# Automatic Parameter Tuning for the Morpheus Vehicle Using Particle Swarm Optimization

B. Birge, PhD[1]
*L-3 STRATIS Division/NASA JSC Engineering Directorate, Houston, Texas, 77058*

**A high fidelity simulation using a PC based Trick framework has been developed for Johnson Space Center's Morpheus test bed flight vehicle. There is an iterative development loop of refining and testing the hardware, refining the software, comparing the software simulation to hardware performance and adjusting either or both the hardware and the simulation to extract the best performance from the hardware as well as the most realistic representation of the hardware from the software. A Particle Swarm Optimization (PSO) based technique has been developed that increases speed and accuracy of the iterative development cycle. Parameters in software can be automatically tuned to make the simulation match real world subsystem data from test flights. Special considerations for scale, linearity, discontinuities, can be all but ignored with this technique, allowing fast turnaround both for simulation tune up to match hardware changes as well as during the test and validation phase to help identify hardware issues. Software models with insufficient control authority to match hardware test data can be immediately identified and using this technique requires very little to no specialized knowledge of optimization, freeing model developers to concentrate on spacecraft engineering. Integration of the PSO into the Morpheus development cycle will be discussed as well as a case study highlighting the tool's effectiveness.**

## I. Introduction

### A. The Morpheus Vehicle

The Morpheus project is a concept vehicle designed by NASA and based at Johnson Space Center (JSC).[1] The purpose is for use as a vertical test bed demonstrator for autonomous landing and hazard avoidance as well as "green" propellant propulsion systems.

Historically the vehicle has evolved from an earlier design called Project M that was developed for lunar exploration and designed to launch on an Atlas V rocket. This was merged with the Autonomous Lander and Hazard Avoidance Technology (ALHAT) project, which is a technology demonstrator furthering efforts towards precision landing. Though no longer lunar-centric, the current Morpheus project has inherited many of the baseline specifications of Project M.

Manufactured by JSC engineers in partnership with Armadillo Aerospace, the whole project has been designed, developed, and managed in-house at NASA JSC. In addition to performing as a technology demonstrator for tank material and manufacture, reaction control thrusters, main engine performance improvements, helium pressurization systems, etc. it is also being used as an exercise in "lean development" engineering practices.

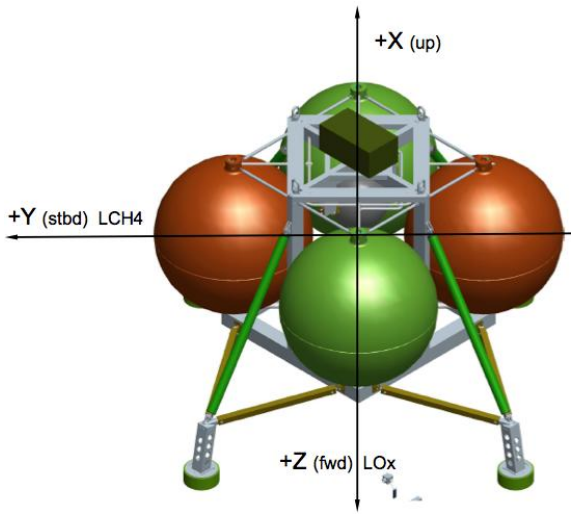


**Figure 1. Morpheus Spacecraft.** *Schematic showing the physical layout of the vehicle.*

[1] Aerospace Engineer, Software, Robotics, and Simulation Division, Spacecraft Software Systems Engineering Branch, 1002 Gemini, Suite 200, Houston, TX, 77058, AIAA Regular Member.

Morpheus uses a liquid methane / liquid oxygen combination for propellant which has a number of advantages. One advantage is that methane can be manufactured from lunar or Martian ice, making in-situ exploitation capabilities a feature of future missions. Another is relative ease of handling, less cryogenic concerns, compared to liquid hydrogen for example.

