2013-07-10

Variable Fidelity Optimization with Hardware-in-the-Loop for Flapping Flight

Michael Luke Duffield
Brigham Young University - Provo

Variable Fidelity Optimization with Hardware-in-the-Loop for Flapping Flight

M. Luke Duffield

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

July 2013

ABSTRACT

Variable Fidelity Optimization with Hardware-in-the-Loop for Flapping Flight

M. Luke Duffield
Department of Mechanical Engineering, BYU
Master of Science

Hardware-in-the-loop (HIL) modeling is a powerful way of modeling complicated systems. However, some hardware is expensive to use in terms of time or mechanical wear. In cases like these, optimizing using the hardware can be prohibitively expensive because of the number of calls to the hardware that are needed. Variable fidelity optimization can help overcome these problems. Variable fidelity optimization uses less expensive surrogates to optimize an expensive system while calling it fewer times. The surrogates are usually created from performing a design of experiments on the expensive model and fitting a surface to the results. However, some systems are too expensive to create a surrogate from. One such case is that of a flapping flight model. In this thesis, a technique for variable fidelity optimization of HIL has been created that optimizes a system while calling it as few times as possible. This technique is referred to as an intelligent DOE. This intelligent DOE was tested using simple models of various dimension. It was then used to find a flapping wing trajectory that maximizes lift. Through testing, the intelligent DOE was shown to be able to optimize expensive systems with fewer calls than traditional variable fidelity optimization would have needed. Savings as high as 97% were recorded. It was noted that as the number of design variables increased, the intelligent DOE became more effective by comparison because the number of calls needed by a traditional DOE based variable fidelity optimization increased faster than linearly, where the number of hardware calls for the intelligent increased linearly.

Keywords: variable fidelity optimization, hardware-in-the-loop modeling, flapping flight

ACKNOWLEDGMENTS

I would like to thank Dr. Chris Mattson, my graduate advisor, for his help and guidance throughout my graduate work. I am also grateful to my parents, Mike and Joan Duffield and my brother Matt for all of their encouragement and support. Also, I would like to thank my committee, Dr. Mark Colton and Dr. Scott Thomson, for their help with my thesis and various other things throughout my graduate experience. Finally, thank you to the Air Force Office of Scientific Research, who funded this research through award FA9550-10-1-0334.

TABLE OF CONTENTS

LIST OF FIGURES . . . . **vi**

NOMENCLATURE . . . . **vii**

Chapter 1 Introduction **1**

1.1 Literature Search 1

1.2 Approach 3

Chapter 2 The Intelligent DOE **5**

2.1 Assumptions 5

2.2 Algorithm 6

Chapter 3 Verification of the Intelligent DOE **11**

3.1 One Dimensional Example 11

3.1.1 IDOE Optimization 12

3.1.2 Fractional Factorial Best Point Optimization 13

3.1.3 Surrogate Response Surface Optimization 14

3.1.4 Performance Comparison 14

3.2 Two Dimensional Example 16

3.2.1 IDOE Optimization 17

3.2.2 Fractional Factorial DOE Optimizations 19

3.2.3 Performance Comparison 21

3.3 Twenty Dimensional Example 22

3.3.1 IDOE Optimization 23

3.3.2 Box-Behnken DOE Optimizations 26

3.3.3 Performance Comparison 28

3.4 One Dimensional Example with Different Physics Model Form 29

3.4.1 IDOE Optimization 29

3.4.2 Fractional Factorial Best Point Optimization 30

3.4.3 Surrogate Response Surface Optimization 30

3.4.4 Performance Comparison 31

3.5 Verification Discussion 32

Chapter 4 Application To Flapping Flight **35**

4.1 Flapping Flight Optimization 35

4.1.1 Physical Hardware 36

4.1.2 Optimization Setup 38

4.1.3 Optimization Steps 40

4.1.4 Optimization Results 43

Chapter 5 Limitations and Challenges **47**

5.1 Limitations Associated with Physics Based Model 47

5.2 Challenges Associated with Implementation of the IDOE 48
5.3 Other Limitations 49

Chapter 6 Conclusions and Recommendations 51
6.1 Conclusions 51
6.2 Future Work 52

REFERENCES 54

Appendix A Matlab Code From Flapping Flight Optimization 58
A.1 Main Function 58
A.2 Physics Model 62
A.3 Find Point with Matching Gradient 66

LIST OF FIGURES

1.1 Two Different,Yet Related Systems 4

2.1 Flowchart of the Intelligent DOE 7
2.2 Progression of the IDOE Through an Optimization 9

3.1 Flowchart of the Intelligent DOE 12
3.2 Performance of One Dimensional Sine Wave Tests 15
3.3 Performance of Two Dimensional Sine Wave Tests 21
3.4 Performance of Twenty Dimensional Sine Wave Tests 27
3.5 Progression of the IDOE Through Optimization of Models with Different Forms . . 33
3.6 Performance of One Dimensional Tests with Different Model Forms 34

4.1 The Flapping Mechanism 37
4.2 Flowchart of the Intelligent DOE 40
4.3 Performance of Flapping Flight Model Tests with Insufficient Accuracy in the Matched Gradient 45
4.4 Successful Intelligent DOE Optimization 46
4.5 Performance of Flapping Flight Model Tests 46

5.1 Progression of the IDOE Through an Optimization of a Stretched Model 50

NOMENCLATURE

α	Scaling factor between physics and hardware models
α_h	Artificial objective scaling between hardware and physics models used to create the hardware model
C_η	Term that controls the shape of $\eta(t)$
f	Wing flapping frequency
γ	Shift within physics model space
γ_h	Artificial shift within physics model space used to create the hardware model
H	Hardware model
K	Term that controls the shape of $\phi(t)$
$\mu_{h,b}$	Best objective value measured in the hardware model
$\mu_{p,i}$	Physics model objective value at $x^*_{p,i}$
η_0	Constant offset in η
$\eta(t)$	Wing pitch as a function of time
η_m	Maximum wing pitch
∇H	Gradient of the hardware model
∇P	Gradient of the physics model
N	Harmonic used in θ
P	Physics model
$\phi(t)$	Wing sweep as a function of time
ϕ_m	Maximum wing sweep
Φ_η	Phase shift in η
θ_0	Constant offset in theta
θ_m	Maximum deviation
Φ_θ	Phase shift in θ
$\theta(t)$	Wing deviation as a function of time
x_h	Coordinates of a point within the hardware model design space
x^*_h	Coordinates of the optimal point within the hardware model
x_p	Coordinates of a point within the physics model design space
x^*_p	Coordinates of the optimal point within the physics model
$x^*_{p,i}$	Initial optimal point in physics model space

CHAPTER 1. INTRODUCTION

1.1 Literature Search

Hardware-in-the-loop (HIL) modeling is a powerful way of modeling complicated systems. HIL means using physical hardware to represent a system rather than an analytical or physics-based model. This has some advantages and disadvantages when compared to analytical modeling. One advantage of HIL modeling is that it can avoid problems caused by inaccurate models. Complicated phenomena may be difficult to model accurately, but they can act on a hardware model similarly to how they would act on a real system. Another benefit of HIL modeling is manifest when accurate models cannot be obtained because of time or other constraints. In cases like these, a system can be modeled by a HIL model, which is treated as a black box [1] that returns performance outputs given inputs.

HIL modeling is used extensively in a few fields. HIL models are frequently used to test electrical components. Here, the rest of the system is usually simulated by computers so that the electrical component can be tested quickly with real inputs and outputs. This tactic is common in the testing of electronic control units (ECUs) for engines [1, 2]. This allows the ECU to be tested using many different inputs to verify that it works properly under a wide variety of circumstances [3]. Here, the hardware is considered to be inexpensive to call, which allows for this exhaustive testing. HIL is also used in automotive crash tests [4, 5]. Here, the actual hardware (the vehicle) is expensive, so large numbers of tests are not practical. VEhicle Hardware-In-the-Loop (VEHIL) [4, 5] is used to study a vehicle's responses to certain inputs leading up to a crash, frequently without actually crashing the vehicle [6]. VEHIL is a type of HIL that monitors the outputs of various components with simulated inputs.

Some researchers have used HIL in the study of flapping flight [7, 8]. Flapping flight is difficult to model and creating an accurate analytical model would be too complicated and time consuming [9] for the scope of the present research. However, HIL has some challenges when used in optimization. Sometimes, HIL models are too expensive to call exhaustively [10]. The HIL system used by George et al. has a large, complex design space and sampling the whole space with a central composite design of experiments could require over 32,000 calls [10]. While this test would take a long time to complete, it would also wear the system out. In cases like this, mechanical wear can contaminate the results of the test [11]. This makes the hardware too expensive to use in a normal optimization algorithm. Instead, a less expensive model is needed for optimization.

One potential solution to this problem is variable fidelity optimization (VFO), or variable complexity optimization [12]. Some models are too expensive to use in traditional optimization [13], and VFO can help optimize with less calls to the model. VFO attempts to find the solution to a model using many calls to a less expensive surrogate model and a smaller number of calls to the more expensive model [12, 14]. This minimizes the calls to the expensive model, saving cost. Since the surrogate is usually a response surface fit to data obtained from sampling the expensive model [10, 14, 15], it is not as accurate as the expensive model. Response surfaces are used because they can be obtained quickly, then used in optimization [16]. The surrogate is often a better representation of the expensive model in certain regions of interest around the solution [12] than it is in other areas. Accuracy can be improved by shrinking the region of interest or by increasing the number of points sampled [12]. Prior knowledge of the model can aid in choosing a region of interest, thus reducing the total number of calls needed [12]. Haftka et al. demonstrated this approach for a structural optimization problem [12]. The objective function in this example was very expensive and the optimization needed to call it as few times as possible. George et al. used a similar method to optimize the wing trajectory of a flapping wing [10]. The expensive model was a HIL model of a flapping wing. A design of experiments (DOE) was used to create a surrogate and determine a region of interest, then additional optimization was done in the chosen region of interest. This approach is effective because it removes areas with poor objective values [12]. While this optimization was much less expensive than traditional design of experiment-based optimization using only the hardware, it still required over 2000 calls to the expensive model [10]. In some cases, such as this one, a surrogate developed experimentally is still considered too expensive [17].

In this research, a physics-based model is used as a surrogate. Since it is based on physical laws rather than experimental data, this surrogate is created without any calls to the expensive model. Many analytical, physics-based models have been developed of flapping flight [11, 18–23]. Analytical models almost never match the system being modeled exactly and since these models were not developed to model the specific system being optimized, there is additional error. However, a physics-based model is acted upon by many of the same physical phenomena acting on the expensive hardware model, so a physics model can predict trends similar to those seen in the hardware model.

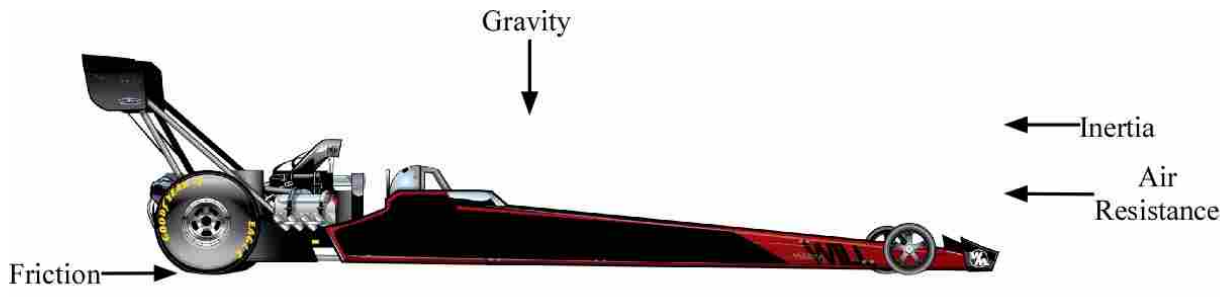
1.2 Approach

In this section, an approach to variable fidelity, hardware-in-the-loop optimization that minimizes calls to the hardware is presented, which is referred to as the hardware model. The basis of this approach is an intelligent DOE. For the purposes of this paper, an *intelligent* DOE is one that is informed by physics. As such, the intent of this research is to create an intelligent DOE. An intelligent DOE chooses points to test that are predicted to have better objectives than the previous points. This requires something that can provide accurate information about the hardware without calling it. A physics-based surrogate model can fill this role. The physics-based model predicts optimal solutions and gives information about how different design variables affect the performance of the hardware. This information can be used to choose the next point or points for the DOE to test. This way, the DOE does not waste time and hardware calls testing points for which the objective is worse.

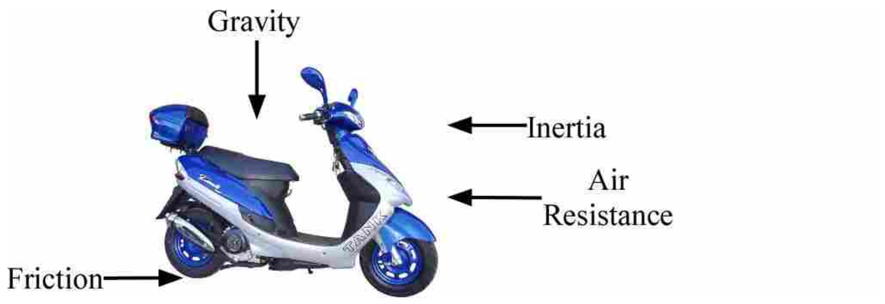
An unintelligent DOE-based optimization systematically or arbitrarily samples the space all around a point, searching for the best objective value. This information is then used to choose a starting point for the next iteration. The only points that are really desired, however, are those for which the objective improves. Points for which the objective values are worse are of little interest. Thus, the unintelligent DOE creates a surrogate with many points that will not be pursued as solutions. This is a problem for hardware in the loop modeling, where these points still require the expensive hardware to be called.

To avoid large numbers of hardware model calls, the surrogate model guiding the intelligent DOE is a physics-based model. This surrogate model does not require any hardware calls to create.

It can be a model made of a related system since building a model is often too time consuming and complex. Since the physics model is meant for another system, it will not match the hardware exactly. In fact, it may not match the hardware very well at all. Despite this, an appropriate model will still be able to aid the DOE since it includes many of the same physical phenomena that are at work on the hardware. Figure 1.1 illustrates the concept. Figure 1.1(a) is a top fuel dragster. Figure 1.1(b) is a scooter. Both have physical forces acting on them including gravity, air resistance, and inertia. In both cases, engine power, the coefficient of friction of the tires, and vehicle weight affect the performance of the system. All of the parameters discussed here are vastly different between the dragster and the scooter. The dragster may have an engine with over 1000 horsepower, while the scooter's engine is significantly less powerful. However, increasing engine power will make both vehicles faster. Likewise, decreasing the mass decreases inertia and the friction force in both cases. In the event that a scooter model does not exist, the dragster model could be used to inform choices about the scooter's design.



