynamically when they are close.

- The inter-group or intra-group interactions of CUAVs and AUAVs are local, dynamics and unpredictable.
- The dynamic groups are beneficial if CUAVs need AUAVs to cooperate some tasks together.

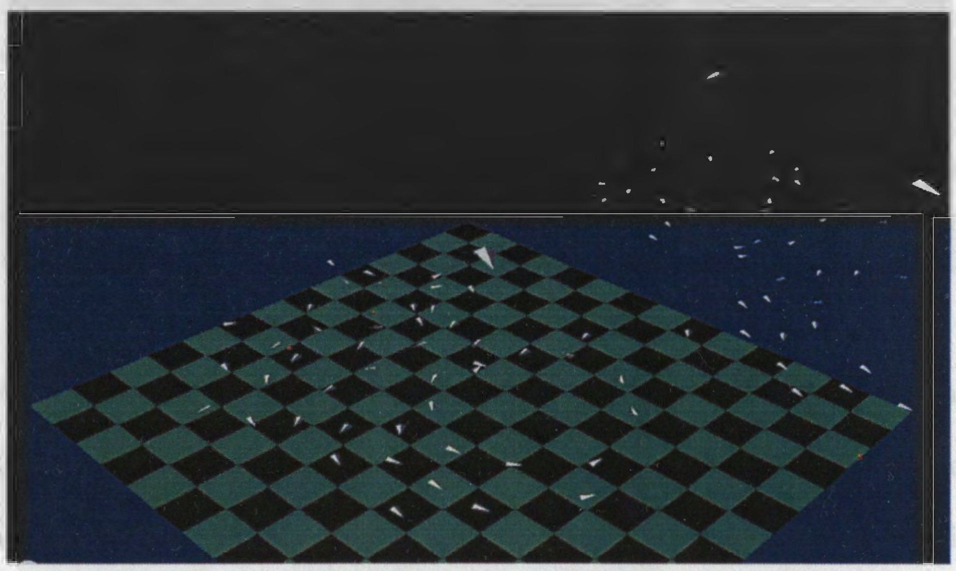


Figure 20. Visual Confirmation of Navy Swarms' Scenarios

In Figure 21, AUAVs have switched their behaviors from follow-leader to follow-target behavior after the control signal was broadcast:

- AUAVs switched to new goal adaptively and autonomously--that is, the switching was done without human supervising.
- AUAVs swapped members among groups when targets are close.
- CUAVs continued their behavior as before.

The WYSIWYS simulations demonstrate that the Navy Swarms' scenarios are successful. Moreover, visual images of battleground can be passed from ACAVs to

CUAVs, and then to remote officers via satellites. The officers have clear view of the battlefield without physical presence.

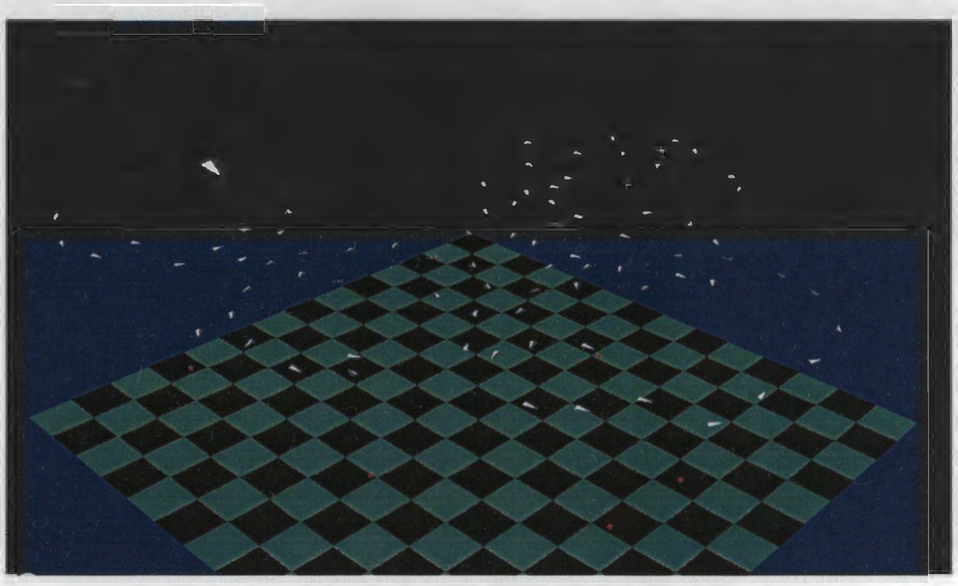


Figure 21. Adaptive Behavior via a Remote Signal

Navy Swarms are lethal to aircraft carriers if an officer decides to send the attack signal (in future work) to AUAVs whose targets are closely monitored and surrounded. The WYSIWYS simulations also show that AUAVs can adapt to new goals smoothly via simple control signals sent by remote offices without messing the local autonomous behaviors. This paradigm can be expanded to accomplish more challenging military tasks without human soldiers.

6.5. Vector Algorithms in Behaviors

Vector algebra is executed by reliable Vector class methods of Java3D that are previously tested. Vector computations are utilized by behaviors to ascertain positions, displacements, ranges, etc. Though there are many class methods for vector computations, vector algorithms combine them intelligently for specific behaviors. The following paragraphs summarize how vector algorithms are applied for various behaviors.

6.5.1. Vector Algorithm for AUAV-follow-CUAV Behavior

```
for each time step {
    if control signal says to follow target, jump out of this loop
    if CUAV is in sensor range and no closest object in close sphere {
        determine the displacement pointed to CUAV
        find the turning that has the smallest angle relative to the displacement above
        suggest the turning to virtual experiment layer
    }
    else if close objects are in close sphere {
        calculate the displacements of each object
        find the shortest displacement
        find the turning that has the greatest angle relative to the displacement above
        suggest the turning to virtual experiment layer
    }
    else if no CUAV is in sensor range and no objects in close sphere {
        maintain current velocity and direction
        suggest current turning to virtual experiment layer
    }
}
```

6.5.2. Vector Algorithm for AUAV-follow-target Behavior

```
for each time step {
    if control signal says to follow CUAV, jump out of this loop
    if target is in sensor range and no closest object in close sphere {
        determine the displacement pointed to target
        find the turning that has the smallest angle relative to the displacement above
```

```
            suggest the turning to virtual experiment layer
        }
        else if close objects are in close sphere {
            calculate the displacements of each object
            find the shortest displacement
            find the turning that has the greatest angle relative to the displacement above
            suggest the turning to virtual experiment layer
        }
        else if no target is in sensor range and no objects in close sphere {
            maintain current velocity and direction
           suggest current turning to virtual experiment layer
        }
}
```

6.6. Navy Swarms' Strengths and Weaknesses

The strengths of Navy Swarms are the following:

- The Navy Swarms' scenarios are proven meticulously via VEDPM.
- The 9-statement virtual forces are simple to implement and comprehend.
- The cheap, expandable and autonomous UAV swarms could sink a mighty aircraft carrier.

The weaknesses of Navy Swarms are examined here:

- CUAVs magically reverse their orientations at the borders.
- The numerical data, if needed, have to be coded for displaying through a console or file.
- There are 100 UAVs in the simulations; thus, the numerical data are chaotic.

6.7. Conclusion for Navy Swarms

The realistic and proven WYSIWYS simulations, validated by the 9 code statements, demonstrate that some results cannot be ignored by U.S. Navy in near future:

- The aircraft carriers are vulnerable to swarms of unpiloted vehicles.
- CUAVs and ACAVs are difficult to be targeted or destroyed by naval munitions due to their random and reactive motions.
- The cheap UAVs can be used as decoys to deplete munitions.
- UAVs are fearless.

High-level goals are able to incorporate into local behaviors seamlessly via remote signals. The paradigm is more powerful and robust than one adopted by Predators, which are controlled by remote pilots. The VEDPM methodology is not a one-time wonder. The author has also developed other complex swarm behaviors successfully via VEDPM-- NASA Tetwalkers. They are described in next chapter.

CHAPTER 7. VEDPM APPLICATION II--NASA TETWALKERS

7.1. Introduction of Tetwalkers

Tetwalker (Tetrahedral Walker) is a novel and futuristic NASA project for Mars explorations with unique structures:

- Unlike Mars Rover, Tetwalker does not have wheels. NASA scientists believe it can traverse difficult terrains like Martian craters.
- It consists of nodes connected by struts, and each strut has a motor at its center.
- The struts must expand/contract cooperatively for locomotion.
- It moves by shifting its CG.

Tetwalker's gaits via motorized struts are complicated despite its unassuming appearance. For comparison, the human legs have similar structures such as knees (nodes) and elongated bones (struts), and each leg essentially has 3 nodes and 3 struts (ignoring toes). Human children need several years for learning how to walk properly by balancing their "nodes" and "struts" and predicting the next move "intuitively." As a result, VEDPM is a valuable tool for assessing or studying the feasibility of gait models in virtual environments. Moreover, VEDPM has been employed successfully in verifying Navy Swarms' autonomous behaviors, which are similar to Tetwalkers' gaits. The Tetwalker simulations have the following scenarios:

- A set of ordered waypoints, which form a path, are laid over the crater and served as goals for Tetwalkers.
- Two Tetwalkers are competing for each waypoint.
- Tetwalker moves by shifting its CG along a direction.

- Tetwalker tumbles if its CG is outside the base triangle. After tumbling, it will first restore to regular tetrahedral shape before selecting a direction.
- Shifting CG in any orientation requires "proper" coordination of all expanding/contracting struts connected by the nodes.
- If they are dangerously close, they will avoid each other by moving temporary in opposite directions.
- The avoid or path-following behavior is triggered by proximity sensors, whereas shape restoring is activated by gyroscopes that measure angular velocities.

7.2. Background of Tetwalkers

NASA has two types of Tetwalker: 1-Tet (4 nodes and 6 struts) and 12-Tet (9 nodes and 26 struts). 1-Tet's gaits were orchestrated manually by several NASA scientists via wireless controllers [29]. From the video, the gaits are obviously slow and preplanned with great effort. 12-Tet has impractical animated gaits that require corresponding human intelligence [30]. Nonetheless, the real and successful 1-Tet's gaits did provide important clues for N-Tet's gait model design:

- The shifting of CG enables locomotion and tumbling successfully.
- The large triangular base provides stability.
- 1-Tet is able to restore regular tetrahedral shape.
- Despite wheels, 1-Tet flip-flops towards a goal in zigzag pattern.