Conceptually the vehicle is being proposed for rendezvous and fuel transfer studies in low earth orbit as well as asteroid rendezvous. The precision autonomous landing and hazard avoidance technology is a key part of the vehicle. So far Morpheus has flown on several tethered flights; free flight has yet to be tested. Testing is done at the JSC Vertical Test Bed Flight Complex (VFC).

Morpheus is a complete spacecraft. It has avionics, software, guidance, navigation and control, power and associated distribution, structures, propulsion, and instrumentation. The part of Morpheus that this paper addresses is the control response of the gimbals' actuators that determine the thrust vectoring of the propulsion system.

### B. Trick

Trick© is a simulation framework and library that allows rapid development of real time modeling as well as high fidelity analysis. It was developed by L-3 Communications and is in use in many facets of JSC where a high level simulation is required. For example Trick is used as the simulation framework for astronaut training of the robotic arms on the Space Shuttle and the International Space Station. It is C based, more recent revisions have migrated to C++ and Python hybrids, and it is highly distributed with tools for running simulations across labs, centers, etc. Trick is data driven. All parameters that define the dynamic simulation can be specified at the start in user configurable input files. The high fidelity vehicle simulation used in this paper was built using Trick.

### C. Matlab

Matlab is a matrix based computer language developed by the Mathworks company. It is interpreted, meaning there is no separate compile step before run. The user can type commands in a terminal window and have them executed immediately. What makes it particularly powerful for this application is it's extensibility via toolboxes, either commercial, free, or user generated, as well as powerful graphing routines that make visualization and analysis of complex data much easier to code than building up from a traditional programming language such as C or C++. The Particle Swarm Optimization toolbox used as the core optimization package in this paper is a modified freely available Matlab toolbox.

## II. Particle Swarm Optimization

### A. Particle Swarm Optimization Basics

Particle Swarm Optimization (PSO) is an optimization technique that has had great success in solving some difficult problems, such as cost scheduling, printed circuit board (PCB) routing, air traffic control, trajectory generation in uncertain environments, and more. [2]

It is a population based stochastic algorithm and as such can be classed in with a group of algorithms called Evolutionary Computation (EC). It can also be classed alternatively as a member of Direct Search methods (DS). It is highly resistant to falling into local minima, a problem that plagues many traditional optimization methods. Also, it requires no ballpark first guess, again unlike traditional approaches. It is algorithmically compact making it fairly straightforward to program, a trait that is not shared with its conceptual cousin Genetic Algorithms with which it does share performance results.

PSO is a nonlinear optimizer. Discontinuities in the solution space require no

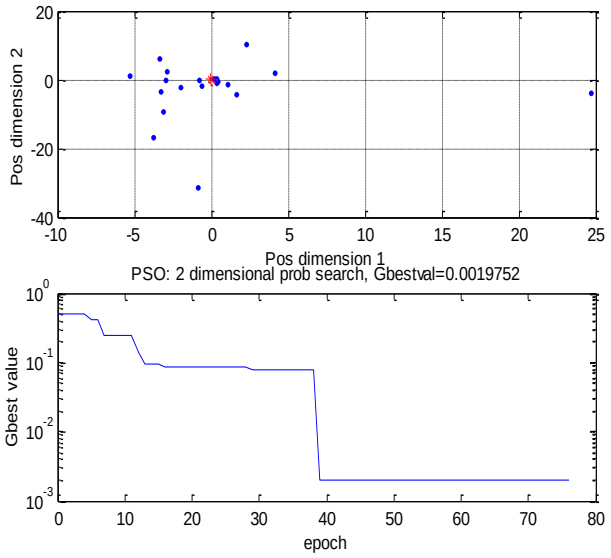


**Figure 2. Particle Swarm Optimization.** *Simple 2 dimensional Schaffer F6 minimization example.*

special treatment, other than what one requires of the specific problem under study.