(a) Dragster



(b) Scooter

Figure 1.1: Two Different,Yet Related Systems

CHAPTER 2. THE INTELLIGENT DOE

In this chapter, the intelligent design of experiments (IDOE) is described. The intelligent DOE is the algorithm that performs variable fidelity optimization (VFO) on a hardware in the loop (HIL) system while minimizing calls to the hardware, or hardware model. It does this by using a physics-based model as a surrogate. This physics model makes predictions about the hardware, or hardware model, allowing the intelligent DOE to find the optimal hardware performance, or hardware solution, with a minimal number of calls to the hardware model.

2.1 Assumptions

The main assumption upon which this research is based is that the objective surface of a hardware model has a similar shape to that of a representative physics-based model. In other words, a feature at a certain point in the physics model surface is formed by simulating the effects of various physical phenomena and these phenomena create a similar feature in the hardware model surface. Due to inaccuracies inherent in modeling, these similar features may not lie at the same point within the two spaces and the features may not have the same size. It is also possible that one model will have features not found in the other. This can be the case when a physics-based model does not account for all phenomena acting on the system. However, provided that the physics-based model is sufficiently representative of the hardware system, these features can be small compared to those found in both models.

In order to take advantage of the similarities between a physics-based model and a hardware model, some mapping between the expensive hardware model and the inexpensive physics model must be made. This mapping allows the physics model to provide useful information about the hardware model by relating points within the physics model space to corresponding ones in the hardware model space. In order to create this mapping, some assumptions must be made about the differences between the models. Specifically, it is assumed that a point in the hardware model space, x_h, can be represented as a point in the physics model with some shift, γ. Also, the objective values may differ by some scaling factor α. The two models, then, can be related by the expression $H(x_h) = \alpha P(x_p - \gamma)$, where x_p is a point within the physics model space and x_h is a point within the hardware model design space. H represents the hardware model and P represents the physics model. Since x_p is an element within the design space and γ is defined within the same space, both are vectors of the same number of dimensions as the physics model has. There are other potential differences between the models, such as stretching, which will be discussed in Section 5.1; however, for many situations, the differences discussed here are sufficient to map the models sufficiently well for optimization purposes.

2.2 Algorithm

Here, the steps in the intelligent DOE are outlined. The optimization problem can be stated as

$$\underset{x}{min}\, H(x) \tag{2.1}$$

subject to:

$$\text{linear constraints} \leq 0 \tag{2.2}$$

$$\text{nonlinear constraints} \leq 0 \tag{2.3}$$

where:

$$H(x) \text{ is the hardware model.} \tag{2.4}$$

Using the assumptions discussed in Section 2.1 above, the next step to finding the hardware model solution, x_h^*, is to find values for α and γ for which $H\left(x_p^*\right) = \alpha P\left(x_{p,i}^* - \gamma\right)$, where $x_p^* = x_{p,i}^* - \gamma$, and $x_{p,i}^*$ is the initial optimal solution of the physics model. When these values have been found, $x_h^* = x_p^*$.

Steps 1 through 10, above, show the algorithm used to find values for α and γ and Figure 2.1 shows a flowchart of the entire method. First, the user finds the optimal solution, $x_{p,i}^*$, to the

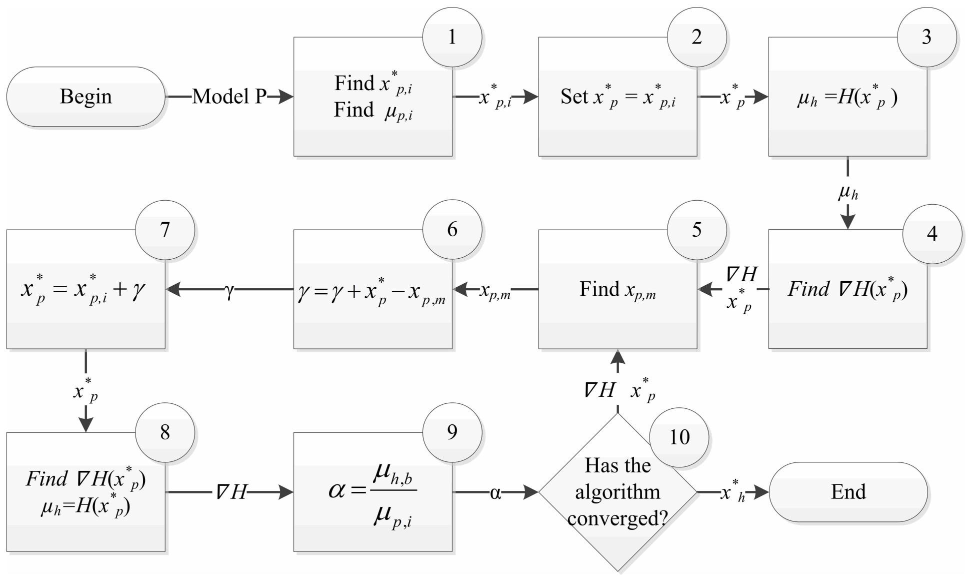


Figure 2.1: Flowchart of the Intelligent DOE

low cost physics model. This involves setting bounds on the space, choosing a starting point, and using a known optimization technique. The optimization problem is expressed by

$$\min_{x_p} P(x_p) \tag{2.5}$$

subject to:

$$\text{linear constraints} \leq 0 \tag{2.6}$$

$$\text{nonlinear constraints} \leq 0 \tag{2.7}$$

where:

$$P(x) \text{ is the physics model.} \tag{2.8}$$

In this research, gradient-based algorithms were used, however, genetic or other algorithms could be effective, depending on the nature of the model. $\mu_{p,i}$ is the objective value at $x^*_{p,i}$. Step 2 sets $x^*_p = x^*_{p,i}$. $x^*_{p,i}$ is the initial solution to the physics model and does not change throughout the algorithm, while x^*_p does change as the physics model is shifted. Step 3 is calling the hardware model, H, at the physics model solution, x^*_p, and setting the result as μ_h. Step 4 finds the gradient of the hardware model at x_p^*. Steps 5 through 10 are repeated until the algorithm converges. Once $\nabla H(x_p^*)$ has been found, step 5 finds the point, $x_{p,m}$, where the physics model gradient matches the gradient measured in the hardware. This step is an optimization problem that can be stated as

$$\min_x L(x) \tag{2.9}$$

subject to:

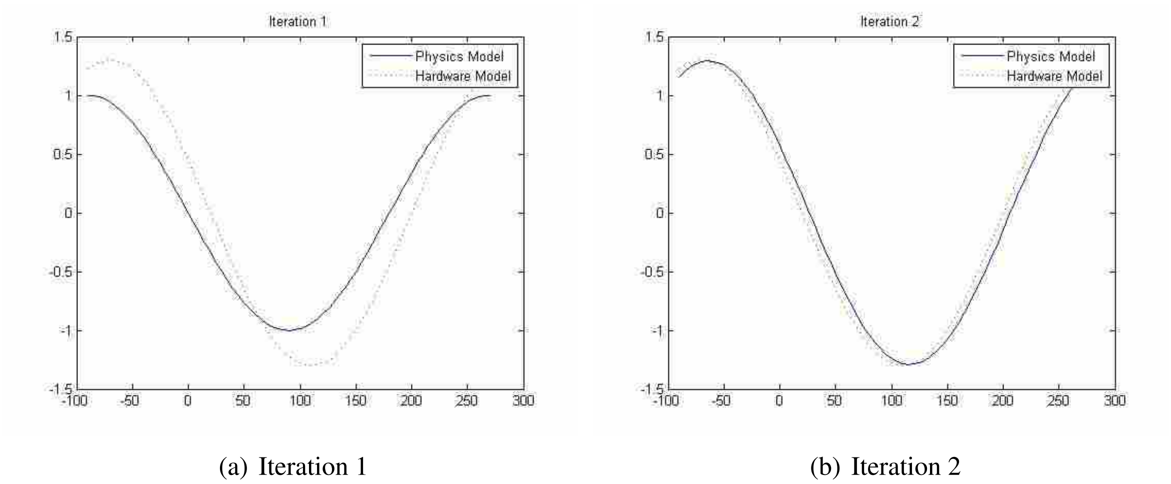
$$\text{linear constraints} \leq 0 \tag{2.10}$$

$$\text{nonlinear constraints} \leq 0. \tag{2.11}$$

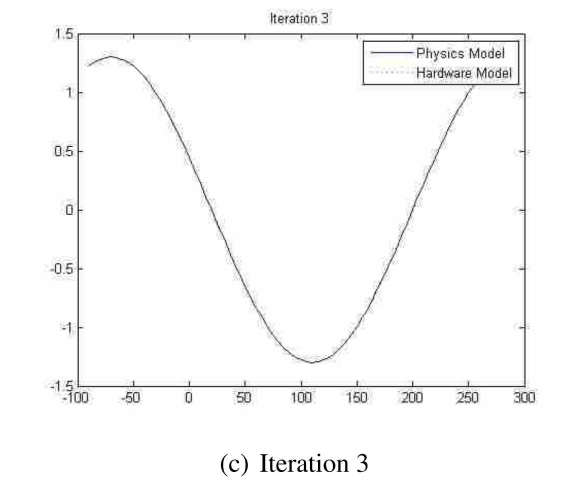
where:

$$L = \|\nabla H(x_p^*) - \nabla P(x_{p,m})\| \tag{2.12}$$

This is done with known optimization techniques. Here, gradient-based and genetic algorithms were used as needed by the problem. Step 6 finds the offset between the hardware and physics models, γ, by subtracting $x_{p,m}$ from x_p^*. This difference is added to the existing offset, which is initially zero, and the physics model is shifted by this amount. Next, step 7 defines the new x_p^* by shifting it by the offset so that it does not move relative to the physics model. μ_h and ∇H are found at this new x_p^* in step 8. Step 9 updates α by dividing the best value returned by the hardware, $\mu_{h,b}$, by $\mu_{p,i}$, the value of the physics model at $x_{p,i}^*$. This scale must be accurate in order for the physics gradients to be correct. Finally, 10 checks for convergence. This can be done using any appropriate criteria. A convergence criteria for variable fidelity optimization is presented in the literature [24], however, this uses a traditional response surface, which allows for many calls. This method can require a prohibitive number of function calls when calling the expensive physics model [24]. Two alternative convergence criteria requiring no additional hardware calls are presented here. One criterion is whether the gradient was zero. As is the case with some gradient-based algorithms [25], this can terminate the algorithm at local maxima, minima, or saddle points, so care should be taken to ensure that solutions reached with this criteria are optimal. In many cases, solutions to engineering problems are constrained, which means, among other things, that the gradients are non-zero [25]. For cases like this, convergence can be triggered when $\nabla P(x_p^*) = \nabla H(x_h^*)$. This criterion, in effect, is triggered when the hardware point that corresponds to the physics model solution, x_p^*, has been found. This assumes that the scale, α, and the offset, γ have been found correctly. Other convergence criteria could be used, depending on the nature of the problem being solved. Using these steps, the physics model shifts and scales until it matches the hardware model at the solution and $x_p^* = x_h^*$. As can be seen in the flowchart, each time steps 5-10 are iterated, the hardware gradient must be found. Thus, the number of hardware calls needed for the IDOE to converge is directly related to the number of iterations of the algorithm and it is desirable to find the solution with as few iterations as possible.



(a) Iteration 1 (b) Iteration 2



(c) Iteration 3

Figure 2.2: Progression of the IDOE Through an Optimization

Figure 2.2 shows plots of hypothetical hardware and physics models after each iteration of a one dimensional test. The objective of this problem is to minimize the value returned by the hardware. A similar example is discussed in more detail in Section 3.1. Here, the hardware model has been scaled by a factor of $\alpha = 1.3$ and shifted by $\gamma = 20^\circ$. Figure 2.2(a) shows the models in their initial state. Notice that it has larger negative value than $\mu_{p,i}$ which is caused by the fact that the hardware model is scaled larger than the physics one. In iteration 2, α and γ have been updated so that the physics model is nearly equal to the hardware model. By the third iteration, shown in Figure 2.2(c), the models are equal and the solution to the hardware model, x_h^* equals the known solution to the physics model, x_p^*.

CHAPTER 3. VERIFICATION OF THE INTELLIGENT DOE

After development, the intelligent DOE (IDOE) was verified on several problems of varying complexity. This allowed for testing of a variety of scenarios. Each example problem was optimized by the IDOE and two DOE-based VFO routines for comparison. The results are shown.

3.1 One Dimensional Example

Initially, the IDOE was tested with a one dimensional sine wave. Details of this example are also found in Section 2.2. The physics model was of the form $P(x_p) = -\sin(x_p)$ and the hardware was represented by $H(x_h) = -\alpha_h \sin(x_h - \gamma_h)$. The values of α_h and γ_h were varied throughout testing to simulate various scenarios. The objective was to find the minimum value of the hardware model. This optimization problem can be stated as

$$\underset{x_h}{min}\, H(x_h) \tag{3.1}$$

subject to:

$$-90^\circ \leq x_h \leq 270^\circ \tag{3.2}$$

where:

$$H(x_h) = -1.3\sin(x_h - 20^\circ). \tag{3.3}$$

After optimizing this problem with the IDOE, two DOE-based VFO routines were performed on it for comparison. Both used a fractional factorial (FF) DOE. FF designs use only a subset of the tests prescribed by a full factorial DOE in order to sample the space while minimizing the number of tests needed [26]. For these tests, $\alpha_h = 1.3$ and $\gamma_h = 20^\circ$.