A feasible gait model must be motorized by proven expansive or contractive strut force. Applying VEDPM principles, the author first employed mathematical constructs, 3D vectors, as virtual strut forces with magnitudes and orientations; two sound virtual experiments were designed for authenticating the forces, which, in turn, were powerful tools in searching a feasible gait model.

After several failures, the successful model, validated by the forces, has two parts: an imaginary Tetwalker, generated by mathematical equations with perfect and valid gaits, and a virtual Tetwalker with a goal of mimicking the shape of imaginary one--the resulted gaits, though not perfect (as slacks are allowed), are sufficient for mission objectives described in Tetwalker scenarios.

As every virtual action derived from the gait model is reinforced by the proven forces, the resulted simulations are WYSIWYS. Moreover, the WYSIWYS gaits of virtual 1-Tet match closely to the real 1-Tet's gaits demonstrated by NASA. And the model can be extended for proving N-Tet's gaits. The designs, experiments, gait model and proofs are explained in detail in the relevant subsections below.

7.3. Tetwalkers' Design Strategies

The strategies address various design issues in developing WYSIWYS simulations of Tetwalkers such as virtual experiments, forces/laws, autonomous behaviors, etc. Autonomous behaviors have been studied by some NASA scientists [31, 32, 33, 34, 35, 36] and the author [20, 21]. The following paragraphs briefly discuss the design issues, and readers can regard some of them are motivated by VEDPM.

7.3.1. Virtual Experiment

The virtual experiments in Tetwalkers are simpler than those in Navy Swarms as struts are moved only by two basic forces. To simplify the experiments, the following steps were implemented:

- The experiments assume strut forces are conserved notwithstanding conditions of the ground or Tetwalkers' mechanical parts--the heat energy is not generated due to frictions on ground or between parts.

- Java3D's vector class methods are used exclusively for calculating or manipulating resultant forces since they have been proven reliable and accurate in virtual experiments from past projects [21].
- The strut's CG is at its very center and mass is uniformly distributed.

7.3.2. Virtual Forces and Physical Laws

Virtual forces that represent sound physical laws are proven via virtual experiments. The virtual strut expansive and contractive forces are represented by Java3D's vectors, which implement Newtonian laws. The Double data type stores all, either intermediate or final, results from vector computations. Although Double had some abnormalities in the past, it is now reliable and accurate to about 15 significant digits. Furthermore, the author has applied Double type to other experiments that yielded predicted outcomes. The experiments and model here, nevertheless, require fewer significant digits.

The two forces are assumed to have equal, constant magnitude but opposite orientations. Without constant magnitude, the forces are difficult to manipulate and design. The author does not believe that the complex variable forces can produce a better solution. Moreover, the complex force design may not be realized into products or controlled easily by human operators.

7.3.3. Autonomous Behaviors

The autonomous behavior design is appropriate for Tetwalkers' scenarios due to following reasons:

- Applying minimalist approach, the resulted behaviors are simple, autonomous and spontaneous, which are best suited for Mars exploration. Coupled with switchable behaviors invented by the author, Tetwalkers can conduct missions autonomously via simple control signals.

- The work like Navy Swarms strongly suggests that the autonomous behaviors are fit for Tetwalkers' missions.
- Local behaviors are loosely coupled to higher-goal commands. The gait behaviors, for instance, will ignore the Tetwalker's higher objectives if immediate obstacles or threats are close.

7.3.4. Center of Gravity (CG)

Tetwalker utilizes CG formula to realize that:

- It is in tumbling state if CG projected on the ground is outside the base triangle.
- It is in unstable state if CG is too high.
- It is immovable if CG is unchanged for several time steps.

There are other decisions that are based on the state of CG; consequently, CG is like "machine consciousness" of Tetwalkers.

7.3.5. Frictions

Frictions are significant part of model design in some NASA reports concerning Tetwalkers [37]. The author, however, thinks that frictions should not be considered as evidenced by natural movements of insects, birds, animals or even humans [35, 36]:

- The low-intelligent insects, for example, do not concern themselves about friction coefficients while moving.
- No animals, including humans, calculate frictions when entering on an icy

ground.

- The highly intelligent species such as humans does not work out frictional values while walking.

Instead of computing frictional values, all species (excluding computers) follow their instinct or heuristic in the circumstances below:

- If the ground is slippery, CG has to be lower for stabilizing the gaits.
- If one slips or tumbles, one simply struggles to recover by moving the limbs.
- If the limbs are stuck, one will exert maximum effort to be unstuck.

The observations above have been applied to the design of gait model developed here; as a result, the gaits are more natural and simple. If frictions are not omitted from the design, one has to tackle the following problems:

- Tetwalkers have limited computing capabilities and data storage, especially in extreme space environments. A large frictional data require more computing resources and storage.
- If the data become part of logics with higher priority, they will inhibit local behaviors and sensors.
- The models that include frictions are complex and difficult to construct.

7.3.6. Vectorized Goals

A vectorized goal is simply a line vector, coupled with sensors, pointing to a target. It has several advantages:

- The line vector with inputs from sensors can change its goals dynamically.
- It can follow waypoints computed from a global path planner by pointing to the closest one at each time step.

- It can point to an object in 3D space as its goal.
- It uses vector algebra for computing its orientation and magnitude.

7.3.7. Tetwalkers' Group Behaviors

The author envisions the inexpensive, expandable Tetwalkers to be employed in large number in future Mars exploration. Thus, a sensible design of group behaviors is important for effective cooperation among group members. The following simple group behaviors are desirable:

- Avoid behavior--it is needed for self-preservation. That is, Tetwalker protects itself before completing its mission. By ensuring self-survival, it indirectly defends the safety of other nearby Tetwalkers. Other related but individualize behaviors are flipping and shifting CG.
- Path-following behavior--this is a high-level behavior, which enables Tetwalkers to move along a preplanned path by following the waypoints that benefit the missions. It has lower priority than avoid behavior since its goals are not immediate.
- Nearest-waypoint behavior--each waypoint provides a high-level goal for Tetwalkers. Tetwalkers will compete for the nearest waypoint but abandon it if other Tetwalkers are dangerously close.

Though there are only three behaviors, the WYSIWYS simulations demonstrate that the group interactions are successful--that is, each Tetwalker is able to avoid obstacles, follow path, and compete for nearest goal dynamically and effectively.

7.3.8. Minimalist

Minimalist can be explained in one simple word--"simplicity." The author believes minimalist is a sensible design to minimize costs, software complexity, the number of hardware components and sensors. At the same time, it is maximizing robustness, independence and intelligence of Tetwalkers. Simplicity, nonetheless, does not mean that the design is easy and straightforward. One the contrary, a great effort, amount of time and ingenuity are necessary to determine the sensible design. The principle is applied in the following examples:

- Virtual experiments are just designed for two basic forces and thus minimizing the proving effort.
- Though Tetwalkers have a few behaviors, they can accomplish complicated tasks successfully. By implementing minimalist in avoid behavior, for instance, Tetwalkers avoid close objects by moving in just one direction (backward) instead of three options (backward, rightward or leftward). From the WYSIWYS simulations, the one direction only yields robust evading.

7.4. Tetwalkers' Simulation and Software Architecture

To fulfill the design concepts above, the author hereby proposes a software architecture that has two main components: gait behaviors and virtual experiments. Gait behaviors, a model, make predictions about movements toward a goal. However, virtual forces, proven via virtual experiments, enforce Newtonian physics on the predictions. Like Navy Swarms, if proven forces validate the predictions, the behavior model is then verified, which in turn, provides WYSIWYS simulations or data.

The gait behaviors and virtual forces described in the following subsections are applicable to N-Tet. Indeed, from the WYSIWYS simulations, 4-Tet is best suited for Mars exploration as it is simpler than 12-Tet with the same payload at its center node--unlike 1-

Tet that has no safe location for payload.

The gait-behaviors layer in Figure 22 can be expanded into three detailed behaviors shown in Figure 23, which are controlled by subsumption scheme for achieving goals.

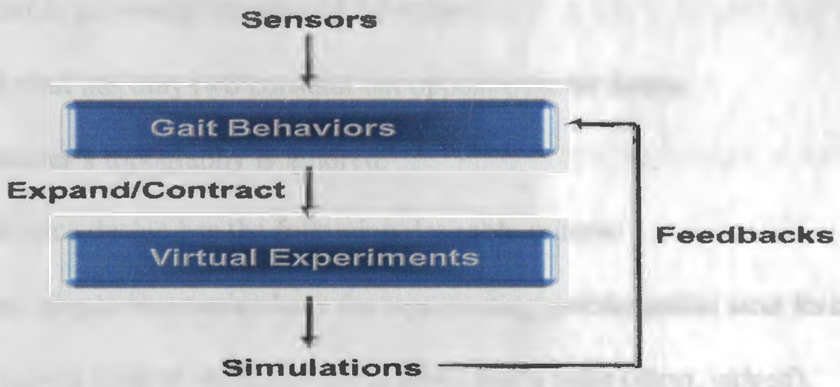


Figure 22. Software Architecture for Expanding or Contracting Struts

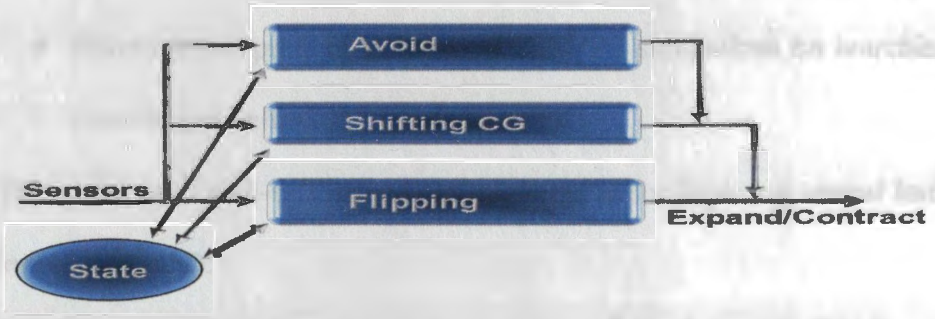


Figure 23. Modified Subsumption Control Scheme for Tetwalkers

The main behaviors perform the following functions:

- Avoid behavior enables each Tetwalker to evade close objects by moving in opposite orientation.
- Shifting CG behavior allows Tetwalker to move by shifting its CG towards a chosen direction. It, nevertheless, does not shift CG to a direction but simply imitates the shape of imaginary Tetwalker who has the given orientation.
- The main purpose of flipping behavior is for restoring Tetwalker to regular tetrahedral shape after falling.