PSO was initially inspired by efforts to model animal behavior dynamics, bird flocking, herding, and fish schooling. Plots of the dynamics often impress observers with "lifelike" behavior.

Figure 2 shows a simple PSO application on a 2 dimensional problem. The top part of the figure shows individual particle positions in 2D space at a single instant in time. The [0,0] point (red) is the current epoch/iteration's best position. Each particle is evaluated for cost with the current best particle acting as an attractor for other particles. Over time (bottom figure) the cost decreases. It may help to think of a 'particle' simply as a candidate solution.

The optimization problem of tuning control parameters to behave in conformation with hardware test data is cast as a weighted multi-objective minimization sum problem. This can be viewed as a single objective optimization in the grand sense. That is, there is a cost associated for our simulation model not matching with the flight test results. The lower the cost, the better the match achieved between the simulation and real world.

**B. The 'particle' in PSO**

The following set of equations illustrates the reason for the term 'particle' in PSO. There is a velocity equation and a position equation for each particle (candidate solution). The position equation is simply "current position equals past position + current velocity", analogous to physical dynamics, hence the 'particle' term. The velocity term is a bit more complicated. The first term on the right hand side is an expression that objects in motion tend to stay in motion. This is then multiplied by an inertia term. Often, but not always, the inertia term is a linearly decreasing function meant to limit exploration more as time goes on. The other two terms express a balance between a current particle's personal best position versus the entire swarm's global best position, with a slight random 'wiggle' factor.

Another way of thinking is that the first term encourages exploration of new territory in the error space while the other two terms encourage exploitation of already searched areas. [3] As time goes on, the particles concentrate on fine tuning already explored areas. So the velocity term is the driver for the PSO algorithm and continuously forces the particles to search the error space efficiently.

For each particle at iteration $k$:

$$v_k = (w \cdot v_{k-1}) + r_1 b_1 (p - x_{k-1}) + r_2 b_2 (g - x_{k-1}) \quad (1)$$

$$x_k = x_{k-1} + v_k \quad (2)$$

where:

$v_k$ = velocity at iteration $k$
$x_k$ = position at iteration $k$
$p$ = personal best position
$g$ = global best position
$w$ = inertia (can be function or scalar)
$b$ = attraction/balance coefficients
$r$ = random values over [0,1]

The Trelea 'set 1' convention is used, where $[b_1, b_2]$ are set as a constants to [0.6, 1.7] and w is constant = 1. There are other conventions and variations of expressing the balance between global and personal bests as well as varying degrees of exploration vs. exploitation of the error space. [4]

## III. Tool Architecture

**A. Morpheus Simulation details**

The simulation used for in-the-loop development of the Morpheus vehicle is Trick based which means data driven. Once the simulation has been compiled for the end user's particular platform then changes to a single 'input' file are all that is required to create different scenarios and analysis runs. However, being a high fidelity simulation there are hundreds if not thousands of parameters that can be data driven so some expertise in the vehicle domain is required, either by running canned routines prepared by developers or by diving in and modifying by the end user.

For the purposes of this paper, the internal details of the simulation are secondary to the overall architecture that controls optimization.

The concept is simple. A central manager sets up runs and controls the optimization progress. It spawns individual simulations in batch on one or many remote computers that may have varying levels of workload capability. As simulations are finished the central manager collates the data and determines optimization convergence. This is an iterative process that continues until stopping criterion is met. So an uneven cluster with a single processor computer, a 10 node computer, and a typical local workstation could all be used out of the box with this architecture to perform a single optimization search. The

goal is to make the optimization process as simple as possible for the end user, who it is assumed may be more of a domain expert on the problem that needs optimized rather than optimization theory itself.

Alternatively, the end user does not need to be a domain expert in either the simulation or optimization in order to run the tools. Simply providing a list of parameters in need of optimization and some reference data (that can be in-simulation or external) is all that is needed in order to complete optimization tasks.