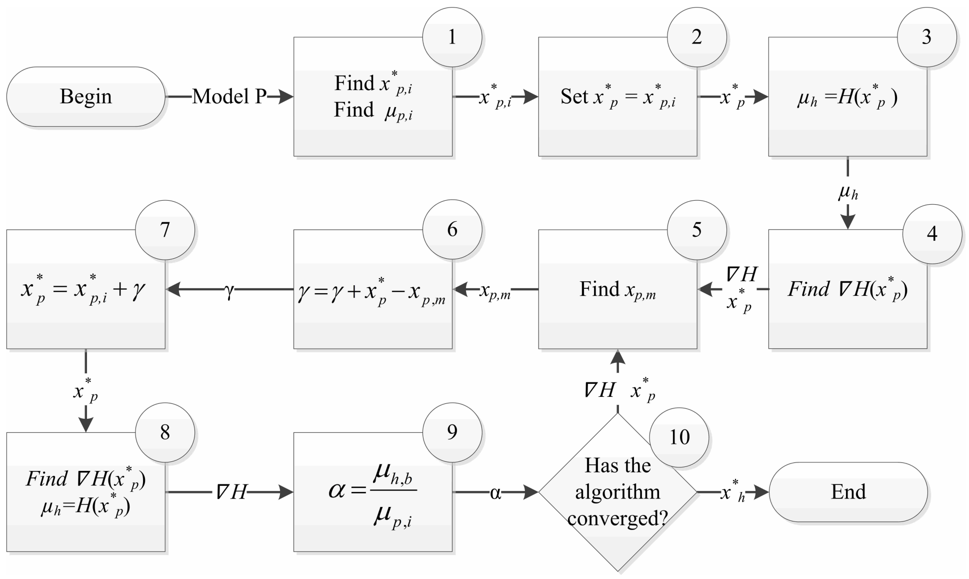


Figure 3.1: Flowchart of the Intelligent DOE

3.1.1 IDOE Optimization

The IDOE followed the steps outlined in Section 2.2. For convenience, the flowchart, Figure 2.1 is shown again here as Figure 3.1.

First, the starting point was set to be

$$x_p = 10^\circ. \tag{3.4}$$

This value was chosen arbitrarily. This value was used by the initial optimization, step 1 in Figure 3.1. The optimizer returned the point

$$x^*_{p,i} = 89.9983^\circ \tag{3.5}$$

where $\mu_{p,i}$ = -1.0000. Next, x^*_p was defined and set equal to $x^*_{p,i}$, as shown in Figure 3.1, step 2. During step 3, μ_h was found to be -1.2216 and in step 4, $\nabla H(x^*_p)$ was found to be -0.4446. Step 5 is to find the point in the physics model where the gradient equals $\nabla H(x^*_p)$. This point was found to be

$$x_{p,m} = 63.6046^\circ. \tag{3.6}$$

Using $x_{p,m}$ and x_p^*, the shift was defined to be

$$\gamma = 26.3937^\circ. \tag{3.7}$$

As is shown above, the actual shift between the two models is

$$\gamma = 20^\circ, \tag{3.8}$$

so the value found by the IDOE in equation 3.7 is substantially different from what it should be. The differences between these equations originate primarily from the fact that the artificial scaling has altered the gradient values. Next, the physics model and x_p^* were shifted within the design space by γ. The new value for x_p^* was 116.3920°. Moving on to step 8, the new μ_h was found to be -1.2919 and

$$\nabla H = 0.14481^\circ. \tag{3.9}$$

Using the best value of μ_h to date, -1.2919, α was found to be 1.2919 in step 9. Step 10, the convergence criteria, were not met, so one more iteration was needed. When the IDOE converged, it found a solution of $\mu_h = -1.3000$ where $x_p^* = 19.9553^\circ$ and $\alpha = 1.3000$. These values are very close to the artificial ones shown above, indicating that the solution is correct.

3.1.2 Fractional Factorial Best Point Optimization

During the first optimization, a DOE was run over the entire space with the center point being the middle of the range and the other tests being on the edges of the range. After this test, the point with the best objective value was chosen as the new center point and the range of the experiment was cut in half. This was repeated until a solution was converged on. This test was constructed to follow the method used by George [10], except that it used a fractional factorial design rather than a Box-Behnken design since Box-Behnken designs are not defined for problems with less than three variables.

3.1.3 Surrogate Response Surface Optimization

Next, an optimization was performed using a low fidelity surrogate response surface model. This was also performed by George and is a common technique in VFO [14, 15]. To form the surrogate, a FF design was performed, then a response surface was fit to the results. This surrogate was a third order polynomial of the form $f(x_h) = p_1 x_h^2 + p_2 x_h + p_3$, where p_1 through p_3 are coefficients found using least squares methods with the polyfit function in MATLAB. The surrogate solution, the minimum value, was found using gradient-based optimization. The optimization problem is shown below.

$$\underset{x_h}{min}\, f(x_h) \tag{3.10}$$

subject to:

$$-90^\circ \leq x_h \leq 270^\circ \tag{3.11}$$

where:

$$p_1 = \text{coefficient 1 from least squares fit} \tag{3.12}$$

$$p_2 = \text{coefficient 2 from least squares fit} \tag{3.13}$$

$$p_3 = \text{coefficient 3 from least squares fit} \tag{3.14}$$

$$f(x_h) = p_1 x_h^2 + p_2 x_h + p_3 \tag{3.15}$$

Then, the surrogate solution was set as the center point for the next iteration and the steps were repeated until a solution was found.

3.1.4 Performance Comparison

The actual optimum value of the hardware, $\mu_{h,A}^*$ was found for comparison using known optimization methods. The optimization problem solved here is

$$\underset{x_h}{min}\, H(x_h) \tag{3.16}$$

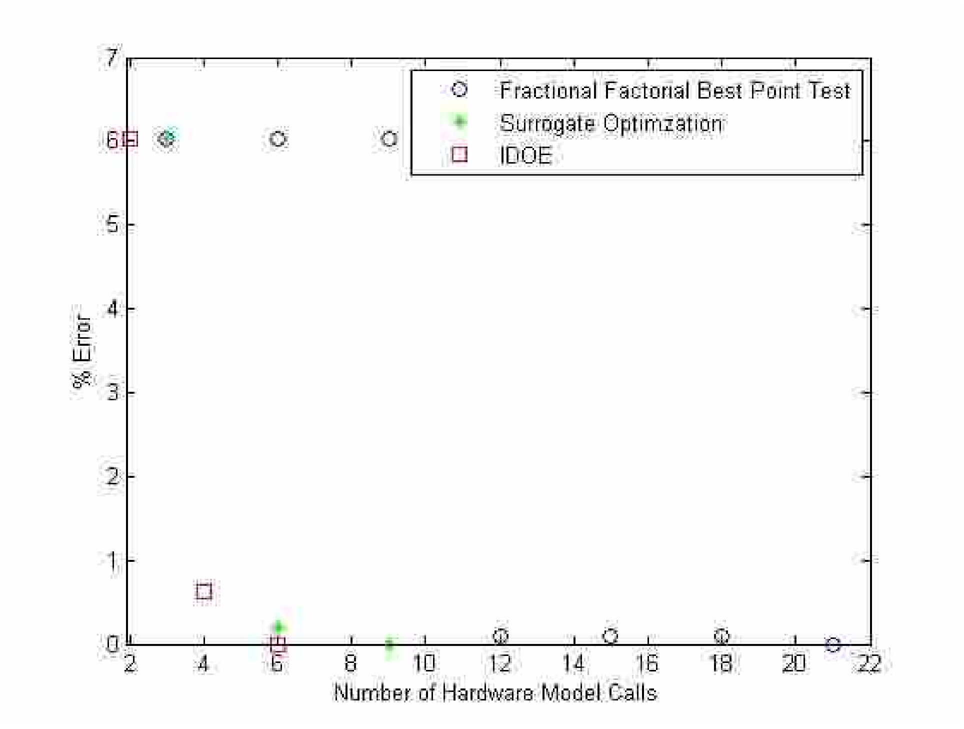


Figure 3.2: Performance of One Dimensional Sine Wave Tests

subject to:

$$-90^\circ \le x_h \le 270^\circ \tag{3.17}$$

where:

$$H(x_h) = -1.3\sin(x_h + 30^\circ). \tag{3.18}$$

$\mu^*_{h,A}$ could only be found in this way because the hardware was actually represented by a physics-based model. This was done outside of the other optimizations so that $\mu^*_{h,A}$ was unknown to them. The value of $\mu^*_{h,A}$ was used only to asses the accuracy of the solutions returned by the other optimization methods. For comparison, a DOE-based optimization was performed on the flapping flight model. The results are shown in Figure 3.2. The dependent axis, % Error, shown in the figure, was calculated for each data point using the equation

$$\% \text{ Error} = \left| \frac{\mu^*_{h,A} - \mu_{h,C}}{\mu^*_{h,A}} \right| * 100, \tag{3.19}$$

where $\mu_{h,C}$ is a solution value of one of the optimization routines being examined.

Figure 3.2 shows the performance of both DOE optimizations along with the performance of the IDOE. Each of the methods eventually found the solution, however, each method has different cost in terms of the number of hardware calls needed. The FF best point optimization described above requires the most hardware calls to find the solution, in this case, 21. The surrogate response surface optimization required nine calls to reach the solution and the IDOE took six calls. It is not surprising that the surrogate response surface optimization required less calls than the FF best point optimization because the surrogate provides a way to explore solutions that are not explicitly tested. This, combined with the gradient-based optimization, makes educated guesses about the location of the actual solution, rather than just choosing the best of the tested points. As Figure 3.2 shows, the IDOE found the solution in 71% fewer calls than the FF best point optimization and 33% fewer than the low fidelity surrogate optimization.

3.2 Two Dimensional Example

Next, the IDOE was tested with a two dimensional sine wave. The physics model was of the form

$$P(x_p) = -\sin(1.5x_{p1}) - \sin(1.5x_{p2}) \tag{3.20}$$

and the hardware was represented by

$$H(x_h) = -\alpha_h[\sin(1.5(x_{h1} - \gamma_{h1})) + \sin(1.5(x_{h2} - \gamma_{h2}))], \tag{3.21}$$

with $\alpha_h = 0.6$ and $\gamma_{h1} = \gamma_{h2} = -30^\circ$. The objective was to find the minimum value of the hardware model and the optimization problem can be formally stated as

$$\min_{x_h} H(x_h) \tag{3.22}$$

subject to:

$$-90^\circ \leq x_{h1} \leq 270^\circ \tag{3.23}$$

$$-90^\circ \leq x_{h2} \leq 270^\circ \tag{3.24}$$

where:

$$\gamma_h = [-30^\circ; -30^\circ] \tag{3.25}$$

$$H(x_h) = -0.6\sin(1.5(x_{h1} - \gamma_{h1})) - 0.6\sin(1.5(x_{h2} - \gamma_{h2})). \tag{3.26}$$

3.2.1 IDOE Optimization

The IDOE followed the steps outlined in Section 2.2. First, the starting point was set to be

$$[10^\circ; 10^\circ]. \tag{3.27}$$

These values were chosen arbitrarily. Step 1 in Figure 2.1 returned the point

$$x^*_{p,i} = [89.9983^\circ; 89.9983^\circ] \tag{3.28}$$

using the optimization problem

$$\underset{x_{p,i}}{min}\, P(x_{p,i}) \tag{3.29}$$

subject to:

$$-90^\circ \leq x_{p,i1} \leq 270^\circ \tag{3.30}$$

$$-90^\circ \leq x_{p,i2} \leq 270^\circ \tag{3.31}$$

where:

$$P(x_{p,i}) = -\sin(1.5(x_{p,i1})) - \sin(1.5(x_{p,i2}))]. \tag{3.32}$$

and the solution was $\mu_{p,i}$ = -2. Next, x_p^* was defined and set equal to $x_{p,i}^*$, as shown in Figure 2.1, step 2. During step 3, μ_h was found to be -1.0392 and in step 4, $\nabla H(x_p^*)$ was found to be

$$[-0.2533°; -0.2533°]. \tag{3.33}$$

Step 5 is to find the point in the physics model where the gradient equals $\nabla H(x_p^*)$. This point was found to be

$$x_{p,m} = [70.3101°; 70.3101°] \tag{3.34}$$

using the optimization problem

$$\underset{x_{p,m}}{min}\, L(x_{p,m}) \tag{3.35}$$

subject to:

$$-90° \leq x_{p,m1} \leq 270° \tag{3.36}$$

$$-90° \leq x_{p,m2} \leq 270° \tag{3.37}$$

where:

$$L = \|\nabla H(x_p^*) - \nabla P(x_{p,m})\|. \tag{3.38}$$

Using $x_{p,m}$ and x_p^*, the shift, γ, was defined to be

$$[19.6882°; 19.6882°]. \tag{3.39}$$

As is shown above, the actual shift between the two models is

$$[-30°; -30°], \tag{3.40}$$

so the values found by the IDOE in equation 3.39 had significant differences from what they should have been. The differences between these equations originated from the scaling differences between the models. When the hardware gradients were found in step 4, the different scaling changed the measured gradient. When $x_{p,m}$ was found, it led to a shift that was in the correct direction, but did not have the correct magnitude. Next, the physics model and x_p^* were shifted within the design space by γ. The new value for x_p^* was

$$[109.6865°;109.6865°]. \tag{3.41}$$

Since the shift was in the right direction, it still caused the physics model to be a more accurate approximation of the hardware. Since the shift was not of the right magnitude, though, more iterations were eventually needed before the IDOE could converge on a solution. Moving on to step 8, the new μ_h was found to be -1.1806 and

$$\nabla H = [-0.0555°;-0.0555°]. \tag{3.42}$$

Using the best value of μ_h to date, -1.1806, α was found to be 0.5903 in step 9. Step 10, the convergence criteria, did not signal convergence after this iteration, so the algorithm looped back to step 5. These steps were repeated two more times before the convergence criteria were tripped. When that happened, the value found for α was 0.6000 and γ equalled $[30.0875°;30.0875°]$. These are both very close to the actual artificial scale and shift, 0.6 and $[-30°;-30°]$, respectively. The values of γ could have been obtained more closely if the convergence criteria had been more strict. The solution returned by the IDOE was -1.2000, which is correct.