At this early stage of investigation, some restrictions are applied to Tetwalker scenarios:

- The gyroscope, angle and motor sensors are the only three sensing tools needed in providing the state of Tetwalker.
- Each strut has only two constant but opposite motor forces.
- The crater's topography is ignored.

The virtual force design has the following desirable criteria:

- It uses simple Newtonian laws for representing problematical strut forces.
- The source code of virtual forces is about half a page (short, indeed).
- Two simple strut forces are easier to manage, comprehend and verify.
- Proven forces produce valuable WYSIWYS simulations for searching feasible gait behaviors.

The author used well-established Newtonian laws for depicting virtual forces as vector arrows in Java3D:

1. The first law is about the property of inertia. It states that all objects with mass remain in their current state of motions and orientations unless an external force is introduced. A vector arrow can represent an object's current motion (arrow length) and orientation (arrowhead).
2. The second law is the rate of change of momentum, which is depicted by the rate of changing arrow length and/or arrowhead orientation.
3. The third law states that action and reaction are equal but in opposite directions--two arrows with equal lengths but directly opposite orientations capture the law.

The vector arrows embodying the laws are employed extensively in the scenarios, for example:

1. A pair of arrows with their tails at the strut center and arrowheads pointing to both end nodes replaces the expansive forces (first law); the contractive forces are similar but have directly opposite orientations.
2. The expansion/contraction requests from gait behaviors alter the arrow lengths correspondingly (second law).
3. The actions at ground nodes produce two equal but opposing arrows (third law).

One initial but failed gait model strongly suggests that the virtual experiments are indispensable for validating models. The erroneous model is explained in the following subsections and figures.

In Figure 24, the intended goal of the model was to move 1-Tet's CG horizontally without raising its height.

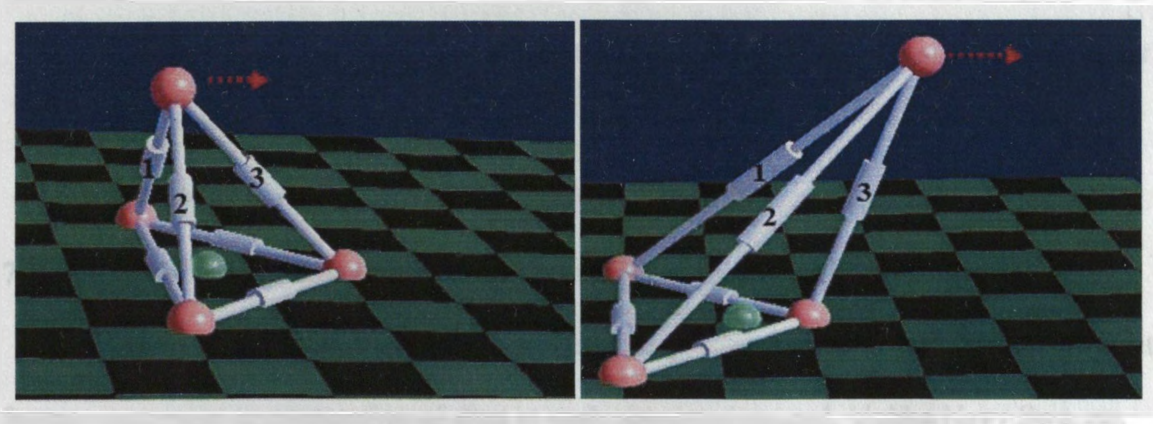


Figure 24. VEDPM Rejecting the Failed 4-Tet's Gaits

As a result, the following suggested gaits would have appeared to be a reasonable solution:

- Only the top node and connected struts (label 1, 2 and 3) were allowed to shift. The nodes and struts on the ground were motionless.

- The model calculated a position x unit length away from top node along the horizontal arrow's orientation.
- It subsequently computed the length differences between current and imaginary struts as lengthen or shorten requests to strut forces.
- The proven forces strictly implement the Newtonian laws on model's requests.

The left picture of Figure 24 shows the gaits worked as intended initially. However, once CG was outside the base triangle, the top node and struts shot up rapidly and unexpectedly as shown in the right picture. The proven forces did not hide the model's weaknesses despite initial confidence on the "commonsense" design. This example demonstrates VEDPM is a valuable tool for the following reasons:

- VEDPM can eliminate unfeasible gait behaviors.
- It improves the model by showing the abnormal behaviors.
- The intended scenes can be verified easily via WYSIWYS simulations and/or numerical data.
- Most importantly, the proven scenes are realistic and applicable in real world.

7.5. Validation of Tetwalker's Virtual Strut Forces via Virtual Experiments

The following subsections describe virtual experiment designs for proving two strut forces--expansion or contraction of strut at each simulated second. Unlike Navy Swarms, the author cannot utilize the well-established circular equation in Tetwalker's virtual experiments due to its tetrahedral structure. This is expected since real experiments in real world require new experiment design when new objects are under investigation. Likewise, VEDPM also emphasizes new design for Tetwalker. The simple and unambiguous design involves great effort, time, and ingenuity. The author, finally, was able to figure out creative experiments that utilize the property of regular tetrahedral:

1. All strut lengths of regular tetrahedral are equal. If a constant expansive or contractive force is applied at each strut, the enlarged or shrinked tetrahedral must maintain the same strut length.
2. If the Tetwalker's CG in Java3D is located directly above the origin (0,0,0), due to perfect symmetry, the CG's x & y coordinates will not change (both zeros) while z coordinate (height) varies proportionately with the strut force.

7.5.1. Proving Virtual Expansive Force via Virtual Experiment

The regular 1-Tet's experiment design, shown in Figure 25, for proving expansive force is as follows:

- The starting regular tetrahedral has 100 cm length at each strut.
- An expansive force of 1.5 cm per second (0.015 m/s) is assigned for each strut.
- The ending regular tetrahedral has 181 cm length at each strut.
- The expansive force is proven if both visual and numerical observations agree with the ending regular tetrahedral above.
- If the state of ending regular tetrahedral is achieved, it implies that source code for the force is bugs free.
- Since the expansive force design applied to N-Tet, a proof on 1-Tet implies the expansive forces in 4-Tet or 12-Tet are valid as well, provided the source code for all proven forces are preserved entirely.

Figure 26 captures the starting and ending states of the virtual expansion experiment. The left picture shows the starting 1-Tet has equal strut lengths of 100 cm. The right picture demonstrates that after 27 seconds, the enlarged regular tetrahedral has equal length of 181 cm. The added length is equal to 1.5 cm/s * 2 * 27 s = 81 cm. Thus, the author has visually confirmed that the enlarged lengths appeared to be equal.

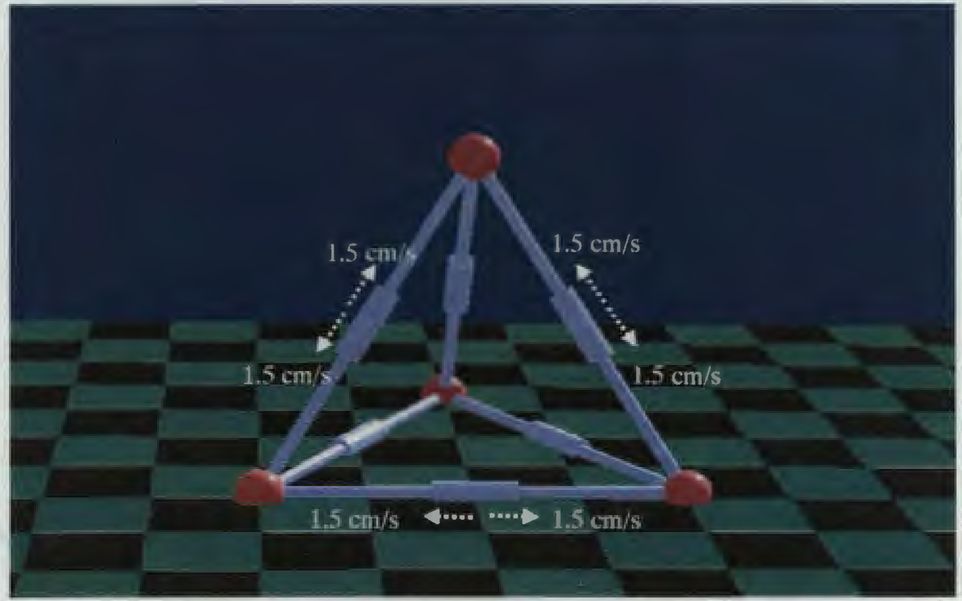


Figure 25. Virtual Experiment for Strut Expansions

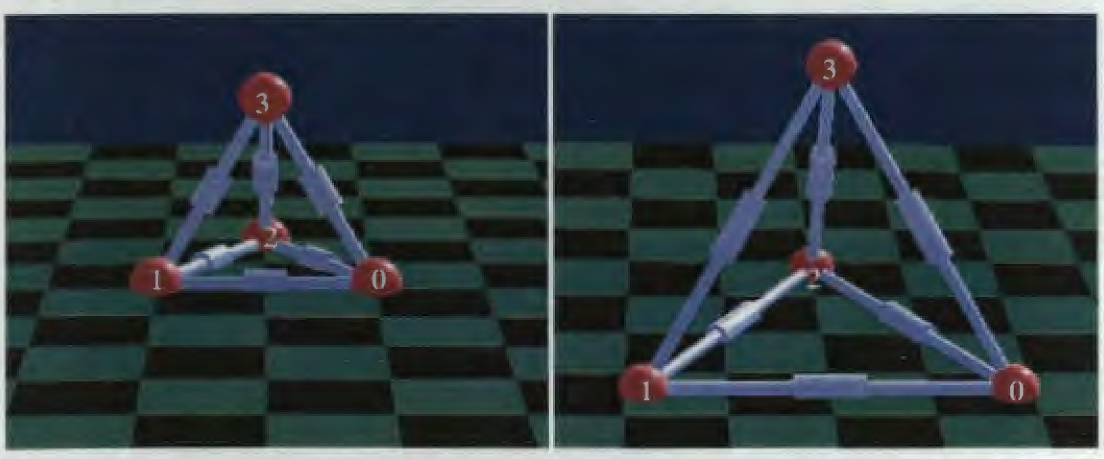


Figure 26. Visual Confirmation of Enlarged Regular Tetrahedral

Visual confirmation is not exact. The experiment, nonetheless, was coded to provide exact numerical data for supporting the visual confirmation as listed in Table 4, which shows the length is exactly 181 cm after 27 seconds for each strut, as expected.