**Figure 3. High level view of end to end code architecture.** *The central manager spawns parallel simulations on remote hosts.*

**B. Optimizing the Simulation with PSO**

The problem is cast as a weighted multi-objective minimization sum problem which conceptually can be viewed as a single objective optimization in the grand sense. That is, there is a cost associated for our simulation model not matching with the reference data. The lower the cost, the better the match between the simulation and reference (presumably real world data). Of course that statement hides the more complicated detail under the hood.

Multi-objective optimization (MOP) is the optimization of parameters to possibly competing criteria.[5] For example if you wanted to optimize a hypothetical science mission on the moon you would want the least amount of fuel used for the most amount of time spent. Time spent and fuel used would be competing goals. That is, maximizing the time spent would increase fuel usage and vice versa.

Mathematically, a MOP problem can be stated as:

$$\min_{x \in C} F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} \qquad \ldots(MOP) \tag{3}$$

Where $x$ can be a scalar but is most likely a vector of parameters that need to be tuned in order to optimize each sub function $f_n(x)$, and $n$ is $\geq 2$. To express the constraints and variable bounds as $C$:

$$C = \{x : h(x) = 0,\ g(x) \leq 0,\ a \leq x \leq b\} \tag{4}$$

But when it comes down to practicalities, most people just take a weighted sum of the multi-objectives and turn it into a single objective function:

$$\sum_{i=1}^{n} \alpha_i f_i(x), \qquad \alpha_i > 0,\ i = 1, 2, \ldots, n. \tag{5}$$

The weighted sum approach was used for this project. The cost function with the most weight is a mean squared error based function that compares simulation time curves with user supplied reference data. The two sets of data (simulation and reference) need not be the same size or step as long as there is some overlap in the times for comparison. The Matlab general manager will automatically take care of any data re-sizing and/or interpolation in order to compare curves on an equal basis.

At the end of each iteration, each particle's cost is evaluated, and compared with the previous global best. If the current cost is better, the global best is updated, otherwise it is left alone. The optimization encourages a close fit to the reference data without getting bogged down in local minima traps.

## IV. Gimbal Actuator Case Study

### A. Case Study Setup

On the Morpheus vehicle there are 2 electromechanical actuators (EMA A,B) that provide engine gimbaling or Thrust Vector Control (TVC) of the Main Engine (ME). Additionally there is a third actuator used for controlling the ME throttle valve.

Comparing the response of the actuators to commanded position between simulation and hardware revealed some discrepancies. In the high frequency response the simulation did not track controller overshoot very well. Also, there appeared to be a position bias on both EMA's.

In order to get the simulation to match the hardware response, optimization of the simulation parameters responsible for controlling the actuator was performed. The vehicle's gimbals were put through a 'cross' maneuver and measurements of position taken at many intervals. This became the reference data.

A list of the simulation parameters that controlled the actuator was obtained. These parameters included R, Kt, Kb, Jm, Ng, and EMA_MAX_VOLTAGE. They are all physics based model parameters and for the purposes of this tool and the goal of not needing domain expertise we will ignore their meanings. Suffice it to say that the simulation code was based on physical modeling principals and the idea was to manipulate some of the constants to achieve better fits with reality.

The list of parameters is only important in noting its size. There are six parameters of interest. By tuning those parameters it was hoped that EMA positions would identically match to some arbitrary precision. So, a 6 dimension optimization problem for each EMA A,B was cast.

In a typical optimization problem the first guess of these parameters is extremely important as is ensuring adherence to boundary conditions. Due to the complexity and size of many of the high fidelity simulations, they are by nature brittle to boundary violations. A simple 10% value change might unexpectedly cause the simulation to have a segmentation fault, a catastrophic crash from which useful data might not be gleaned. This makes using normal parameter searches difficult, though not impossible. Using PSO we can ignore the issue. If a particle (remember particle = potential solution) causes a simulation crash, that simply means it is a high cost solution. The equations treat this discontinuity as a place of non-interest.