3.2.2 Fractional Factorial DOE Optimizations

After optimizing this problem with the intelligent DOE, two DOE-based optimization routines were performed on it for comparison. The fractional factorial best point optimization was similar to the one described in Section 3.1.2. The surrogate response surface optimization also used a fractional factorial DOE, but the response surface was of the form

$$f(x_h) = Bx_h + x_h^T C x_h + (x_h^T D x_h)(x_h^T D x_h), \tag{3.43}$$

where

$$x_h = \begin{bmatrix} x_{h1} \\ x_{h2} \end{bmatrix} \tag{3.44}$$

$$B = \begin{bmatrix} \beta_1 & \beta_2 \\ \beta_3 & \beta_4 \end{bmatrix} \tag{3.45}$$

$$C = \begin{bmatrix} \beta_5 & \beta_6 \\ \beta_7 & \beta_8 \end{bmatrix} \tag{3.46}$$

$$D = \begin{bmatrix} \beta_9 & \beta_{10} \\ \beta_{11} & \beta_{12} \end{bmatrix}. \tag{3.47}$$

This form was chosen because it fit the data from the design of experiments well. The surface was created using nonlinear regression with the nlinfit function in MATLAB. The minimum value of the response surface was found using the optimization problem

$$\underset{x_h}{min}\, f(x_h) \tag{3.48}$$

subject to:

$$-90^\circ \leq x_{h1} \leq 270^\circ \tag{3.49}$$

$$-90^\circ \leq x_{h2} \leq 270^\circ \tag{3.50}$$

where:

$$B = \beta \text{ values 1 through 4 from linear regression} \tag{3.51}$$

$$C = \beta \text{ values 5 through 8 from linear regression} \tag{3.52}$$

$$D = \beta \text{ values 9 through 12 from linear regression} \tag{3.53}$$

$$f(x_h) = B * x_h + x_h^T * C * x_h + (x_h^T * D * x_h) * (x_h^T * D * x_h). \tag{3.54}$$

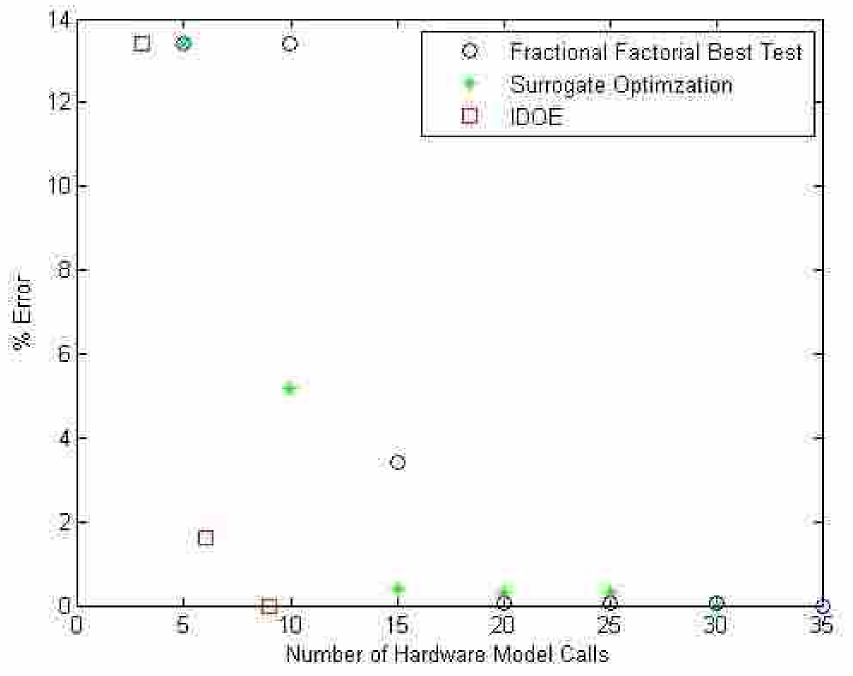


Figure 3.3: Performance of Two Dimensional Sine Wave Tests

3.2.3 Performance Comparison

The actual optimum value of the hardware, $\mu^*_{h,A}$ was found for comparison using known optimization methods. The optimization problem solved here is

$$\underset{x_h}{min}\, H(x_h) \tag{3.55}$$

subject to:

$$-90^\circ \leq x_{h1} \leq 270^\circ \tag{3.56}$$

$$-90^\circ \leq x_{h2} \leq 270^\circ \tag{3.57}$$

where:

$$\gamma_h = [-30^\circ; -30^\circ] \tag{3.58}$$

$$H(x_h) = -0.6\sin(1.5(x_{h1} - \gamma_{h1})) - 0.6\sin(1.5(x_{h2} - \gamma_{h2})). \tag{3.59}$$

$\mu^*_{h,A}$ could only be found in this way because the hardware was actually represented by a physics-based model. This was done outside of the other optimizations so that $\mu^*_{h,A}$ was unknown to them. The value of $\mu^*_{h,A}$ was used only to asses the accuracy of the solutions returned by the other optimization methods. For comparison, a DOE-based optimization was performed on the flapping flight model. The results are shown in Figure 3.3. The dependent axis, % Error, shown in the figure, was calculated for each data point using equation 3.19, where $\mu_{h,C}$ is a solution value of one of the optimization routines being examined. As Figure 3.3 shows, the FF best point optimization took 35 hardware tests to converge on the solution and the FF with surrogate optimization took 30 . The first IDOE point was far away from the solution. This point comes from the hardware being initially called at $x_h = x^*_{p,i}$. Since the γ values are relatively large, this point was far away from the hardware solution and the objective value was poor. The next iteration of the IDOE got closer to the actual solution after calling the hardware six times. It found the solution on the next iteration, requiring a total of nine calls. This represents a 74% improvement over the FF best point optimization and a 70% improvement over the FF with surrogate optimization.

3.3 Twenty Dimensional Example

Next, the IDOE was tested with a twenty dimensional sine wave. The physics model was of the form

$$P(x_p) = -\sin(x_{p1}) - \sin(x_{p2}) - ... - \sin(x_{p19}) - \sin(x_{p20}) \tag{3.60}$$

and the hardware was represented by

$$H(x_h) = -\alpha_h[\sin(x_{h1} - \gamma_{h1}) + \sin(x_{h2} - \gamma_{h2}) + ... + \sin(x_{h19} - \gamma_{h19}) + \sin(x_{h20} - \gamma_{h20})], \tag{3.61}$$

where $\alpha_h = 0.8$ and

$$\gamma_h = [13°; 8°; 14°; 10°; 15°; 9°; 10°; 17°; 12°; 10°; 13°; 10°; 14°; 11°; 8°; 16°; 15°; 9°; 10°; 10°]. \tag{3.62}$$

The objective was to find the minimum value of the hardware model using the formal problem

$$\min_{x_h} H(x_h) \tag{3.63}$$

subject to:

$$0^\circ \leq x_{h1}, x_{h2}, \cdots, x_{h20} \leq 180^\circ \tag{3.64}$$

where:

$$\gamma_h = [13^\circ; 8^\circ; 14^\circ; 10^\circ; 15^\circ; 9^\circ; 10^\circ; 17^\circ; 12^\circ; 10^\circ; 13^\circ; 10^\circ; 14^\circ; 11^\circ; 8^\circ; 16^\circ; 15^\circ; 9^\circ; 10^\circ; 10^\circ] \tag{3.65}$$

$$H(x_h) = -0.8[\sin(x_{h1} - \gamma_{h1}) + \sin(x_{h2} - \gamma_{h2}) + ... + \sin(x_{h19} - \gamma_{h19}) + \sin(x_{h20} - \gamma_{h20})]. \tag{3.66}$$

3.3.1 IDOE Optimization

The IDOE followed the steps outlined in Section 2.2. First, the starting point was set to be

$$[0^\circ; 1^\circ; 0^\circ; 90^\circ; 100^\circ; 0^\circ; 0^\circ; 0^\circ; 0^\circ; 0^\circ; 0^\circ; 45^\circ; 110^\circ; 5^\circ; 0^\circ; 0^\circ; 0^\circ; 0^\circ; 0^\circ; 0^\circ;]. \tag{3.67}$$

These values were chosen arbitrarily. Step 1 in Figure 2.1 returned the point

$$x^*_{p,i} = \begin{array}{c}[90^\circ; 90^\circ; 90^\circ; 90^\circ; 90.0001^\circ; 90^\circ; 90^\circ; 90^\circ; 90^\circ; 90^\circ; 90^\circ; \\ 90^\circ; 90.0001^\circ; 90.0002^\circ; 90^\circ; 90^\circ; 90^\circ; 90^\circ; 90^\circ; 90^\circ;]\end{array} \tag{3.68}$$

using the optimization problem

$$\min_{x_{p,i}} P(x_{p,i}) \tag{3.69}$$

subject to:

$$0^\circ \leq x_{p,i1}, x_{p,i2}, \cdots, x_{p,i20} \leq 270^\circ \tag{3.70}$$

where:

$$P(x_{p,i}) = -\sin(x_{p,i1}) - \sin(x_{p,i2}) - \cdots - \sin(x_{p,i19}) - \sin(x_{p,i20}) \tag{3.71}$$

where $\mu_{p,i} =$ -20. Next, x_p^* was defined and set equal to $x_{p,i}^*$, as shown in Figure 2.1, step 2. During step 3, μ_h was found to be -15.9856 and in step 4, $\nabla H(x_p^*)$ was found to be

$$[-0.0367°;-0.0226°;-0.0395°;-0.0280°;-0.0423°;-0.0253°;-0.0280°;\\ -0.0483°;-0.0339°;-0.0280°;-0.0367°;-0.0282°;-0.0396°;-0.0310°;\\ -0.0224°;-0.0454°;-0.0425°;-0.0251°;-0.0281°;-0.0281°]. \tag{3.72}$$

Step 5 is to find the point in the physics model where the gradient equals $\nabla H(x_p^*)$. This point was found to be

$$x_{p,m} = \begin{matrix}[79.63°;83.61°;78.83°;82.01°;78.08°;82.81°;82.00°;76.46°;\\ 80.43°;82.01°;79.63°;82.02°;78.84°;81.22°;83.60°;77.26°;\\ 78.05°;82.80°;82.01°;82.02°]\end{matrix} \tag{3.73}$$

using the problem statement

$$\underset{x_{p,m}}{min}\, L(x_{p,m}) \tag{3.74}$$

subject to:

$$0° \leq x_{p,m1}, x_{p,m2}, \cdots, x_{p,m20} \leq 270° \tag{3.75}$$

$$L = \|\nabla H(x_p^*) - \nabla P(x_{p,m})\|. \tag{3.76}$$

Using $x_{p,m}$ and x_p^*, the shift, γ, was defined to be

$$[10.36°;6.383°;11.16°;7.985°;11.91°;7.181°;7.990°;13.53°;9.566°;7.986°;\\ 10.36°;7.972°;11.15°;8.772°;6.393°;12.73°;11.94°;7.195°;7.984°;7.979°]. \tag{3.77}$$

As is shown above, the actual shift between the two models is

$$\gamma_h = [13°;8°;14°;10°;15°;9°;10°;17°;12°;10°;13°;10°;14°;11°;8°;16°;15°;9°;10°;10°]. \tag{3.78}$$

so the values found by the IDOE in equation 3.77 again had significant differences from what they should have been. The differences between these equations originated from the scaling differences between the models. When the hardware gradients were found in step 4, the different scaling changed the measured gradient. When $x_{p,m}$ was found, it led to a shift that was in the correct direction, but did not have the correct magnitude. Next, the physics model and x_p^* were shifted within the design space by γ. The new value for x_p^* was

$$\begin{gathered}[100.3°;96.38°;101.1°;97.98°;101.9°;97.18°;97.99°;103.5°;99.56°;97.98°;\\ 100.3°;97.97°;101.1°;98.77°;96.39°;102.7°;101.9°;97.19°;97.98°;97.97°].\end{gathered} \tag{3.79}$$

Since the shift was in the right direction, it still caused the physics model to be a more accurate approximation of the hardware. Since the shift was not of the right magnitude, though, more iterations were eventually needed before the IDOE could converge on a solution. Moving on to step 8, the new μ_h was found to be -15.9856 and

$$\begin{aligned}&[-0.0367°;-0.0226°;-0.0395°;-0.0280°;-0.0423°;-0.0253°;-0.0280°;\\ \nabla H = &-0.0483°;-0.0339°;-0.0280°;-0.0367°;-0.0282°;-0.0396°;-0.0310°;\\ &-0.0224°;-0.0454°;-0.0425°;-0.0251°;-0.0281°;-0.0281°]\end{aligned} \tag{3.80}$$

Using the best value of μ_h to date, -15.9856, α was found to be 0.7993 in step 9. Step 10, the convergence criteria, did not signal convergence after this iteration, so the algorithm looped back to step 5. These steps were repeated three more times before the convergence criteria were tripped. When that happened, the value found for α was 0.8000 and γ equalled

$$\begin{gathered}[12.99°;8.000°;14.00°;9.999°;14.95°;9.001°;10.00°;16.99°;12.00°;10.00°;\\ 13.00°;10.00°;13.99°;11.00°;8.002°;16.00°;15.00°;8.997°;10.00°;9.998°].\end{gathered} \tag{3.81}$$

These are both very close to the actual artificial scale and shift, 0.8 and

$$[13°;8°;14°;10°;15°;9°;10°;17°;12°;10°;13°;10°;14°;11°;8°;16°;15°;9°;10°;10°], \tag{3.82}$$

respectively. The values of γ could have been obtained more closely if the convergence criteria had been more strict. The solution returned by the IDOE was -16, which is correct.