Nth (Seconds)	Strut Length with Node (X,Y), cm					
	(0,1)	(1,2)	(2,0)	(3,0)	(3,1)	(3,2)
0	100	100	100	100	100	100
3	109	109	109	109	109	109
6	118	118	118	118	118	118
9	127	127	127	127	127	127
12	136	136	136	136	136	136
15	145	145	145	145	145	145
18	154	154	154	154	154	154
21	163	163	163	163	163	163
24	172	172	172	172	172	172
27	181	181	181	181	181	181

Table 4. Numerical Values for Expanded Regular 1-Tet

7.5.2. Proving Virtual Contractive Force via Virtual Experiment

The regular 1-Tet's experiment design, shown in Figure 27, for proving contractive force is as follows:

- The starting regular tetrahedral has 181 cm length at each strut.
- An expansive force of 1.5 cm per second (0.015 m/s) is assigned for each strut.
- The ending regular tetrahedral has 100 cm length at each strut.
- The contractive force is proven if both visual and numerical observations agree with the ending regular tetrahedral.
- If the state of ending regular tetrahedral is achieved, it implies the source code for the force is bugs free.

- Since the contractive force design applied to N-Tet, a proof on 1-Tet implies the contractive forces in 4-Tet or 12-Tet are valid as well, provided the source code for all proven forces are preserved entirely.

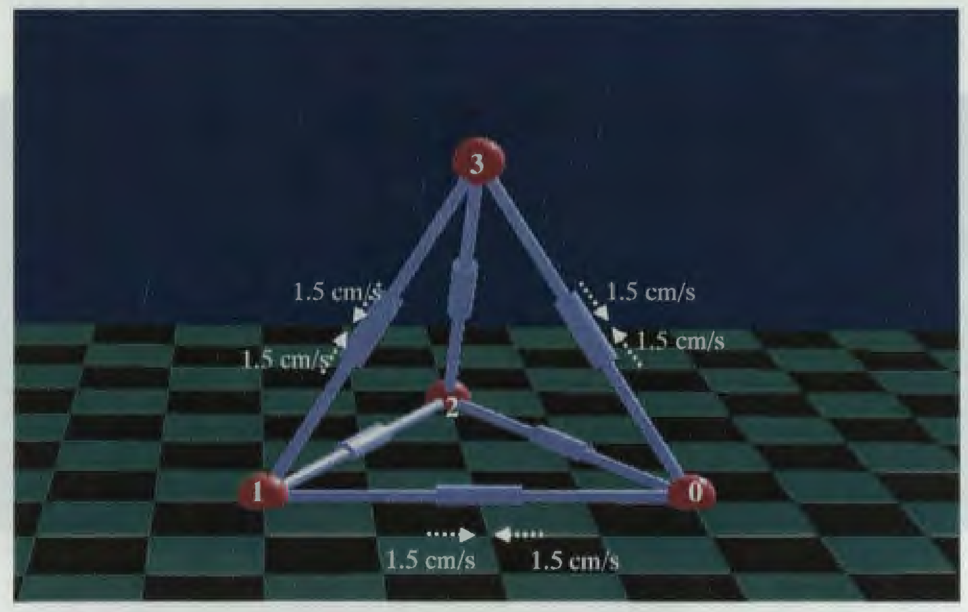


Figure 27. Virtual Experiment for Strut Contractions

Figure 28 captures the starting and ending states of the virtual expansion experiment. The left picture shows the starting 1-Tet has equal strut lengths of 181 cm. The right picture demonstrates that after 27 seconds, the shrinked regular tetrahedral has equal length of 100 cm. The subtracted length is equal to 1.5 cm/s * 2 * 27 s = 81 cm. Thus, the author has visually confirmed that the shrinked lengths seemed to be equal.

Visual confirmation is not accurate. The experiment, nevertheless, was coded to provide exact numerical data for supporting the visual confirmation as listed in Table 5, which shows the length is exactly 100 cm after 27 seconds for each strut, as expected.

7.5.3. Proving Virtual Expansive Force via CG Virtual Experiment

The regular 1-Tet's experiment design for proving virtual expansive force via CG is as follows:

- A regular tetrahedral is placed at the origin in Java3D where CG has coordinate (0, 0, 20.41).
- An expansive force of 1.5 cm per second (0.015 m/s) is assigned for each strut.

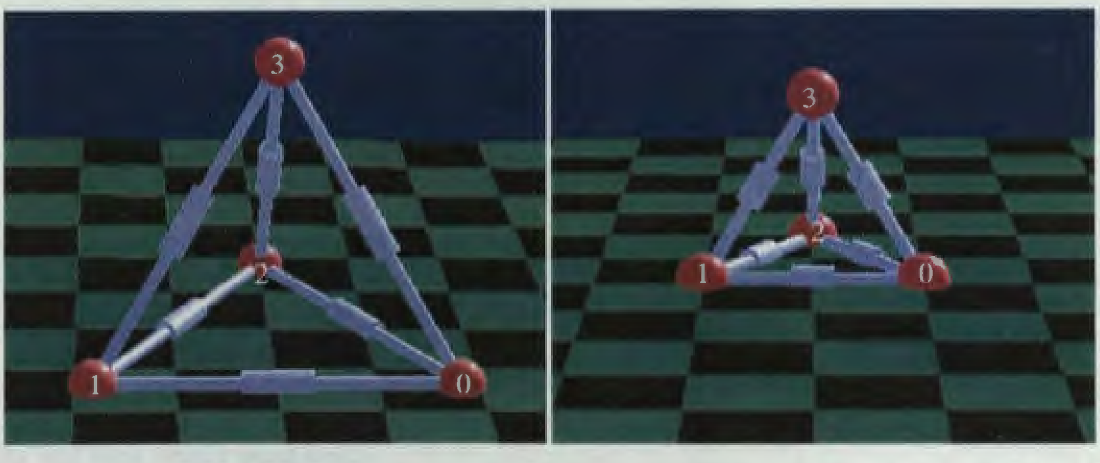


Figure 28. Visual Confirmation of Shrinked Tetrahedral

Nth (Seconds)	Strut Length with Node (X,Y), cm					
	(0,1)	(1,2)	(2,0)	(3,0)	(3,1)	(3,2)
0	181	181	181	181	181	181
3	172	172	172	172	172	172
6	163	163	163	163	163	163
9	154	154	154	154	154	154
12	145	145	145	145	145	145
15	136	136	136	136	136	136
18	127	127	127	127	127	127
21	118	118	118	118	118	118
24	109	109	109	109	109	109
27	100	100	100	100	100	100

Table 5. Numerical Values for Shrinked Regular 1-Tet

- The ending regular tetrahedral has CG at coordinate (0, 0, 36.95).

- The expansive force is proven if both visual and numerical observations agree that CG is at (0, 0, 36.95).
- If CG is at (0, 0, 36.95), it entails the source code for expensive force is bugs free.
- Since virtual expansive force design applied to N-Tet, a proof on 1-Tet infers the expansive forces in 4-Tet or 12-Tet are valid as well, provided the source code for all proven forces are preserved entirely.
- The proven expansive force is an indirect proof that the virtual contractive force, with directly opposite orientation, is valid as well.

CG is computed from the following formula, which involves masses of nodes and struts and their vector positions:

$$CG = (M_1C_1 + M_2C_2 + \dots M_mC_m + m_1c_1 + m_2c_2 + \dots m_nc_n) / (M_1 + M_2 + \dots M_m + m_1 + m_2 + \dots m_n)$$

Where,

m = number of struts

$M_1 + M_2 + \dots M_m$ *= masses for m struts*

$C_1 + C_2 + \dots C_m$ *= CG of each strut represented by a 3D vector point.*

n = number of nodes.

$m_1 + m_2 + \dots m_n$ *= masses for n nodes*

$c_1 + c_2 + \dots c_n$ *= CG of each node represented by a 3D vector point.*

$M_1 + M_2 + \dots M_m + m_1 + m_2 + \dots m_n$ *= total masses of struts and nodes.*

To simplify the formula, the following assumptions are made:

- Each node is a perfect sphere and local CG is exactly at its center. Moreover, the mass is evenly distributed.

- The strut is a perfect rod with even mass, and local CG is exactly at its mid-point.
- The mass of each strut or sphere is exactly 1 kg.

Thus, the simple version of CG formula used in experiments is stated below:

$$C.G. = (C_1 + C_2 + \ldots C_m + c_1 + c_2 + \ldots c_n) / (m + n)$$

CG is therefore the vector sum of each strut CG and node CG. It is a vector as mass gives force, which can be denoted by vector arrows.

Figure 29 captures the starting and ending states of the expansion experiment. The left picture shows the starting 1-Tet's CG (the green ball at 1-Tet's center) is at coordinate (0.0, 0.0, 20.41). The right picture demonstrates that after 27 seconds, the green ball is at (0, 0, 36.95). Therefore, the author has visually confirmed that CG was indeed close to the intended coordinate and x and y values appeared unchanged if one views from the bottom of 1-Tet as evidenced by the green ball being covered by the top node.