Setting up the problem from an end-user point of view is very simple. Provide a list of parameters to optimize, a set of reference data, and a goal (perfect match is default but is up to user's decision). These lists and parameters are specified in a structure that is passed around to the functions that need it. Essentially only one file needs to be user edited in order to complete an optimization run.

Because each individual run of the simulation took between 30 and 50 seconds, and PSO runs many simulations concurrently and iteratively, the main drawback to this method is computation time. Therefore the code architecture was designed to take advantage of parallelism to cut down runtime. The code can run on as many remote computers as the user has access to. Also, the user can specify a maximum number of spawns per computer to help with load balancing, useful when sharing resources.

The Gimbal Actuator Case Study took just over one day to run on a 12 node remote computer. 24 particles were used, so each iteration consisted of two passes of 12 simulations each running in parallel. An additional single pass at the end of the iteration was used to calculate the global best, purely for graphing purposes.

### B. Case Study Results

The results were unexpected. In short, there was a bias in EMA B that it was impossible to tune out with the simulation model. EMA A fared better and achieved a nice match between the reference data and simulation. Several runs with different parameters were attempted but no amount of optimization could remove the position offset from EMA B. Figure 4 shows both EMA A and B at the end of one of the optimization runs.

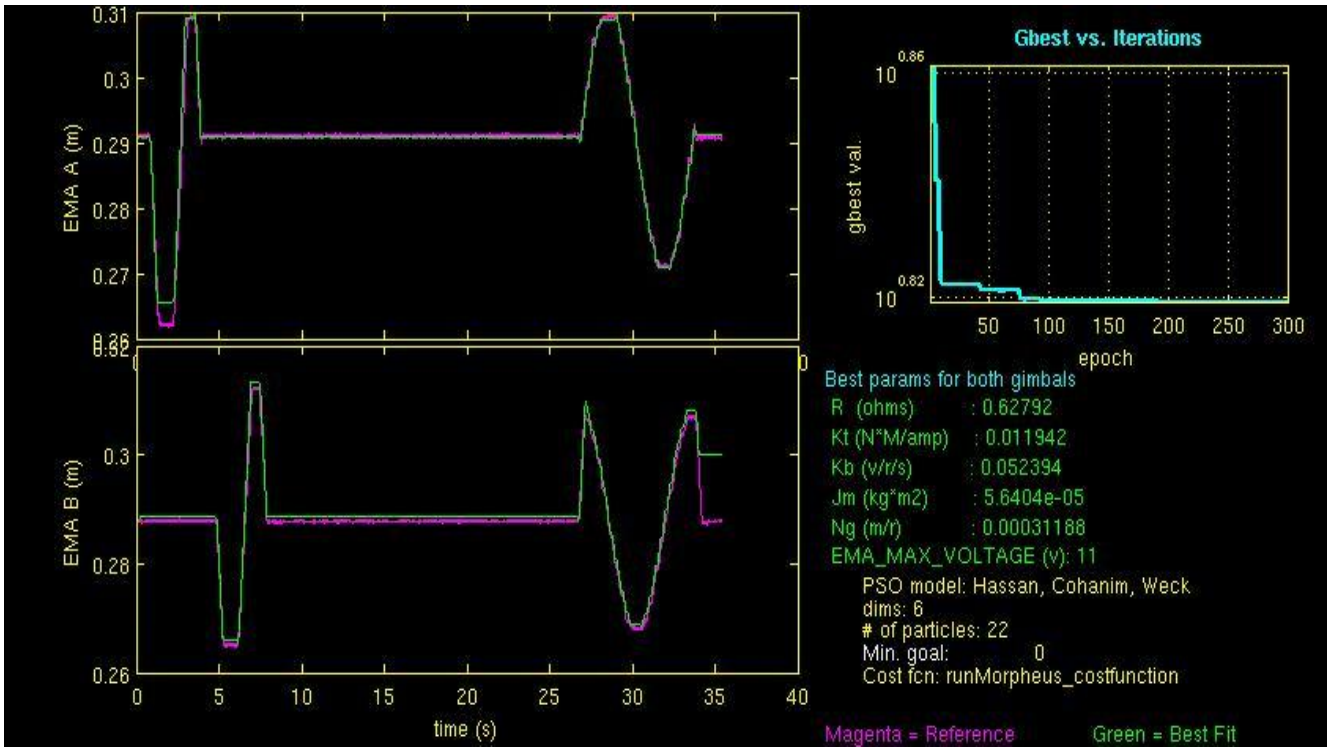


**Figure 4. PSO on Morpheus.** *An optimization for gimbal actuators A, B showing run details. End user can track progress visually.*

The graph shows a very good match for EMA A in the region of interest and also clearly shows the offset in EMA B. On the right hand side of the graph the cost as a function of iteration/epoch is shown as well as the best values at that moment for the parameters we wish to tune. Reference data is in purple, simulation data is shown in green. The next graph shows an expanded view of EMA B only, to focus in on the bias issue.

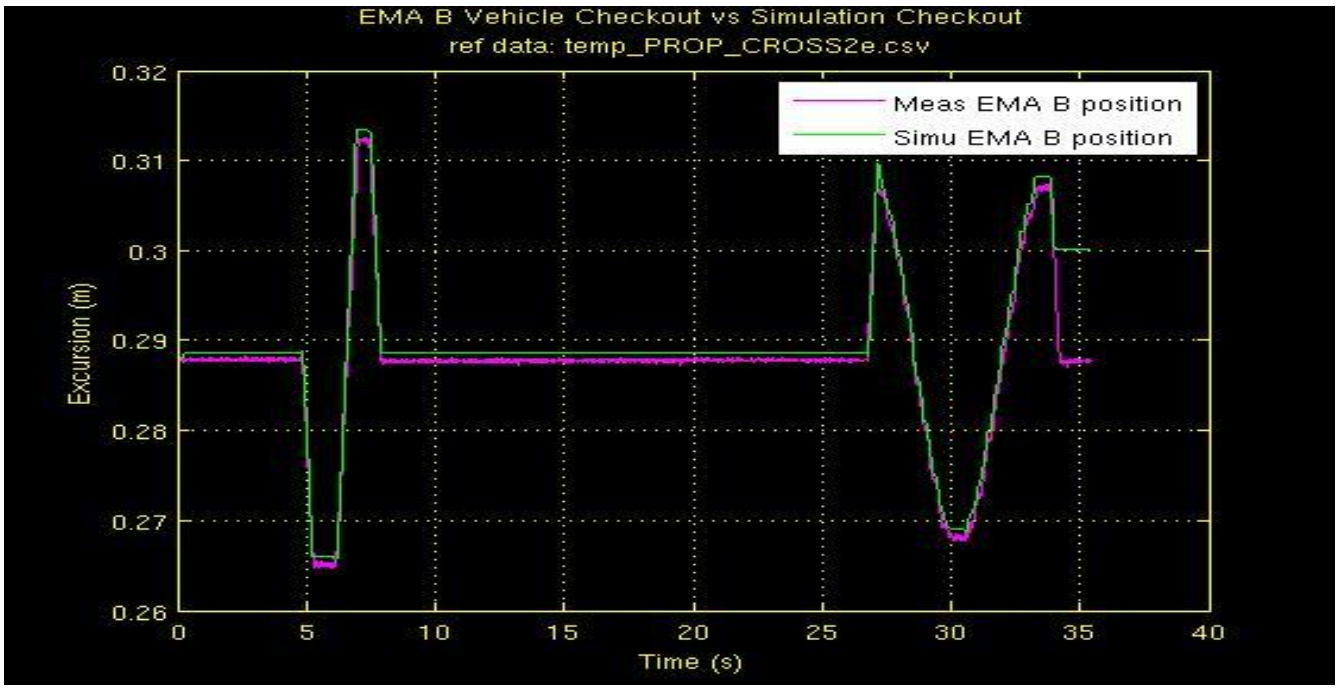


**Figure 5. Bias on EMA B.** *Small but significant position offset between simulation and telemetry data existed even after optimization.*