3.3.2 Box-Behnken DOE Optimizations

After optimizing this problem with the intelligent DOE, two DOE-based optimization routines were performed on it for comparison. The first was a Box-Behnken best point optimization that was similar to the one described in Section 3.1.3, except that it used a Box-Behnken DOE rather than a fractional factorial DOE. The Box-Behnken is a very sparse design, although it is still able to explore the design space [27,28]. It was chosen for this reason. The second was a surrogate response surface optimization-based on data from Box-Behnken DOEs. The response surface was of the form $f(x_h) = Bx_h + x_h^T C x_h$, where

$$x_h = [x_{h1}; x_{h2}; ...; x_{h20}] \tag{3.83}$$

$$B = \begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_{20} \\ \beta_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \beta_{381} & \cdots & \cdots & \beta_{400} \end{bmatrix} \tag{3.84}$$

$$C = \begin{bmatrix} \beta_{401} & \beta_{402} & \cdots & \beta_{420} \\ \beta_{421} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \beta_{781} & \cdots & \cdots & \beta_{800} \end{bmatrix}. \tag{3.85}$$

This form was chosen because it fit the data from the design of experiments well. The surface was created using nonlinear regression with the nlinfit function in MATLAB. Optimizing the surface used the optimization problem statement

$$\underset{x_h}{min}\, f(x_h) \tag{3.86}$$

subject to:

$$0^\circ \leq x_{h1}, x_{h2}, \cdots, x_{h20} \leq 180^\circ \tag{3.87}$$

where:

$$B = \beta \text{ values 1 through 400 from linear regression} \tag{3.88}$$

$$C = \beta \text{ values 401 through 800 from linear regression} \tag{3.89}$$

$$f(x_h) = B * x_h + x_h^T * C * x_h. \tag{3.90}$$

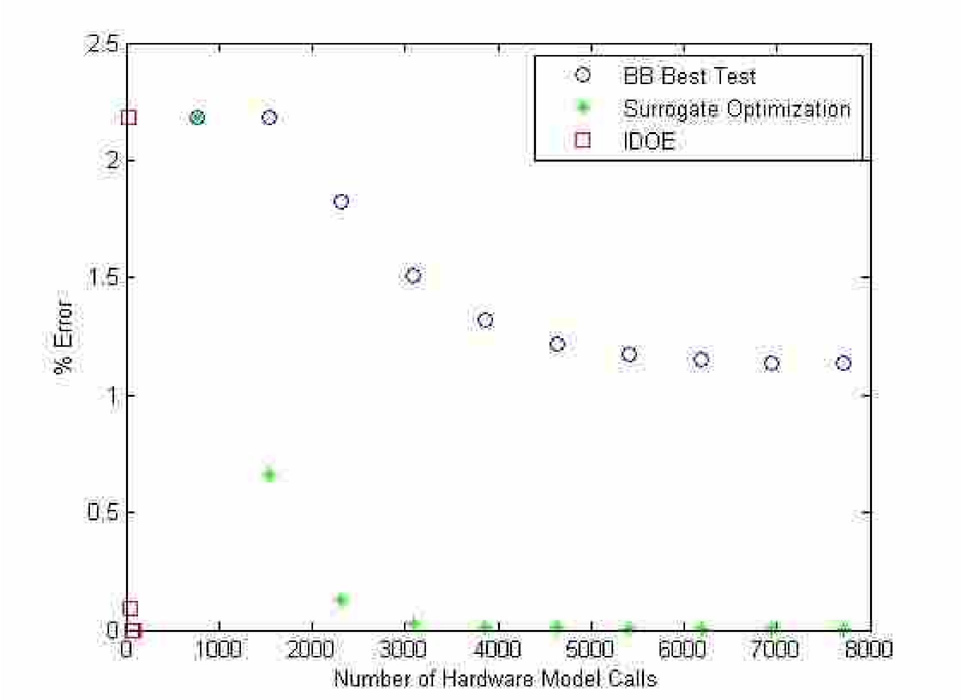


Figure 3.4: Performance of Twenty Dimensional Sine Wave Tests

3.3.3 Performance Comparison

The actual optimum value of the hardware, $\mu^*_{h,A}$ was found for comparison using known optimization methods. The optimization problem solved here is

$$\min_{x_h} H(x_h) \tag{3.91}$$

subject to:

$$0^\circ \leq x_{h1}, x_{h2}, \cdots, x_{h20} \leq 180^\circ \tag{3.92}$$

where:

$$\gamma_h = [13^\circ; 8^\circ; 14^\circ; 10^\circ; 15^\circ; 9^\circ; 10^\circ; 17^\circ; 12^\circ; 10^\circ; 13^\circ; 10^\circ; 14^\circ; 11^\circ; 8^\circ; 16^\circ; 15^\circ; 9^\circ; 10^\circ; 10^\circ] \tag{3.93}$$

$$H(x_h) = -0.8[\sin(x_{h1} - \gamma_{h1}) + \sin(x_{h2} - \gamma_{h2}) + ... + \sin(x_{h19} - \gamma_{h19}) + \sin(x_{h20} - \gamma_{h20})]. \tag{3.94}$$

$\mu^*_{h,A}$ could only be found in this way because the hardware was actually represented by a physics-based model. This was done outside of the other optimizations so that $\mu^*_{h,A}$ was unknown to them. The value of $\mu^*_{h,A}$ was used only to asses the accuracy of the solutions returned by the other optimization methods. For comparison, a DOE-based optimization was performed on the flapping flight model. The results are shown in Figure 3.4. The dependent axis, % Error, shown in the figure, was calculated for each data point using the equation 3.19, where $\mu_{h,C}$ is a solution value of one of the optimization routines being examined. In this example, the Box-Behnken best point test and the Box-Behnken with a surrogate response surface did not converge on the solution exactly, so convergence was said to have occurred when the solution was within 0.01% of the exact solution. Using this criteria, the Box-Behnken best point optimization did converge on a solution, but not the global minimum. This is because the bounds were shrunk around an inaccurate point and the exact solution was then outside of the region being explored. If this optimization had converged on the global minimum at the same rate it converged on the solution it did find, it would have required 6948 calls. The Box-Behnken with the surrogate response surface required 3860 hardware calls to converge. As in example 3.3, the IDOE started far away from the solution, then found it two iterations later. The IDOE required 84 calls, which represents a 98.7% reduction over the Box-Behnken best test optimization and a 97.8% reduction in cost compared to the response surface optimization.

3.4 One Dimensional Example with Different Physics Model Form

All of the previous tests used hardware models that were scaled and shifted from what the physics models. This test used hardware and physics models that were of different forms. The physics model is of the form $P(x_p) = -5\sin(x_p)$ and the hardware was represented by $H(x_h) = 2.5\left(\frac{x_h - 40^\circ}{50}\right)^2 - 10$. The objective is to find the minimum value of the hardware model. This optimization problem can be stated as

$$\min_{x_h} H(x_h) \tag{3.95}$$

subject to:

$$-90^\circ \leq x_h \leq 270^\circ \tag{3.96}$$

where:

$$H(x_h) = 2.5\left(\frac{x_h - 40^\circ}{50}\right)^2 - 10. \tag{3.97}$$

After optimizing this problem with the IDOE, two DOE-based VFO routines were performed on it for comparison. Both used the same fractional factorial DOE explained in Section 3.1.2.

3.4.1 IDOE Optimization

The IDOE followed the steps outlined in Section 2.2. Pictures showing the results from each iteration of the IDOE are shown in Figure 3.5.

3.4.2 Fractional Factorial Best Point Optimization

During the first optimization, a DOE was run over the entire space with the center point being the middle of the range and the other tests being on the edges of the range. After this test, the point with the best objective value was chosen as the new center point and the range of the experiment was cut in half. This was repeated until a solution was converged on. This test was similar to the one descried in Section 3.1.2.

3.4.3 Surrogate Response Surface Optimization

Next, an optimization was performed using a low fidelity surrogate response surface model. This test was the same as the one shown in 3.1.3. As before, the surrogate was a third order polynomial of the form $f(x_h) = p_1 x_h^2 + p_2 x_h + p_3$, where p_1 through p_3 are coefficients found using least squares methods with the polyfit function in MATLAB. The surrogate solution, the minimum value, was found using gradient-based optimization. The optimization problem is shown below.

$$\underset{x_h}{min}\, f(x_h) \tag{3.98}$$

subject to:

$$-90^\circ \leq x_h \leq 270^\circ \tag{3.99}$$

where:

$$p_1 = \text{coefficient 1 from least squares fit} \tag{3.100}$$

$$p_2 = \text{coefficient 2 from least squares fit} \tag{3.101}$$

$$p_3 = \text{coefficient 3 from least squares fit} \tag{3.102}$$

$$f(x_h) = p_1 x_h^2 + p_2 x_h + p_3 \tag{3.103}$$

Then, the surrogate solution was set as the center point for the next iteration and the steps were repeated until a solution was found.

3.4.4 Performance Comparison

The actual optimum value of the hardware, $\mu_{h,A}^*$ was found for comparison using known optimization methods. The optimization problem solved here is

$$\min_{x_h} H(x_h) \tag{3.104}$$

subject to:

$$-90^\circ \leq x_h \leq 270^\circ \tag{3.105}$$

where:

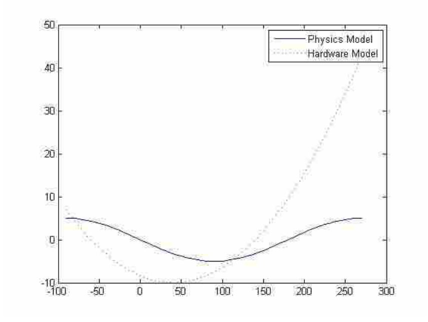
$$H(x_h) = 2.5\left(\frac{x_h - 40^\circ}{50}\right)^2 - 10. \tag{3.106}$$

$\mu_{h,A}^*$ could only be found in this way because the hardware was actually represented by a physics-based model. This was done outside of the other optimizations so that $\mu_{h,A}^*$ was unknown to them. The value of $\mu_{h,A}^*$ was used only to asses the accuracy of the solutions returned by the other optimization methods. For comparison, a DOE-based optimization was performed on the flapping flight model. The results are shown in Figure 3.6. The dependent axis, % Error, shown in the figure, was calculated for each data point using equation 3.19.

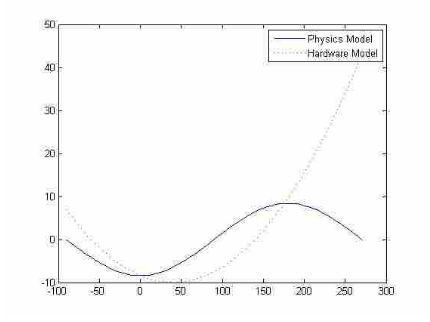
Figure 3.6 shows the performance of both DOE optimizations along with the performance of the IDOE. Each of the methods eventually found the solution, however, each method has different cost in terms of the number of hardware calls needed. The FF best point optimization described above requires the most hardware calls to find the solution, in this case, 30. The surrogate response surface optimization required 6 calls to reach the solution and the IDOE took 14 calls. It is not surprising that the surrogate response surface optimization required less calls than the FF best point optimization did because the surrogate provides a way to explore solutions that are not explicitly tested. This, combined with the gradient-based optimization, makes educated guesses about the location of the actual solution, rather than just choosing the best of the tested points. In this case, the response surface form used for the surrogate was of a very similar form to the hardware model. This is why it was able to optimize so quickly. As Figure 3.2 shows, the IDOE found the solution in 53% fewer calls than the FF best point optimization but 130% more calls to the low fidelity surrogate optimization.

3.5 Verification Discussion

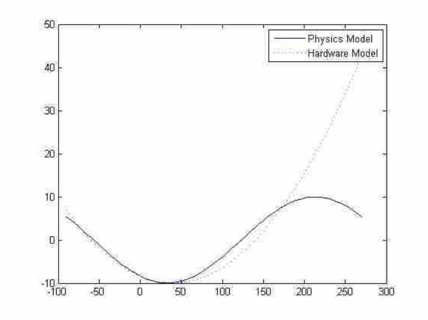
Based on the results of these tests, it is clear that, in most of the situations tested, the IDOE converges on a solution with fewer calls to the physical hardware than DOE-based optimizations do. It is interesting to note that as the number of design variables increases, the percent cost savings from the IDOE increases. This is due to the fact that the number of calls needed by a DOE increases faster than linearly as the number of variables increases. On the other hand, the number of calls needed by the IDOE is driven by the cost of taking gradients, which increases linearly with the number of variables. Also in these tests, the IDOE converges on the global solution more reliably than the DOE-based optimizations do. This is due to the fact that, in the DOE optimizations, the bounds of the region of interest are shrunk in between experiments in order to get a more accurate approximation of the surface. Sometimes, shrinking the bounds can make it so that the global solution is out of bounds and, therefore, out of the search area.



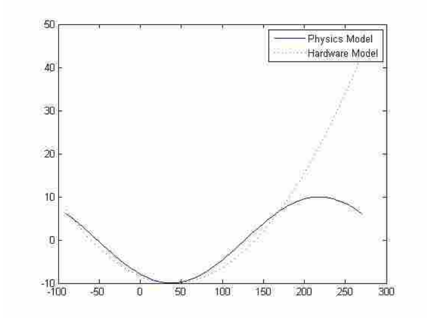
(a) Iteration 1



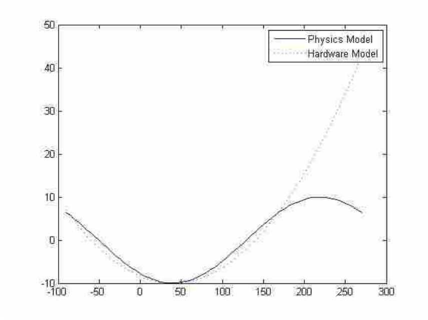
(b) Iteration 2



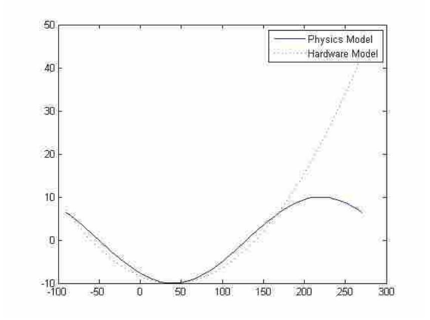
(c) Iteration 3



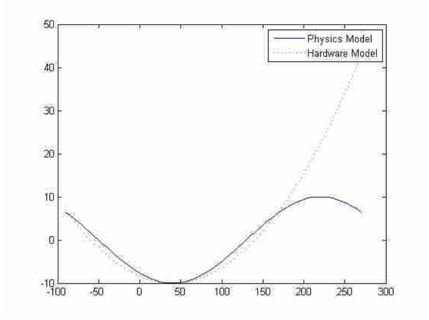
(d) Iteration 4



(e) Iteration 5



(f) Iteration 6



(g) Iteration 7

Figure 3.5: Progression of the IDOE Through Optimization of Models with Different Forms

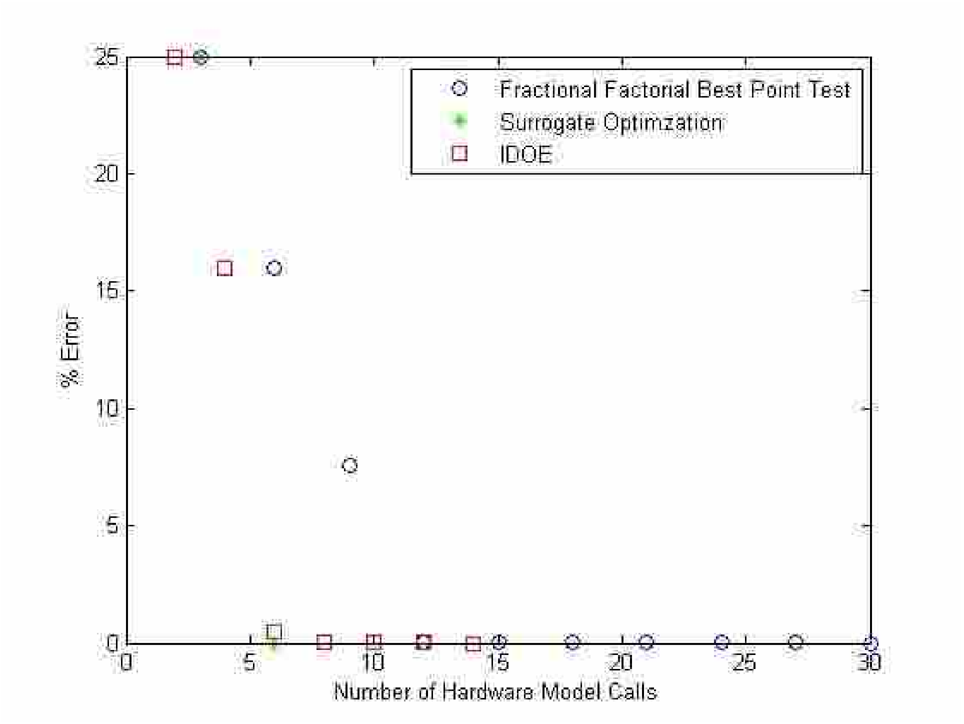


Figure 3.6: Performance of One Dimensional Tests with Different Model Forms

CHAPTER 4. APPLICATION TO FLAPPING FLIGHT

The intelligent DOE was developed to enable VFO on HIL systems where the hardware is too expensive for traditional DOE-based surrogate optimization. Chapter 3 shows that this IDOE is capable of solving simple problems with small or large numbers of design variables. This chapter shows the IDOE optimizing a more useful system, in this case, one representing a flapping wing. The objective of this optimization is to find a hovering wing trajectory that produces the maximum amount of lift.