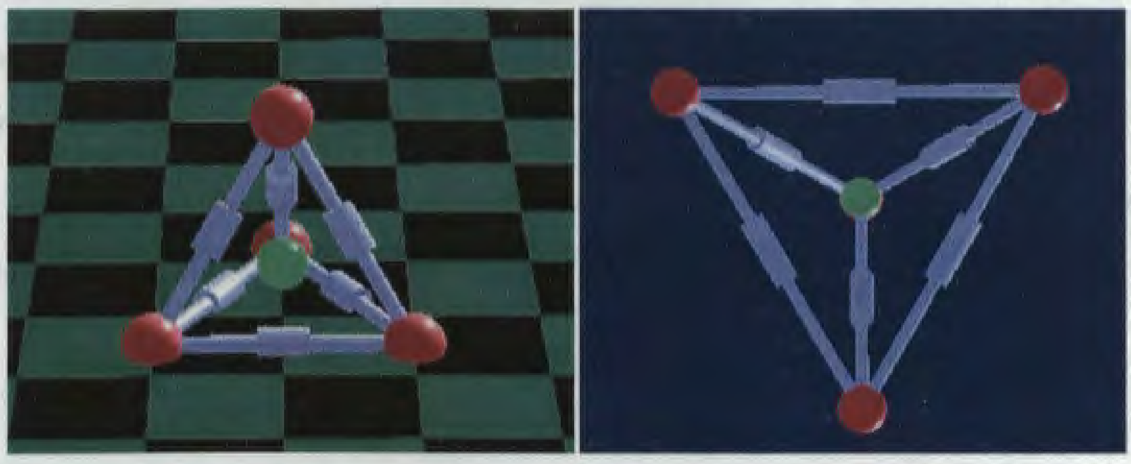


Figure 29. Visual Confirmation of CG

Visual confirmation is not all. The experiment also provides numerical confirmation as listed in Table 6, which demonstrates clearly that CG is moving along the z axis from (0, 0, 20.41) to (0, 0, 36.95) within 27 seconds.

Nth (Seconds)	3D coordinates, (x,y,z)		
	x (cm)	y (cm)	z (cm)
0	0.0	0.0	20.412414
3	0.0	0.0	22.249531
6	0.0	0.0	24.086649
9	0.0	0.0	25.923766
12	0.0	0.0	27.760883
15	0.0	0.0	29.598001
18	0.0	0.0	31.435118
21	0.0	0.0	33.272235
24	0.0	0.0	35.109352
27	0.0	0.0	36.946470

Table 6. Numerical Confirmation of CG

The proof for virtual contractive force via CG is similar to steps above except the force is in opposite direction. Consequently, it is not necessary to construct a virtual experiment for the contractive force, as it can be inferred from the expansive force experiment above.

7.6. Proven Short Source Code via Valid Strut Forces

The proven source code representing both valid strut forces are short and has only 6 Java statements as listed below:

```
1.  if (node[i].z <= 0.0 && node[otherNode[i][j]].z > 0.0) {
          node[otherNode[i][j]].add(strutForce); }
2.  else if (node[i].z > 0.0 && node[otherNode[i][j]].z <= 0.0) {
          strutForce.scale(-1);
3.        node[i].add(strutForce); }
```

```
4.  else {
        node[otherNode[i][j]].add(strutForce);
5.      strutForce.scale(-1.0);
6.      node[i].add(strutForce); }
```

If a strut connects one ground node and one aerial node, statements 1 to 3 move the aerial node and not the ground one. In fact, the force exerts on the ground node is redirected to aerial one via Newton's third law of forces. If a strut connects two aerial nodes, statements 4 to 6 will be executed, which exert equal force on the two nodes.

7.7. A Successful Gait Behavior Model Developed via VEDPM

VEDPM has finally accepted a successful gait model for N-Tets. The model has two parts: an imaginary Tetwalker generated from mathematical equations, and a virtual Tetwalker that mimics the shape of imaginary Tetwalkers. Figure 30, for example, shows the perfect shape of imaginary 12-Tet, which has the following desirable properties:

- The successive gaits are valid--that is, the nodes and struts, when in motion, are not colliding and at safe distance with each other.
- 12-Tet's CG is able to move horizontally, which is effective for CG shifting--the main mechanism of locomotion--as demonstrated by NASA.

The imaginary gaits are bent by rotating all nodes (excluding the base nodes) toward respective horizontal orientations over an axis of rotation, which connects two base nodes, resulted in constant height for moving CG. The algorithm of mathematical gaits is described below:

- The imaginary N-Tet first determines which base strut is closest to a goal.
- It then rotates each non-base node x degrees toward that base strut and

beyond, horizontally.

- Each non-base node moves in a line parallel to the goal's orientation.

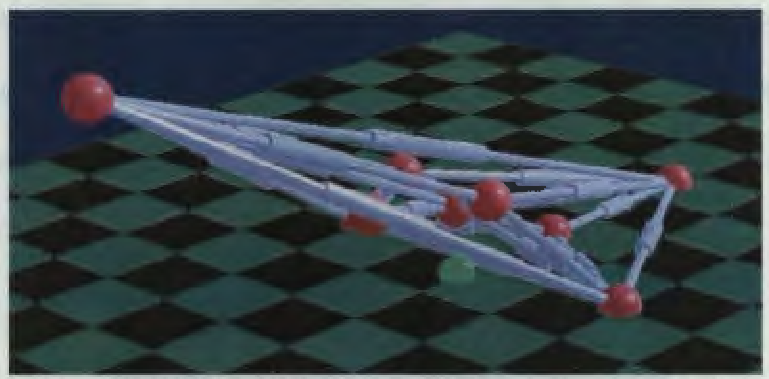


Figure 30. Perfect Shape and Gaits of Imaginary Tetwalker

Figure 31, the semi-transparent nodes are imaginary ones but struts are not depicted due to computing speed. The overall figure, nevertheless, is still recognizable.

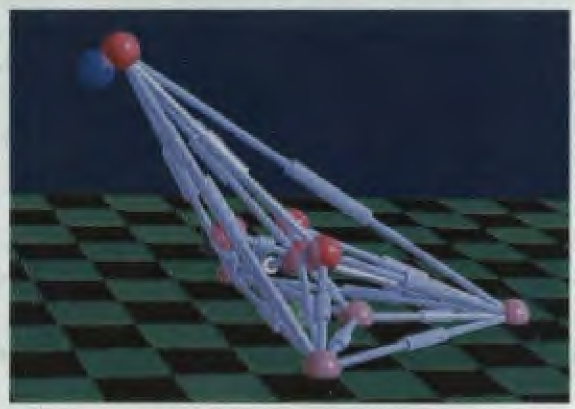


Figure 31. Virtual Tet Chasing Imaginary Tet

The virtual Tetwalker with gait behaviors enforced by proven strut forces are drawn as solid nodes and struts. The solid 12-Tet's goal is to have the same length for each respective strut of Imaginary 12-Tet. To avoid a motionless solid 12-Tet, the goal of achieving the exact strut lengths are not imposed--that is, slacks are allowed for useful motions. From the WYSIWYS simulations, the imaginary Tetwalker was able to lead solid one in achieving its goals. Hence, the mathematical gaits are effective as goal machine.

In Figure 32, the solid 12-Tet has just restored back to its original shape of regular tetrahedral. The reverse process is triggered by the state of tumbling. When solid Tetwalker tumbles, the imaginary one maintains its last mathematical gait. As solid 12-Tet is able to move again, the imaginary one will lead the reverse process. Due to symmetry of regular tetrahedral, the reverse process is possible and implemented effortlessly.

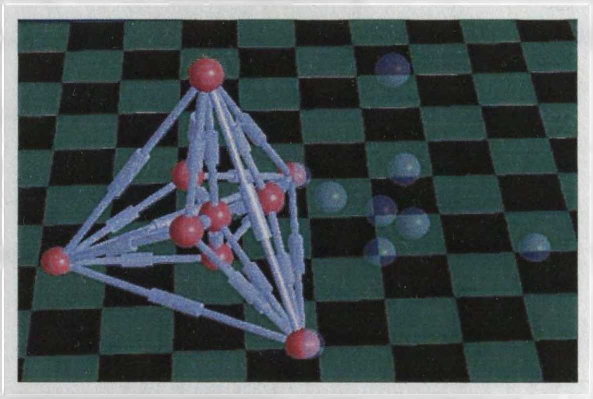


Figure 32. One Complete Gait of Shadow Chasing

The pseudo code for gait behaviors of solid 12-Tet is listed below:

```
identify 3 base nodes
calculate CG
if CG is in base triangle & not recovery {
  imitate the imaginary gait and rotate x degrees in chosen direction
  compute the different lengths between solid & imaginary struts
  pass the differences to strut forces as expansion/contraction requests
    if all struts are close to respective imaginary struts, imitate next gait
    if CG is outside base triangle, exit this if statement
}
else CG is not in base triangle {
    Tumble the 12-Tet
}
else if recovery {
  imitate imaginary gait and rotate x degrees in REVERSE direction
  compute all 26 strut length differences between solid and imaginary struts
  pass the differences to strut forces as expansion/contracting requests
    if all struts are close to respective imaginary struts, imitate next gait
    if it is back to regular tetrahedral, exit this recovery
}
```

The pseudo code for implementing proven strut forces on requests made by gait behaviors:

```
for each strut connected to 2 nodes{
  if one node on ground & other one in air {
    use vector addition method to compute the net displacement on the other node
  }
  if both nodes in the air or on ground {
    use vector addition to compute net displacements on both nodes
  }
}
simulate nodes' positions according to resultant displacements.
simulate net strut lengths nodes
```

7.8. Mathematical Constructs--Vectors and Trigonometry

The virtual forces, nodes, struts, CG and even imaginary Tetwalkers are denoted by vectors and trigonometry, which are matured mathematical constructs for both physics and computer graphics. The Math class in Java provides all required functions of trigonometry for WYSIWYS simulations. The vector algebra is executed by Java3D' Vector3d class methods. By using reliable Java mathematical libraries, the manipulations of vector forces are done automatically. These software tools, nevertheless, must be applied innovatively for solutions demanded by WYSIWYS simulations such as CG, base triangle, imaginary Tetwalker, strut mid-point, rotation, tumbling, etc. Some important Vector class methods are described in the following paragraphs:

- *Vector3d vector_x = new Vector3d()*--this creates a vector object that represents a node's displacement, a strut force, a CG, a goal direction, etc.
- *vector_x.add(vector_y)*--this method implements vector addition on vector_x and vector_y and stores the net vector in vector_x. It is useful for computing the net force experienced by a node, for example.
- vector_x.sub(vector_y)--this method implements vector subtraction on vector_x and vector_y and stores the net vector in vector_x. It is suitable for calculating the differences of two vectors like the node displacements, which is represented by two vector positions, for instance.
- vector_x.angle(vector_y)--this method computes the angle between vector_x and vector_y, and the resulted angle is stored in vector_x. It is valuable for building models that involve trigonometry functions. For example, rotating nodes require angles between 2 vectors.
- vector_x.cross(vector_x, vector_y)--this method performs cross product on vector_x and vector_y and stores the resultant orthogonal one in vector_x. It is employed for computing an orthogonal vector between base strut and

normal.