Why is it important? Visually the curves look like a "pretty good" match, the trends and even some high frequency content are tracked well. The optimizer certainly drove down a lot of error between the reference and simulation, it appears to be working well. Multiple runs and pours over the dynamic equations convinced us that this was no local minima but a true global minimum.

Why isn't this good enough? The importance comes down to what and where the gimbals are. The entire Morpheus vehicle can be thought of as an inverted pendulum with the gimbals at the fulcrum point. Any small deviation at the fulcrum cascades into huge deviations at the ends of the pendulum, or in this case the direction of the thrust vector. Tolerances are tight. Our simulation model simply did not have the control authority to match the reference data, no matter what numbers were applied. That points to the model being wrong on a basic level somewhere.

The simulation models were based in part on manufacturer provided specification sheets. These are numbers that are coded into not the simulation side of software, but the true flight software that controls the vehicle. The simulation ties into the flight software during runs for fidelity.

Given the results of the optimization run, the flight software specialists were able to make physical measurements of the gear linkages installed on Morpheus. What they found was not a problem with the simulation software or parameter choices but a problem within the flight software caused by incorrect information on the manufacturer's specification sheets. Their hardware gear linkage measurement did not match the manufacturer's stated measurement. Calls to the manufacturer confirmed that the specification sheet was wrong, the number in the flight software fixed, and the bias on EMA B disappeared.

## V. Conclusion

None of the individual pieces of this work are particularly ground breaking. However, when put together and working in a cooperative fashion they become something new and powerful. A Matlab based central manager sets up optimization problems on remote computers and runs many simulation spawns in parallel and iteratively. The remote computer carries the bulk of processor work as the Trick based simulation resides there. Trick, being inherently data driven, is tailor made to be leveraged by other programs. At the core of all is a robust non-linear Particle Swarm Optimization algorithm that is forgiving of discontinuities, requires no first guesses, and conforms to the problem rather than vice versa. When combined these pieces are extremely useful to aid in analysis, design, validation & verification, and even mission planning.

In the case study presented here, optimization initially did not work as expected. However this discovery directly led to the flight software team discovering a previously hidden issue in their software, one that would have been very difficult to diagnose without the clues the optimization provided. The code architecture presented here is useful not only on the simulation side of development and analysis but on the hardware and flight side as well, as part of the iterative design and development lifecycle.

Future work continues to refine the code for user friendliness to achieve a true 'easy button' approach to optimization. This is applied to other NASA projects besides Morpheus but for Morpheus itself the optimization continues to be useful as new software revisions and hardware changes manifest. This design allows the team to quickly re-tune, develop black box models for new features, and validate models with a much quicker turnaround and with less required hands on than previous approaches.

## References


[1]Crain, T., Nguyen, L., Ward, L., "Project M – Flight Dynamics Team Simulation Databook Document", 2010, Flt Dyn-M-2010-014

[2]Birge, B., "PSOt - A Particle Swarm Optimization Toolbox for Use With Matlab" *IEEE 2003 Swarm Intelligence Symposium*, Indianapolis, IN, USA, 2013, pp. 182-186

[3]Kennedy, J., Eberhart, R., *Swarm Intelligence*, Academic Press, San Francisco, CA, USA, ISBN 1-55860-595-9, 2001

[4]Trelea, I., "The particle swarm optimization algorithm: convergence analysis and parameter selection", *Information Processing Letters*, Vol. 85, no. 6, 2003, pp. 317-325

[5]Deb, K., *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, New York, ISBN 0 471 87339 X, 2001