4.1 Flapping Flight Optimization

Over the past century, flight has come to play an essential role in many civilian and military applications such as transportation, surveillance, and defense. The overwhelming majority of aircraft today use fixed wings or rotors to produce lift. These both have inherent advantages and disadvantages. Some potential applications, however, require flight characteristics that are not found in either of these traditional methods. For example, both fixed wing and rotor flight produce large amounts of sound [29]. Fixed wing aircraft cannot hover and are difficult to maneuver in tight spaces, such as indoors [30]. Vehicles using rotors have limited endurance [30] and are also not very maneuverable [31]. Also, both fixed wing and rotor flight lend themselves to large vehicles [31, 32].

Insects, which use flapping wings to produce lift, are small and very capable of hover and precise flight in confined areas [23, 31, 33]. For small vehicles that require quiet, precise flight in tight spaces, flapping wing flight shows potential [34]. Flapping flight is being researched for use with Micro Air Vehicles or MAVs [7,11,19–21,23,31,33,35–38]. These vehicles can carry sensors and relay information back to a controller. This makes MAVs well suited for applications such as surveillance and inspection of areas that are dangerous or difficult to access [39].

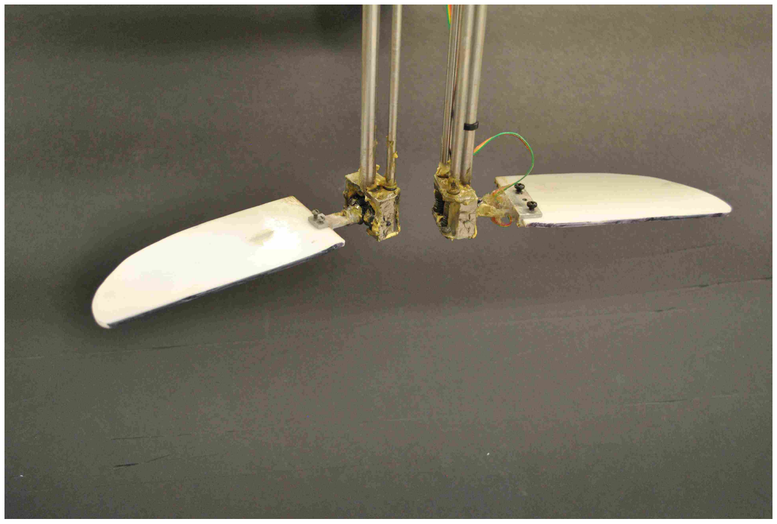
Flapping flight development has lagged behind that of fixed wing and rotor flight due to a number of challenges. Flapping flight relies on complex aerodynamics that are difficult to control [9]. For example, conventional aerodynamics can explain some aspects of flapping flight, but unsteady aerodynamics must be used to explain hovering of small insects [40, 41]. Because of these and other phenomena not accounted for in conventional aerodynamics, flapping wings can produce more lift than would be predicted by conventional aerodynamic analysis [42], sometimes by a factor of two or three [43]. Further, rotational forces on the insect are unstable in some cases [44, 45]. Finally, the performance of a flapping wing involves interdependent relationships between factors like wing planform and flapping trajectory [23]. Flapping flight models are difficult to develop since they must include these complex phenomena [9]. Model verification requires expensive hardware. These obstacles and others have made flapping flight development lag behind that of other types.

4.1.1 Physical Hardware

One way to overcome these challenges is with HIL modeling. HIL modeling allows phenomena in the system being modeled to be treated as a black box [1] and thus study the effects of various parameters on the lift produced without knowing all of the fluid phenomena that are at play [8]. A six degree of freedom (three per wing) flapping mechanism has been constructed [7]. It is instrumented to collect lift and drag data. Each wing is powered by three motors. Two of the motors are attached to worm gears that control η (pitch angle) and θ (deviation angle) via a differential gear [35, 37]. ϕ (sweep angle) is controlled directly by the third motor. See Figure 4.1(b). This set-up allows the wing to execute virtually any trajectory, provided that it does not violate the mechanical limits of the system, which are $\phi = \pm 90^\circ$, $\theta = +55^\circ / -105^\circ$, $\eta = \pm 180^\circ$ [10]. The objective is to find the optimal parameters to maximize lift. The mechanism flaps in water.

Trajectory Equations

The flapping trajectory is specified using 11 independent parameters, which define equations 4.1 through 4.3. These control equations allow the wing to execute a wide variety of trajectories. The objective of the optimization is to determine the combination of these parameters that


(a) View of Both Wings

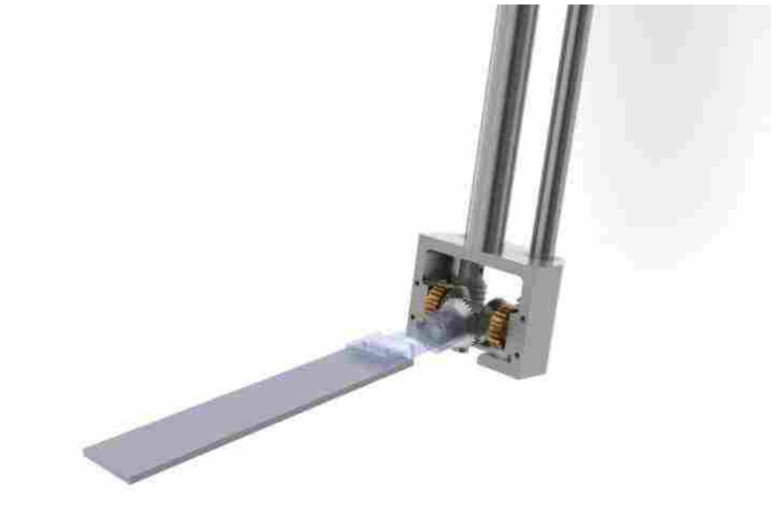

(b) Cad Model of One Wing

Figure 4.1: The Flapping Mechanism

optimizes some combination of lift and thrust. The kinematic pattern that determines the trajectory was proposed by Berman and Wang [20]. ϕ (sweep) is controlled by the equation

$$\phi(t) = \frac{\phi_m}{\sin^{-1}(K)} \sin^{-1}\left[K sin(2\pi f t)\right], \tag{4.1}$$

where $0 < K < 1$. When K approaches 0, $\phi(t)$ approaches a sine wave and as K approaches 1, $\phi(t)$ becomes a smoothed triangular wave. These equations were chosen based on experiments performed by Sane and Dickinson [18]. ϕ_m is the amplitude of the wave in degrees and f is the flapping frequency in Hz. Wing pitch, $\eta(t)$, with units of degrees, is controlled by the relationship

$$\eta(t) = \frac{\eta_m}{\tanh(C_\eta)} \tanh\left[C_\eta \sin(2\pi f t + \Phi_\eta)\right] + \eta_0, \tag{4.2}$$

where $0 < C_\eta < \infty$. When C_η approaches 0, $\eta(t)$ approaches a sine and when C_η approaches infinity, C_η approaches a square wave. η_0 is a constant offset and Φ_η is the phase shift. η_m determines the amplitude of the pitch. The deviation from the stroke plane, θ, is controlled by the equation

$$\theta(t) = \theta_m \cos(2N\pi f t + \Phi_\theta) + \theta_0, \tag{4.3}$$

where θ_m is the amplitude of the deviation in degrees, Φ_θ is the phase shift, and θ_0 is a constant offset. N determines how many cycles are performed per flap.

Once these paths are generated in MATLAB, they are sent to LabVIEW. The LabVIEW outputs are transmitted to an FPGA, where they are combined and sent to a current amplifier, which sends signals to the motors.

4.1.2 Optimization Setup

As is mentioned, in Section 1.1, optimizing this system by exhaustive testing is too expensive in terms of time and mechanical wear. This is largely because of the complexity of the design space. This makes the flapping flight mechanism a good candidate for VFO. Ryan George used traditional VFO with a response surface to optimize the system [10]. However, This approach still required over 2000 calls to the physical hardware. Because of the high cost of calling this system, it is well suited for optimization with an IDOE.

Because of difficulties finding the point $x_{p,m}$, several of the input parameters were held constant. Their values were taken from the optimal ones found by Berman and Wang [20]. The values of these parameters are $\theta_0 = 2.67°$, $\eta_0 = -90°$, $K = 0.796$, $C_\eta = 0.711$, $N = 2$, and $f = 19$Hz.

The physics model used as a surrogate is taken from the work of Berman and Wang [20]. It is a quasi steady model that outputs the lift and thrust generated during a flap. The model is intended to represent a hovering insect, which is the focus of this research.

For this test, the hardware was represented not by the physical system, but by another physics model. This is because of technical difficulties with the hardware. The hardware model was derived from the physics model. This was done using the assumptions discussed in Section 2.1. Using these assumptions, the hardware model was an artificially shifted version of the physics model, where the artificial shift,

$$\gamma_h = [6^\circ; 6^\circ; 6^\circ; 6^\circ; 6^\circ]. \tag{4.4}$$

The values of γ were chosen arbitrarily. $\alpha = 1$. This was chosen because of difficulty in finding $x_{p,m}$. Because it was very difficult to find it accurately, I restricted the search to a small area in which I knew the point existed. This was possible because the artificial shift was known. If α were not equal to one, the location of $x_{p,m}$ would not have been in the same location. This optimization problem can be formally stated as

$$\max_{x_h} H(x_h) \tag{4.5}$$

subject to:

$$0^\circ \leq x_{h1} \leq 90^\circ \tag{4.6}$$

$$0^\circ \leq x_{h2} \leq 180^\circ \tag{4.7}$$

$$-180^\circ \leq x_{h3} \leq 180^\circ \tag{4.8}$$

$$0^\circ \leq x_{h4} \leq 90^\circ \tag{4.9}$$

$$-180^\circ \leq x_{h5} \leq 180^\circ \tag{4.10}$$

where:

$$\gamma_h = [6^\circ; 6^\circ; 6^\circ; 6^\circ; 6^\circ] \tag{4.11}$$

$$H(x_h) = P(x_h - \gamma_h). \tag{4.12}$$

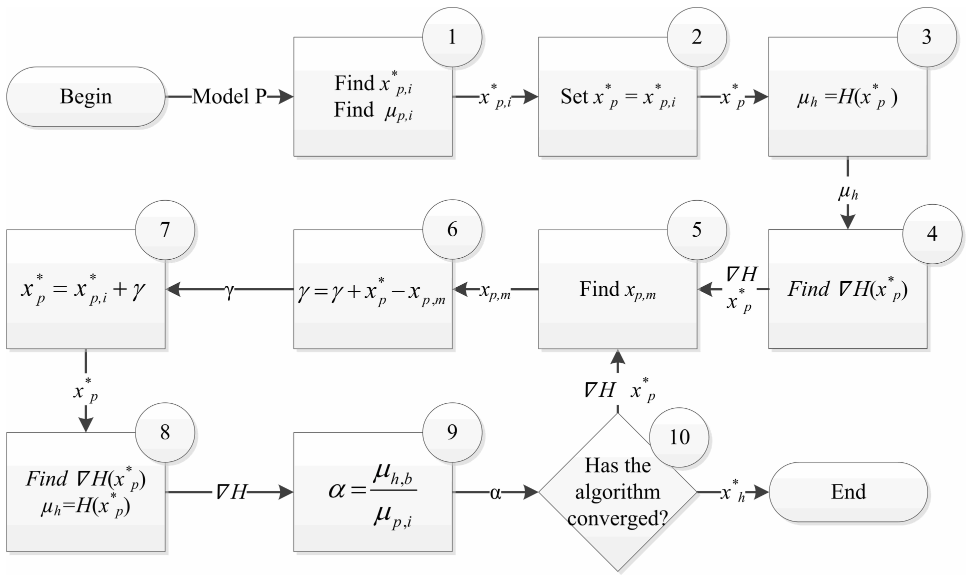


Figure 4.2: Flowchart of the Intelligent DOE

The most important sections of the code used in this optimization are included in appendix A.

4.1.3 Optimization Steps

The IDOE followed the steps outlined in Section 2.2. For convenience, the flowchart, Figure 2.1 is shown again here as Figure 4.2.