The algorithms described in pseudo code below, coupled with the methods above, are useful mathematical models for the experiments, simulations, and gait model.

The mathematical model for computing struts:

```
for each node vector x {
   for each node vector y {
      strut  vector = vector subtraction of x and y
   }
}
```

Mathematical model for computing CG:

```
for each node vector{
   CG vector = vector addition of node vector x
   for each strut vector{
      mid-point vector = 0.5 * strut vector
      CG vector = vector addition of mid-point vector
   }
}
CG vector = CG vector / the sum of nodes and struts
```

Mathematical model for determining 3 base nodes:

```
for each node vector {
   if node vector x is on the ground & is corner node & baseNode array size < 3
      baseNode vector array = assign vector node x to array
      if baseNode array size > 2, exit this loop
}
```

The mathematical model for determining CG in base triangle:

```
for each baseNode vector x {
    for each baseNode vector y {
        X vector = vector subtraction of CG & baseNode x
        Y vector = vector subtraction of CG & baseNode y
        degrees = X.angle(Y)
    }
}
if degrees = 360 degrees, CG is inside base triangle
else CG is outside base triangle
```

The Mathematical model for determining which base strut is closest to a goal:

```
for each baseNode vector x {
    for each baseNode vector y {
        X vector = vector subtraction of baseNode x & y
        direction vector = vector cross product of X and unit z axis
        Y vector = vector subtraction of baseNode x & CG
        if CG in base triangle & Y.angle(direction)<= 90 degrees
            maintain direction
        else reverse the direction vector
        if direction.angle(goal vector) is the smallest, baseStrut vector = X vector.
    }
}
```

```
the chosen strut is baseStrut vector
```

The mathematical model for rotating an imaginary Tetwalker:

```
for each node vector {
    find which base strut vector is axis of rotation
    find mid-point vector on the base strut
    project each node vector on the ground
    ground vector = vector subtraction of base strut & mid-point
    angle = direction.angle( ground vector)
    if angle > 90 degrees {
        angle2 = ground.angle(base strut)
        use angle2, ground and base strut to find the third side of triangle.
        use third side, node vector to find the radius vector of circular path
        choose an angle of rotation
        find the horizontal distance given by this angle
    }
}
```

The mathematical model for tumbling a virtual Tetwalker:

```
for each node vector {
    find which base strut vector is axis of rotation
    find mid-point vector on the base strut
    project each node vector on the ground
    ground vector = vector subtraction of base strut & mid-point
    angle = direction.angle( ground vector)
```

```
      if angle > 90 degrees {
          angle2 = ground.angle(base strut)
          use angle2, ground and base strut to find the third side of triangle.
          use third side, node vector to find the radius vector of a circular path
          choose an angle of rotation
          find the height given by this angle
          find the horizontal distance given by this angle
      }
      if angle <= 90 degrees {
          angle2 = ground.angle(base strut)
          use angle2, ground and base strut to find the third side of triangle.
          use third side, node vector to find the radius vector of a circular path
          choose an angle of rotation
          find the height given by this angle
          find the horizontal distance given by this angle
      }
      use vector addition of height and horizontal vector
     use resultant displacement to rotate the node
  }
  if node hits the ground, stop the rotation
```

The mathematical model for computing virtual strut forces:

```
  for each node i vector {
      for each node j vector {
          compute the resultant displacement of strut i, j
```

```
        if node i on ground & base node & node j in air {
            applying vector addition on node j's displacement
        }
        else if node i in air & node j on ground & base node {
            applying vector addition on node i's displacement
        }
        else{
            divide displacement by half
            applying vector addition of displacement equally on node i and j
        }
    }
}
```

The major mathematical models are mentioned briefly above. The author ignores minor ones for shorter chapter. The readers, nonetheless, can sense the importance and power of these models.

7.9. WYSIWYS Simulation Demos

The author has developed the WYSIWYS simulations implementing the Tetwalkers' scenarios discussed in section 7.1. Though words like WYSIWYS, VEDPM, virtual experiments, proven forces, etc. have been mentioned many times, the author wishes to hammer the concepts again into readers' mind, as VEDPM principles are unique and subtle.

The WYSIWYS simulations are realistic and applicable in real world as they are PROVEN by the following sound validation steps:

1. The WYSIWYS simulations are generated by the proven gait behavior model.

2. The gait model is enforced by the proven forces.
3. The proven forces are tested and verified rigorously via virtual experiments.
4. The virtual experiments are supported by visual and numerical confirmations.

The following 4 sets of WYSIWYS demos (each set has two pictures) are demonstrating the solution for N-Tets' (focused on 4-Tets and 12-Tets) scenarios:

5. The first set is demonstrating both 4 and 12-Tet are able to match the respective strut lengths (their goal) of imaginary Tetwalkers successfully as evidenced by their spontaneous animations.
6. The second set is demonstrating both simulations are able to save computing resources for the same scenarios in (1).
7. The third set is demonstrating both 4 and 12-Tets can traverse a flat path formed by waypoints, avoid close encounters autonomously, compete for every waypoint and achieve the final goal of reaching the destination.
8. The fourth set is similar to (3) except the waypoints are laid over a crater.

The following is the detail description of each set. The first set is Figure 33, which shows both 4-Tet and 12-Tet have the following details related to their respective imaginary Tetwalkers, and how they move:

- The transparent nodes reflect partial structures of imaginary Tetwalkers.
- The solid Tetwalkers represent the virtual struts and nodes.
- The imaginary ones generate a specific shape from mathematical equations for leading the solid Tetwalkers.
- The solid Tetwalkers, oblivious to the goals of the imaginary ones, imitate the specific shape within certain slacks. In the simulations, both solid Tetwalkers

are clearly driven by the imaginary ones, respectively.

- After each tumbling, the imaginary Tetwalkers will resume to their original regular tetrahedral shape by following the imaginary shapes in reverse order.
- The graphics depict both solid and transparent Tetwlakers is expensive. Thus, multiple threads from multi-core processor represent various parts of Tetwalkers. The nodes and struts, therefore, are not perfectly synchronized, which is immaterial or irrelevant to experimental results.

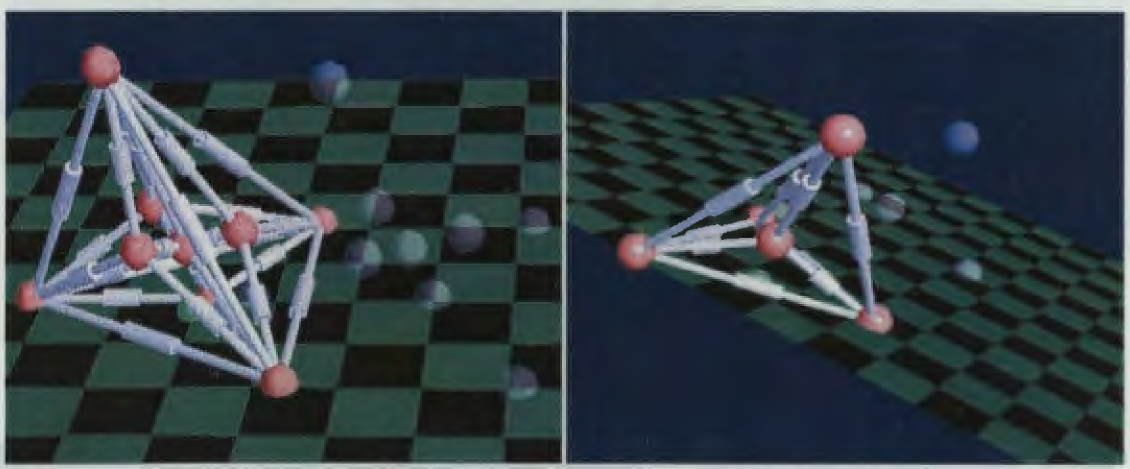


Figure 33. Virtual Experiments on both Virtual & Imaginary Tets

The simulations in Figure 34 are similar to previous two except:

- The simulations are faster without depicting the nodes.
- The imaginary Tetwalkers are not displayed.
- Without the nodes and imaginary Tetwalkers, and fewer threads, the simulations are more efficient in using computing resources.

Figure 35 illustrates the 4-Tet and 12-Tet groups can perform the following:

- They follow waypoints generated from global path planner.
- They can reach each waypoint despite their triangular bases.

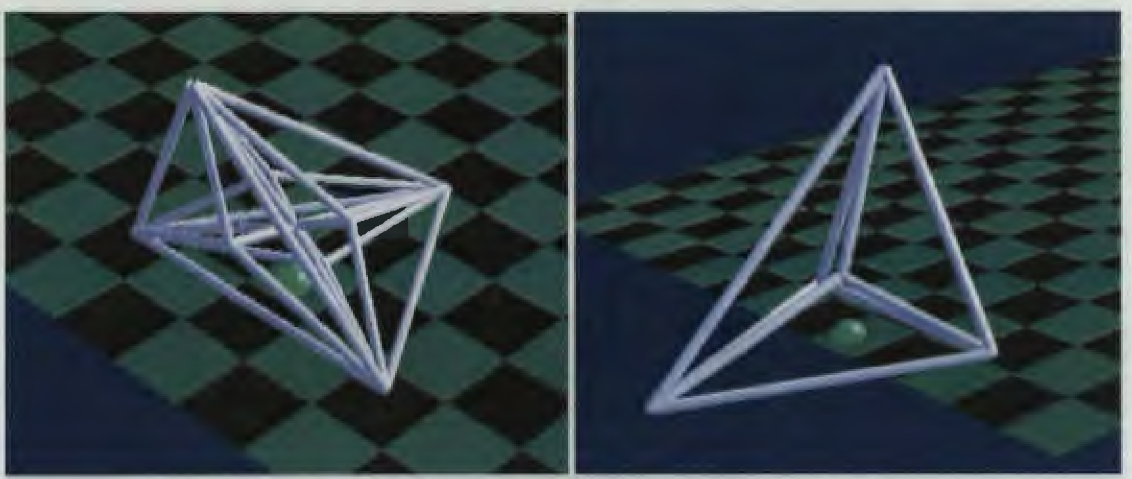


Figure 34. A Tet without Nodes for Fast Simulation

- They rush to the first waypoint but avoid close objects effectively.
- If the waypoint is reached, the process is repeated for the next one.
- The avoid behavior is reactive, robust and unpredictable despite only one avoiding orientation, as evidenced by the WYSIWYS simulations.
- They can select an orientation that is closest to the goal autonomously.