First, the starting point was set to be

$$x_p = \begin{bmatrix} \phi_m \\ \eta_m \\ \Phi_\eta \\ \theta_m \\ \Phi_\theta \end{bmatrix} = \begin{bmatrix} 90^\circ \\ 85.3^\circ \\ -97.9^\circ \\ 8.1^\circ \\ -109.2^\circ \end{bmatrix}. \tag{4.13}$$

$$\tag{4.14}$$

These values were the optimal ones found by Berman and Wang [20]. These optimal values were chosen to aid in the initial optimization, step 1 in Figure 4.2. The optimizer returned the point

$$x^*_{p,i} = [41.6217°; 71.7496°; -156.4045°; 15°; -109.8277°] \tag{4.15}$$

where $\mu_{p,i} = 0.245931$ newtons. This optimization problem can be formally stated as

$$\underset{x_{p,i}}{max}\, P(x_{p,i}) \tag{4.16}$$

subject to:

$$15° \leq x_{p,i1} \leq 75° \tag{4.17}$$

$$15° \leq x_{p,i2} \leq 165° \tag{4.18}$$

$$-165° \leq x_{p,i3} \leq 165° \tag{4.19}$$

$$15° \leq x_{p,i4} \leq 75° \tag{4.20}$$

$$-165° \leq x_{p,i5} \leq 165° \tag{4.21}$$

where:

$$P(x_{p,i}) \text{ is the physics model.} \tag{4.22}$$

Next, x^*_p was defined and set equal to $x^*_{p,i}$, as shown in Figure 4.2, step 2. During step 3, μ_h was found to be 0.2154 newtons and in step 4, $\nabla H(x^*_p)$ was found to be

$$[-0.2813°; 0.2919°; 0.1366°; 0.8570°; -0.0983°]. \tag{4.23}$$

Step 5 is to find the point in the physics model where the gradient equals $\nabla H(x^*_p)$. This can be formally defined as

$$\underset{x_{p,m}}{min}\, L(x_{p,m}) \tag{4.24}$$

subject to:

$$15^\circ \leq x_{p,m1} \leq 75^\circ \tag{4.25}$$

$$15^\circ \leq x_{p,m2} \leq 165^\circ \tag{4.26}$$

$$-165^\circ \leq x_{p,m3} \leq 165^\circ \tag{4.27}$$

$$15^\circ \leq x_{p,m4} \leq 75^\circ \tag{4.28}$$

$$-165^\circ \leq x_{p,m5} \leq 165^\circ \tag{4.29}$$

$$L = \|\nabla H(x_p^*) - \nabla P(x_{p,m})\|. \tag{4.30}$$

This point was found to be

$$x_{p,m} = [35.43^\circ; 65.63^\circ; -162.49^\circ; 8.984^\circ; -115.98^\circ]. \tag{4.31}$$

Using $x_{p,m}$ and x_p^*, the shift, γ, was defined to be

$$[6.182^\circ; 6.110^\circ; 6.088^\circ; 6.015^\circ; 6.156^\circ]. \tag{4.32}$$

As is shown in equation 4.4, the actual shift between the two models is

$$[6^\circ; 6^\circ; 6^\circ; 6^\circ; 6^\circ], \tag{4.33}$$

so the values found by the IDOE in equation 4.32 are close to what they should be. The differences between these equations originate from inaccuracies in how point $x_{p,m}$ was found. Next, the physics model and x_p^* were shifted within the design space by γ. The new value for x_p^* was

$$[47.80^\circ; 77.86^\circ; -150.31^\circ; 21.01^\circ; -103.67^\circ]. \tag{4.34}$$

Moving on to step 8, the new μ_h was found to be 0.2458 newtons and

$$\nabla H = [0.0053^\circ; -0.0083^\circ; -0.0156^\circ; -0.2097^\circ; 0.0027^\circ]. \tag{4.35}$$

Using the best value of μ_h to date, 0.2458, α was found to be 0.999745 in step 9. Step 10, the convergence criteria, did not signal convergence after this iteration, so one more iteration of steps 5 through 10 was needed. When a solution was converged on, the shift was found to be

$$\gamma = [6.046^\circ; 6.000^\circ; 5.986^\circ; 6.034^\circ; 6.038^\circ] \tag{4.36}$$

and the scaling factor was found to be 0.999745. The solution found was 0.2458 newtons. These values are all very close to the actual ones. More accuracy in finding $x_{p,m}$ and more stringent convergence criteria could have produced more accurate results.

4.1.4 Optimization Results

Finding the point where the physics model gradient matched the measured hardware gradient proved to be difficult for this example. It was done by using a genetic algorithm, then passing the best solution from the genetic algorithm into a gradient-based algorithm to improve it somewhat. This method was fairly effective, however, the accuracy of the solutions was not always sufficient to allow the IDOE to converge on a single solution as happened in the examples in Chapter 3. If the solutions are not sufficient, the solutions returned after each iteration of the IDOE quickly approach the actual solution, $\mu_{p,i}$, then bounce around it as shown in Figure 4.3.

Figure 4.3(a) shows three iterations of the unsuccessful optimization. The % error of the last points 11.4%. Figure 4.3(b) shows another unsuccessful optimization that was allowed to run for ten iterations. Notice that the solutions bounce around rather than converging on the actual solution. In both cases, the % error is relatively high. Figure 4.4 shows three iterations of a successful optimization with the IDOE. This run was successful because the genetic algorithm was allowed to run for much longer, which produced better solutions to the problem of where the gradients match. Figure 4.4 shows the values of the objective, lift, after each iteration of the IDOE. As Figure 4.5 shows, the IDOE got close during the second iteration, but did not trip the convergence criteria until the next iteration. The solution it found was within 0.048% of the actual solution.

The actual optimum value of the hardware, $\mu^*_{h,A}$ was found for comparison using known optimization methods. The optimization problem solved here is

$$\max_{x_h} H(x_h) \tag{4.37}$$

subject to:

$$0^\circ \leq x_{h1} \leq 90^\circ \tag{4.38}$$

$$0^\circ \leq x_{h2} \leq 180^\circ \tag{4.39}$$

$$-180^\circ \leq x_{h3} \leq 180^\circ \tag{4.40}$$

$$0^\circ \leq x_{h4} \leq 90^\circ \tag{4.41}$$

$$-180^\circ \leq x_{h5} \leq 180^\circ \tag{4.42}$$

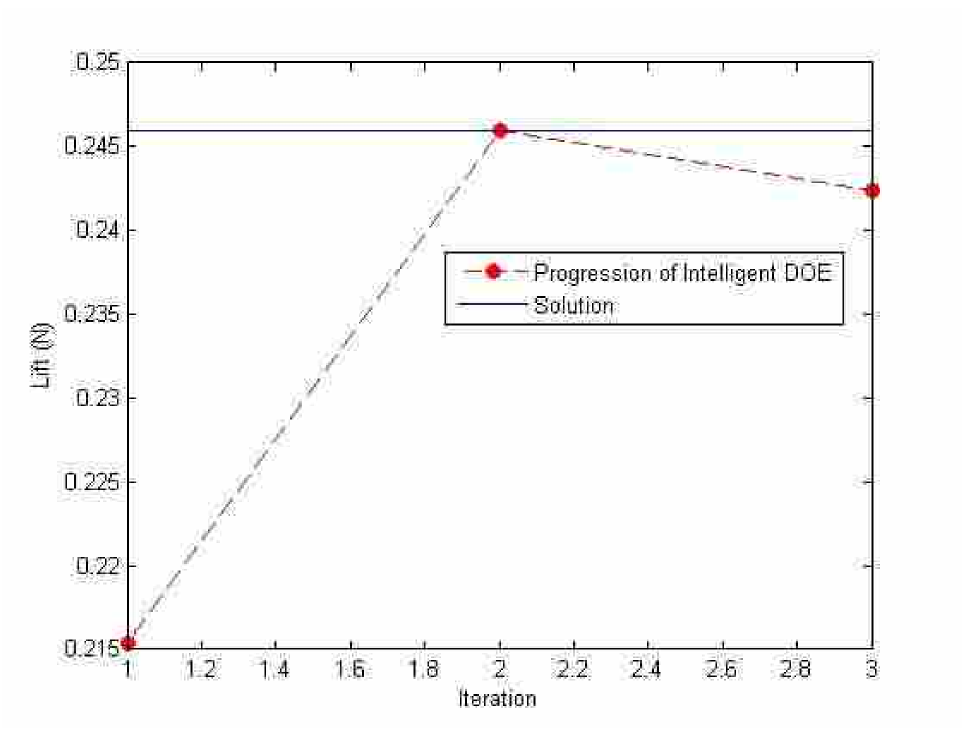
where:

$$\gamma_h = [6^\circ; 6^\circ; 6^\circ; 6^\circ; 6^\circ] \tag{4.43}$$

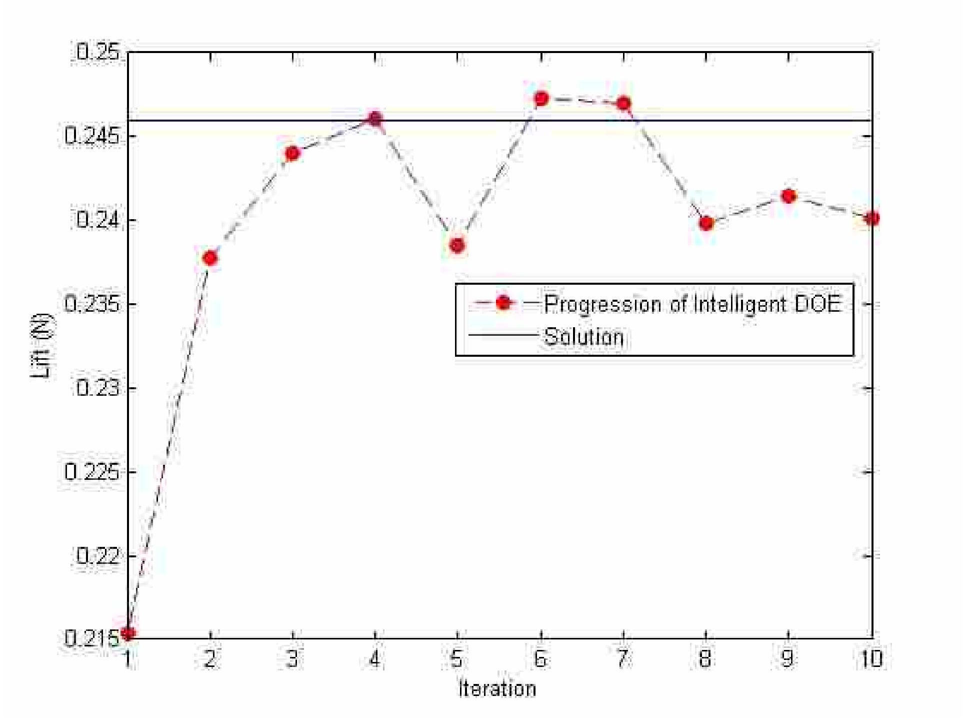
$$H(x_h) = P(x_h - \gamma_h). \tag{4.44}$$

$\mu^*_{h,A}$ could only be found in this way because the hardware was actually represented by a physics-based model. This was done outside of the other optimizations so that $\mu^*_{h,A}$ was unknown to them. The value of $\mu^*_{h,A}$ was used only to asses the accuracy of the solutions returned by the other optimization methods. For comparison, a DOE-based optimization was performed on the flapping flight model. The results are shown in Figure 4.5. The dependent axis, % Error, shown in the Figure, was calculated for each data point using the equation 3.19. The Box-Behnken best point optimization converged on a point that was within 0.62% of the solution, reaching this value after 552 hardware calls. The IDOE found a solution within 0.048% of the solution after 18 calls to the hardware. This is a 96.7% savings.

The IDOE was able to get very close to the solution with many fewer calls to the hardware than the DOE-based optimization runs. This allows for much more in depth study of various aspects of flapping flight due to decreased cost of optimization.



(a) Unsuccessful Run: 3 Iterations



(b) Unsuccessful Run: 10 Iterations

Figure 4.3: Performance of Flapping Flight Model Tests with Insufficient Accuracy in the Matched Gradient

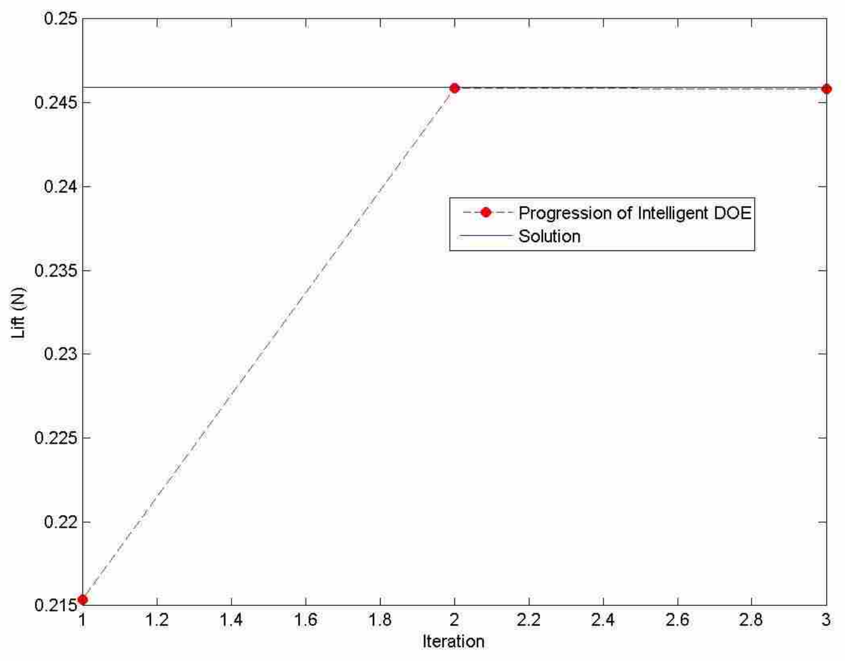


Figure 4.4: Successful Intelligent DOE Optimization

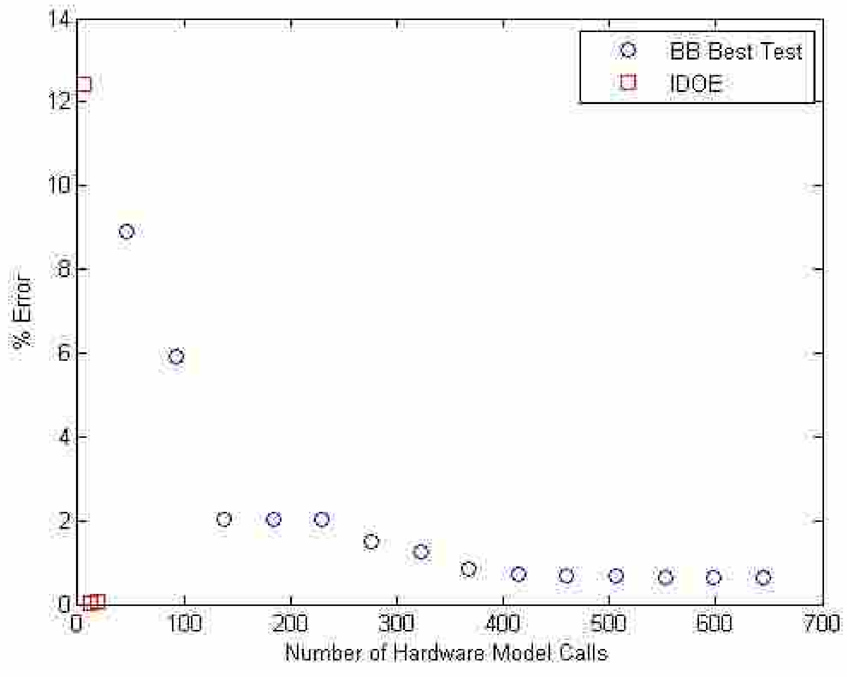


Figure 4.5: Performance of Flapping Flight Model Tests

CHAPTER 5. LIMITATIONS AND CHALLENGES

Chapters 3 and 4 show that the IDOE can solve a wide variety of optimization problems. There are, however, limitations to the effectiveness and applicability of the IDOE. One set of limitations deals with the physics model used. Another category of limitations deals with challenges in the implementation of the IDOE.