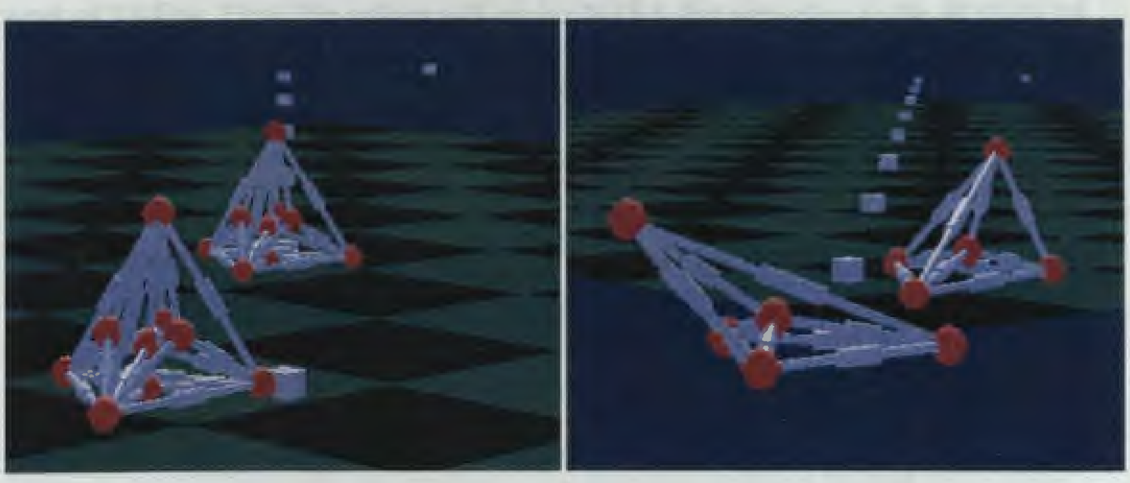


Figure 35. Tet Groups Following Flat Waypoints

The WYSIWYS simulations prove that the autonomous, reactive group behaviors are feasible, which imply potential applications for exploring the unknown Martian environments.

The simulations in Figure 36 are similar to previous two except:

- Tetwalkers can traverse the projected path on the chessboard from the waypoints laid over the crater.
- This ability is helpful in ascending or descending a slope in future work.

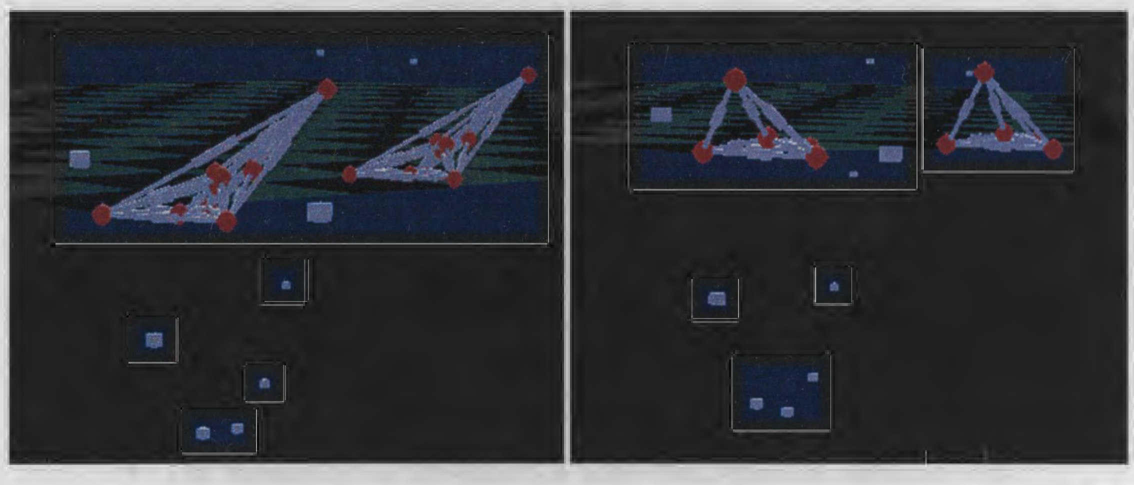


Figure 36. Tet Groups Following Curved Waypoints

From the eight WYSIWYS simulations, it is obviously that 4-Tets can do all the work of 12-Tets. Thus, this author will advise NASA that according to the WYSIWYS simulations, 4-Tets should be chosen over 12-Tets as they have the following benefits:

- The software has less complexity.
- The node and strut structures are simpler.
- They use less computing resources.
- They have fewer motors and thus use less energy.
- They have fewer mechanical problems due to fewer parts.
- They carry the same payload.
- They are more cost effective.
- They are easy to operate.

7.10. Strengths and Weaknesses

The Tetwalker scenarios have the following strengths:

- The animated gaits are supported by the proven forces.
- The proven forces are implemented by short, simple and verifiable source code.
- The 1-Tet simulations behave closely to the real one demonstrated by NASA.
- The gait model is applicable to N-Tet.

The weaknesses are listed below:

- Tetwalkers cannot ascend or descend a slope.
- The gaits are not tested for rugged surface.
- The momentum of Tetwalkers is conserved, which is unrealistic but immaterial.
- There is no gravitational force acting upon Tetwalkers while tumbling. The simulations, nevertheless, ensure the rotating shape is realistic according to the properties of rigid body.

7.11. Conclusion for Tetwalkers

The strut forces that drive WYSIWYS demos have been proven rigorously via virtual experiments. In addition, the forces are short in coding and implementing Newtonian forces. The WYSIWYS simulations show that the gaits of 1-Tet are close to the real ones, which prove that VEDPM is a powerful tool for developing realistic simulations that enable the author to recommend 4-Tets to NASA instead of 12-Tets.

CHAPTER 8. CONCLUSION AND FUTURE WORK

8.1. Summaries on Objectives

The objectives listed in Chapter 1 may be too ambitious. The author is confident that they have been achieved based on the reasoning presented below:

1) To prove that VEDPM is a powerful tool in validating scientific simulations.

   The author has provided four examples on why VEDPM, indeed, is powerful:

   - The gravity and light forces of Solar Sails were successfully proven by the prototype of VEDPM.
   - Likewise, the prototype was able to determine the elusive value of gravitational constant for the third significant digit correctly.
   - Full-fledged VEDPM verified Navy Swarms from the ground up for virtual laws, experiments, behavior models, and simulations.
   - Full-fledged VEDPM validated Tetwalkers from the ground up for virtual laws, experiments, behavior models, and simulations.

2) To demonstrate that scientific method like virtual experiments can be constructed via VEDPM.

   The constructions of sensible virtual experiments are explained in detail in Chapter 6 & 7. The experimental simulations and/or numerical data strongly suggest that the scientific method employed in VEDPM is sound.

3) To illustrate virtual laws require innovative minds to create.

   The author emphasizes on how to choose, modify, and create equations or laws innovatively in several cases below:

   - The author chose the simple yet effective equations in Solar Sails. In particular, the author has ingeniously modified the light force equations.

- The author has creatively proposed an interesting equation that has unlimited significant digits for G value, and using proven gravity code to test that value.
- The author chose wisely the simple velocity vectors instead of complicated aerodynamic formulas.
- The author chose simple velocity vectors instead of complex hydraulic equations.

4) To prove that VEDPM is feasible via two applications.

   The WYSIWYS simulations proved that VEDPM were feasible in Navy Swarms and Tetwalkers.

5) To build WYSIWYS simulations.

   The WYSIWYS simulations were demonstrated successfully in Navy Swarms and Tetwalkers.

6) To provide general software architecture and a modified subsumption control scheme for swarm behavior model.

   The software architecture and subsumption control scheme were constructed successfully, which are proven in WYSIWYS simulations.

7) To demonstrate that the autonomous behaviors can be switched via a simple control signal.

   This was demonstrated in Navy Swarms' simulations where behaviors were switched autonomously via control signals as simple as value 0 and 1 without compromising local behaviors.

8) To show that emergent intelligence is possible at swarm level.

   The Navy Swarms' simulations demonstrated that the numerous, chaotic, and autonomous UAVs appeared to be intelligent emerged from their behaviors and interactions.

9) To explain that behavior models coupled with VEDPM are powerful tools for developing applications for unmanned vehicles.

   The impressive WYSIWYS simulations proved that both tools are ideal for developing applications for unmanned vehicles such as UAVS, ULVs, UWVs, and USVs.

8.2. Future Work of Universal Gravitational Constant

The first future work, testing gravitational constant, is unintended one since the author did not develop virtual laws for the constant but Solar Sails. It will be interesting to know whether the proposed formula, $G = \pi/(\sqrt{3}*e) * 10^{-10}$, is the correct one. If so, the unlimited significant digits may be utilized to calculate large-mass phenomena such as black holes. The data in Table 2 were collected manually by running each G candidate separately. This, of course, is slow and can be improved by rewriting the source code. If further virtual experiments indicate that it does not have the lowest relative ratio, some improvement steps are possible:

- The simulations' time step is one second; a smaller one, like nanosecond, will provide more accurate G.
- Instead of computing the relative error ratios at last second, the code can be rewritten to compute the "average relative error ratios" of each second.
- If the relative error ratio scheme fails, other types of errors may be possible.

8.3. Future Work of Navy Swarms

The work in Navy Swarm is minimal in this dissertation for demonstration of VEDPM. The future work can include other behaviors that are needed for an effective attack on the carriers. Some useful behaviors may be the following:

- Attack--obviously, the attack behavior must be included for destroying a target; otherwise, the swarms are just doing reconnaissance.
- Self-destruction--sometimes the missions are cancelled due to changes of political situations. With this behavior, the swarms are destroyed by themselves.
- Returning--if the UAVs are expensive and self-destruction is not favor, the returning behavior then can guide them back to a base.

8.4. Future Work of Tetwalkers

Though Tetwalkers appear more complicated than Navy Swarms, they have only three behaviors: avoiding, shifting, and tumbling. Additional behaviors may be helpful in Mars exploration:

- Crater descending--this behavior enables Tetwalker to descend the slope of the crater.
- Crater ascending--this behavior enables Tetwalker to ascend the slope of the crater.
- Switchable behaviors--they are useful when earth controllers want to affect Tetwalkers' behaviors for different goals.

8.5. Conclusion of Dissertation

From the two applications developed meticulously via VEDPM, it is clear that

VEDPM is feasible for scientific simulations. This dissertation also demonstrates how to prove virtual laws via virtual experiments rigorously. The proven code is short: 9 Java statements for Navy Swarms and 6 statements for Tetwalkers. VEDPM is a valuable and powerful tool for proving and constructing realistic scientific simulations.

REFERENCES CITED

[1] K, Auer and T. Norris, ""ArrierosAlife" a Multi-Agent Approach Simulating the Evolution of a Social System: Modeling the Emergence of Social Networks with "Ascape"," *Journal of Artificial Societies and Social Simulation*, Vol. 4:1, 2001.

[2] P. Davidsson, "Agent Based Social Simulation: A Computer Science View," *Journal of Artificial Societies and Social Simulation*, Vol. 5:1, 2002.

[3] X. Li, W. Mao, D. Zeng, F. Y. Wang, "Agent-Based Social Simulation and Modeling in Social Computing," Lecture *Notes in Computer Science 5075/2008*, 2008.

[4] J. Pavon, C. Sansores and J. J. Gomez-Sanz, "Modelling and Simulation of Social Systems with INGENIAS," *International Journal of Agent-Oriented Software Engineering*, Vol. 2:2, pp. 196-221, 2008.

[5] P. Terna, "Simulation Tools for Social Scientists: Building Agent Based Models with SWARM," *Journal of Artificial Societies and Social Simulation*, 1:2, 1998.

[6] B. W. Boehm, "Software Risk Management," IEEE Computer Society Press, 1989

[7] C. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graph*, Vol. 21:4, pp. 25-34, 1987.

[8] M. Daniels, "Integrating Simulation Technologies With Swarm," *Proceedings of the Workshop on Agent Simulation: Applications, Models, and Tools*, University of Chicago, 1999.

[9] E. Bonabeau, M. Dorigo, and G. Théraulaz, "Swarm Intelligence: From Natural to Artificial Systems," *Oxford, University Press*, 1999.

[10] E. Bonabeau, and G. Théraulaz, "Swarm Smarts," *Scientific American*, pp. 72-79, 2000.

[11] M. Resnick, “Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds,” Cambridge, MA, *MIT Press*, 1994.

[12] D. Terzopoulos, X. Tu, and R. Grzeszczuk, "Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World," *Artificial Life,* Vol. 1:4, pp. 327–351, 1994.

[13] D. P. Gillen, and D. R. Jacques, "Cooperative Behavior Schemes for Improving the Effectiveness of Autonomous Wide Area Search Munitions," *Proceedings of the Cooperative Control Workshop*, 2000.

[14] K. Passino, M. Polycarpou, D. Jacques, M. Pachter, Y. Liu, Y. Yang, M. Flint, and M. Baum, "Cooperative Control for Autonomous Air Vehicles," *Proceedings of the Cooperative Control Workshop*, 2000.

[15] J. Fredslund and M. J. Mataric, "A General, Local Algorithm for Robot Formations," *IEEE Transactions on Robotics and Automation, special issue on Multi Robot Systems*, Vol. 18:5, 2002.

[16] B. B. Werger, "Cooperation Without Deliberation: A Minimal Behavior-based Approach to Multi-robot Teams," *Artificial Intelligence*, Vol. 110:2, pp. 293-320, 1999.

[17] M. J. Mataric, "Designing and Understanding Adaptive Group Behavior," *Adaptive Behavior*, Vol. 4:1, pp. 51-80, 1995.

[18] C. R. Kube, and H. Zhang, "Collective Robotic Intelligence," *Second International Conference on Simulation of Adaptive Behavior*, pp. 460-468, 1992.

[19] K. Altenburg, J. Schlecht, and K. E. Nygard, "An Agent-based Simulation for Modeling Intelligent Munitions," *Proceedings of the WSEAS Conference*, 2002.

[20] C. A. Lua, K. Altenburg and K. E. Nygard, "Synchronized Multi-Point Attack by Autonomous Reactive Vehicles with Simple Local Communication," *Proceedings of IEEE Swarm Intelligence Symposium*, pp. 95-102, 2003.

[21] C. A. Lua, K. Altenburg and K. E. Nygard, "ANTS with Firefly Communication," *Proceedings of the 2005 International Conference on Artificial Intelligence (ICAI)*, 2005.

[22] R. Blomquist, "Solar Blade Nanosatellite Development: Heliogyro Deployment, Dynamics, and Control," *Proceedings of the 13th USU / AIAA Small Satellite Conference*, 1999.

[23] R. A. Brooks, "A Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. 2:1, pp. 14-23, 1986.

[24] M. A. Arbib, "Perceptual Structures and Distributed Motor Control, in Handbook of Physiology, Section 2: The Nervous System," Vol. II, Motor Control, Part 1, V. B. Brooks (ed.), *American Physiological Society*, pp. 1449-1480, 1981.

[25] R. C. Arkin, "Motor Schema-based Mobile Robot Navigation," *The International Journal of Robotics Research*, pp. 92-112, 1989.

[26] J. R. Andrews, and N. Hogan, "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator," D. E. Hardt and W. J. Book (Eds.), *Control Of Manufacturing Processes and Robotic Systems,* ASME, Boston, pp. 243-251, 1983.

[27] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, Vol. 5:1, 1986.

[28] B. H. Krogh, "A Generalized Potential Field Approach to Obstacle Avoidance Control," *International Robotics Research Conference*, 1984.

[29] S. Curtis, "Stepping Video," Goddard Space Flght Center, accessed: Aug. 13, 2010. http://ants.gsfc.nasa.gov/features/steppin.MPG.

[30] S. Curtis, "12-Tet Video," Goddard Space Flght Center, accessed: Aug. 3, 2010. http://ants.gsfc.nasa.gov/features/12tet_totalact_medium.mov.

[31] P. E. Clark, S. Kessel, M. L. Rilee, G. Brown, C. Cooperrider, S. A. Curtis, "Extreme Mobility: Gaits for Tetrahedral Rovers," *38th Lunar and Planetary Science Conference*, LPI Contribution No. 1338, pp.1172, 2007.

[32] S.A. Curtis, W. F. Truszkowski, M. L Rilee and P.E. Clark, "ANTS for the Human Exploration and Development of Space," *Proceedings of the 2003 IEEE Aerospace Conference*, 2003.

[33] P. E. Clark, M. L. Rilee, S. A. Curtis, C. Y. Cheung, G Marr, W. Truszkowski and M. Rudisill, "LARA: Near Term Reconfigurable Concepts and Components for Lunar Exploration and Exploitation," *IAC Proceedings*, 2004.

[34] P. E. Clark, S. A. Curtis, M. L. Rilee and S. R. Floyd, "ALI (Autonomous Lunar Investigator): Revolutionary Approach to Exploring the Moon with Addressable Reconfigurable Technology," *Lunar and Planetary Science XXXVI*, 2005.

[35] P. E. Clark, S. A. Curtis, M. L. Rilee, W. Truszkowski, G. Marr, C.Y. Cheung and M. Rudisill, "BEES for ANTS: Space Mission Applications for the Autonomous NanoTechnology Swarm," *AIAA Intelligent Systems Technical Conference*, 2004.

[36] P. E. Clark, M. L. Rilee, S. A. Curtis, C. Y. Cheung, G. Marr, W. Truszkowski and M. Rudisill, "PAM: Biologically inspired engineering and exploration mission concept, components, and requirements for asteroid population survey," *IAC Proceedings*, 2004.

[37] R. P. Wesenberg, "Addressable Re-configurable Technology(ART) Tetrahedral

Robotics," *NASA Goddard Space Flight Center*, 2007.

[38] T. Vicsek, "Complexity: The Bigger Picture," *Nature*, Vol. 418, pp. 131, 2002.

[39] G. Fagiolo, C. Birchenhall and P. Windrum (Eds.), Special Issue on "Empirical Validation in Agent-Based Models," *Computational Economics*, Vol. 30:3, 2007.

[40] K. G. Troitzsch, "Validating Simulation Models," *Proceedings of the 18th European Simulation Multiconference*, 2004.

[41] P. Windrum, G. Fagiolo and A. Moneta, "Empirical Validation of Agent-Based Models: Alternatives and Prospects," *Journal of Artificial Societies and Social Simulation*, Vol. 10:2 8, 2007.

[42] V. R Kamat and J. C. Martinez, "Validating Complex Constrution Simulation Models Using 3D Visualization," *Systems Analysis Modelling Simulation*, Vol. 43:4, pp. 455-467, 2003.

[43] R. D. Newman and M. K. Bantel, "On Determining G Using a Cryogenic Torsion Pendulum," *Meas. Sci. Technol.*, Vol. 10, pp. 445-453, 1999.

[44] M. K. Bantel and R. D. Newman, "A Cryogenic Torsion Pendulum: Progress Report," *Class. Quantum Grav.*, Vol. 17, pp. 2313-2318, 2000.

[45] M. K. Bantel and R. D. Newman, "High Precision Measurement of Torsion Fiber Internal Friction at Cryogenic Temperatures," *Journal of Alloys and Compounds*, Vol. 310, pp. 233-242, 2000.

[46] K. Kuroda, "Does the Time-of-Swing Method Give a Correct Value of the Newtonian Gravitational Constant?" *Phys. Rev. Lett.*, Vol. 75:15, pp. 2796-2798, 1995.

[47] A. Thompson and B. N. Taylor, "Special Publication 811: Guide for the Use of the International System of Units (SI)," *National Institute of Standards and Technology*

(NIST), 2008.

[48] International Organization for Standardization, "Quantities and units - Part 3: *Space and time*," ISO 80000-3:2006, 2006.

[49] E. R. Cohen and B. N. Taylor, *Rev. Mod. Phys.*, Vol. 59, pp. 1121, 1987.

[50] Google Search, "Lua: Synchronized multi-point attack by autonomous reactive vehicles with simple local communication," *Google Scholar*, accessed: Aug. 3, 2010. http://scholar.google.com/scholar?cites=4807244610863306992&hl=en&as_sdt=80000000000.