5.1 Limitations Associated with Physics Based Model

The first difficulty in implementing an IDOE is finding a physics based model of a system similar to the hardware being studied. Flapping flight is a popular research topic and many models are available in literature [21–23]. Models of other systems may not be as readily available. When using models of similar systems, it is important that the systems are similar enough, because some models may not match each other well enough to gain useful information. For example, a model of ice cream melting in warm surroundings would probably not be able to assist in the design of the scooter in Figure 1.1 at all, even if the ice cream model were very accurate. If there is not a model available, one could be constructed, but that could be time consuming and costly, maybe prohibitively so. However, the IDOE will not work properly if the systems are not sufficiently similar.

Another limitation to the method is finding a model that is sufficiently representative of the system being studied. Even if the systems are similar, their models may not be. Figure 1.1 shows a dragster and scooter. These models would share many elements, however, a model of a motorcycle would probably inform design of a scooter better than a dragster model would. It is important to find models that match each other as closely as possible. Similar to this is the idea of model stretching. The assumptions in Section 2.1 do not account for the fact that the objective surfaces may not have the exact same shape. In other words, the physics model could be stretched in one or more dimensions, giving it a slightly different shape. This leads to gradients that are slightly different than expected. Usually, the IDOE can still converge on a solution when one model is stretched, but it is less efficient, requiring more calls to the hardware. An example of this is shown in Figure 5.1. This is the same problem that was solved in Section 3.1, except that the hardware has been stretched by a factor of 0.6. When solved with no stretching, the intelligent DOE found the solution in four calls. As shown here with stretching, the intelligent DOE needed 12 hardware calls to solve the problem. This is the same number that was needed by the surrogate response surface optimization. This shows that the IDOE can solve problems that involve stretching, but requires more hardware model calls.

A final limitation associated with the model is that the physics model solution may be infeasible for the hardware. Physical hardware is usually bounded in some way and the physics model may not have these restrictions. If the physics model prescribes a solution that is infeasible for the hardware, the intelligent DOE can stall because it is trying to go to a certain region of the design space, but is repeatedly pushed out of it by constraints.

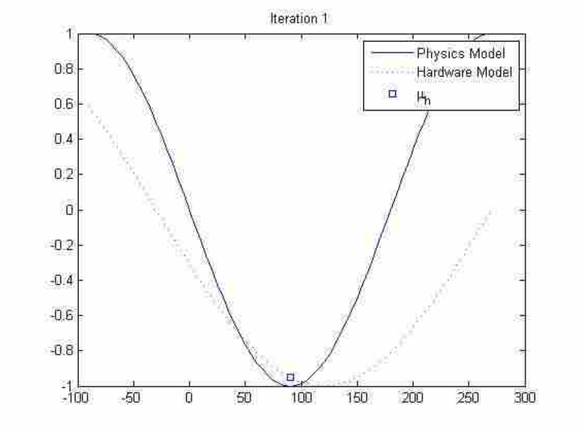
5.2 Challenges Associated with Implementation of the IDOE

One major challenge in using the IDOE is finding the point where the physics model gradient matches the measured hardware gradient. In some cases, such as the examples in sections 3.1 and 3.2, finding this point can be accomplished by gradient based algorithms. This method, however, was ineffective in the 20 dimensional sine problem in Section 3.3, so a genetic algorithm was used, which proved to be sufficient. In the flapping flight model optimization, a genetic algorithm alone was not very effective and needed to be used in conjunction with a gradient based algorithm in order to find the matching gradient. Even this was not as accurate as could be desired, preventing smooth convergence. If this point is not found accurately enough, the IDOE will move in the wrong direction and will be unable to find a solution.

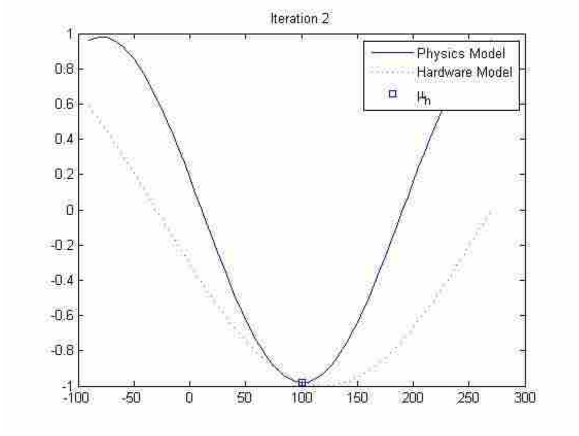
Another challenge is determining when the IDOE has converged on a solution. Some methods for accomplishing this are presented in Chapter 3, but these need to be adjusted to fit the system being optimized. Other convergence criteria besides the ones listed here are possible. Any convergence criteria, especially those involving convergence tolerances, need to be adjusted to fit the model.

5.3 Other Limitations

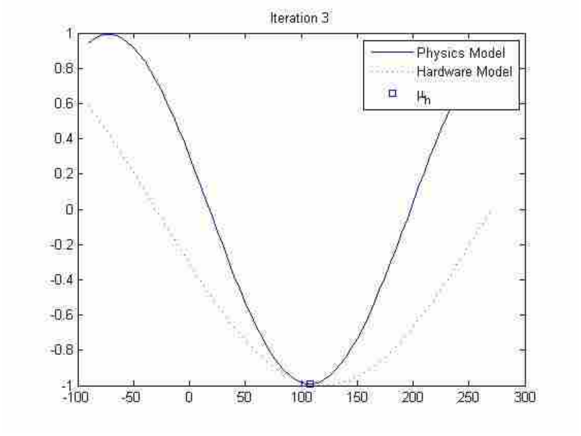
Another challenge is that of noise in the hardware model. Physical hardware usually has inaccuracies so that a single point in the design space will return slightly different values on different runs. This can be a result of many factors. The problem this creates for the IDOE is that it distorts the gradient measured by the hardware. This can send the IDOE in the wrong direction, preventing convergence. One way to overcome this problem is to call the hardware multiple times at each point and average the results. This is effective, but is rarely exactly accurate. Also, it drastically increases the number of calls needed to find a gradient, increasing the overall cost of the optimization. Systems whose noise is small compared to the output signal have less problems with noise than systems with large noise to signal ratios, so it is preferable that the system have as little noise as possible. Another thing that can help mitigate the noise problem is increasing the step size used when taking the gradient. Increasing the step can help decrease the effects of inaccurate hardware values. Choosing an appropriate step size can be difficult [13]. Using a step size that is too small will amplify inaccuracies from the hardware, but a large step size can change the calculated value of the gradient.



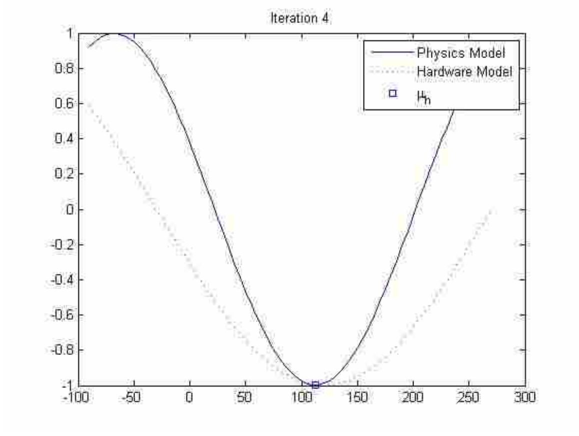
(a) Iteration 1



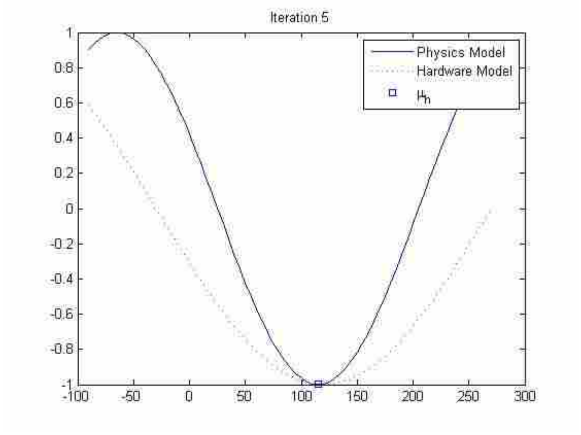
(b) Iteration 2



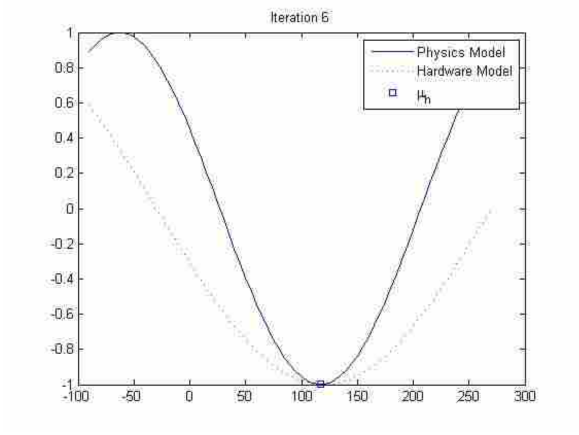
(c) Iteration 3



(d) Iteration 4



(e) Iteration 5



(f) Iteration 6

Figure 5.1: Progression of the IDOE Through an Optimization of a Stretched Model

CHAPTER 6. CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

Hardware in the loop (HIL) modeling is a powerful way of representing complex physical systems. It allows for accurate modeling of systems without dealing with all of the physical phenomena at play. HIL models can be expensive to call, so variable fidelity optimization (VFO) can be used to optimize these models. However, some HIL systems are too expensive even for traditional VFO. A method is needed to optimize systems like this that requires fewer calls to the expensive hardware.

An intelligent design of experiments (IDOE) has been developed to reduce the cost of VFO for HIL systems. It uses an inexpensive physics-based surrogate to guide the optimization of an expensive hardware model. This is accomplished by establishing a mapping between the models, then adjusting the physics model so that it resembles the hardware more closely. The mapping is defined in terms of a shift within the design space, γ, and a scaling on the objective values, α. The values of these mapping parameters are found in Chapter 2 by first, finding the solution to the physics model, second, finding the hardware gradient at the point where the physics model solution is, then finding the point in the physics model space where the physics model shows the same gradient that the hardware measured. Once this is done, the physics model is shifted and scaled so that the points whose gradients match are in the same location. This makes the physics model a better approximation of the hardware. These steps are repeated until the physics model solution is in the same location as the hardware solution, at which point, the hardware solution is known.

Through various tests, this intelligent DOE was shown to reduce the number of calls to the hardware by up to 98% when compared to traditional DOE based optimization and up to 97% when compared to traditional VFO using a DOE based surrogate. The intelligent DOE saved cost on all of the tests conducted where the physics model was of the same form as the hardware.

In Section 3.4, the physics model was of a different form than the hardware model. The IDOE did converge on a solution, but took more calls than the surrogate response surface optimization. The response surface was of the same form as the hardware model, so it fit very well and was able to optimize using fewer hardware calls than the IDOE. As Section 3.5 discussed, the savings became more dramatic as the number of design variables increased. This was due to the fact that the number of calls to perform most unintelligent DOEs increased faster than linearly as the number of variables increased, while the number of calls for the intelligent increased linearly with the number of variables. It was also demonstrated in Section 3.3.3 that traditional DOE-based optimization was sometimes unable to converge on the global solution. This could happen when bounds were shrunk so that the global solution was infeasible.

The intelligent DOE presented in this thesis is able to optimize expensive HIL models using fewer hardware calls than are required by traditional VFO. This allows engineers to better optimize expensive hardware systems by drastically decreasing the cost of doing so. This means that more can be learned from an expensive hardware model, which increases its usefulness.

6.2 Future Work

The effectiveness of the IDOE could be greatly improved if there were a better way to find the point where the physics model has the same gradient as the measured hardware gradient. This is mentioned as being a problem in Section 5.2. At present, this is a major limitation to the usefulness of the IDOE. In problems with small numbers of design variables and simple objective surfaces, such as those in Sections 3.1 and 3.2, this step was relatively straight forward and could be quickly accomplished by gradient based algorithms. In Section 3.3, however, the large number of variables, 20, created a more complex objective surface, which prevented a gradient-based algorithm from finding a solution. For this problem, a genetic algorithm was able to find the point accurately enough. However, in the flapping flight example, Section 4.1.4, it was not able to do so. In this case, a genetic algorithm was used to get close to the solution, then a gradient based algorithm was used to converge to a solution. This method worked moderately well, but required many generations of the genetic algorithm in order to get close enough to the correct solution. Unsurprisingly, increased accuracy came as the number of generations increased. In summary, current methods work well enough to verify the effectiveness of the intelligent DOE, but are time intensive enough to be cumbersome. A better way to find the matching gradient would greatly increase the usefulness of the IDOE.

REFERENCES

[1] Kruse, P. M., Wegener, J., and Wappler, S., 2009. "A highly configurable test system for evolutionary blackbox testing of embedded systems." In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1545–1552. 1, 36

[2] Bringmann, E., and Stuttgart Kramer, A., 2008. "Model-based testing of automotive systems." In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pp. 485–493. 1

[3] Boot, R., and Richert, J., 1999. "Automated test of ecus in a hardware-in-the-loop simulation environment." In *Proceedings of the 1999 EEE International Symposium on Computer Aided Control System Design*. 1

[4] Verhoeff, L., Verburg, D., Lupker, H., and Kusters, L., 2000. "Vehil: A full-scale test methodology for intelligent transport systems, vehicles and subsystems." In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*. 1

[5] Gietelinkab, O., Ploeg, J., Schutter, B. D., and Verhaegen, M., 2006. "Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations." *Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility,* **44**, pp. 569–590. 1

[6] Gietelink, O., Labibes, K., Verburg, D., and Oostendorp, A., 2004. "Pre-crash system validation with prescan and vehil." In *IEEE Intelligent Vehicles Symposium*. 1

[7] George, R., Colton, M., Mattson, C., and Thomson, S., 2012. "A differentially driven flapping wing mechanism for force analysis and trajectory optimization." *International Journal of Micro Air Vehicles,* **4**(1), pp. 31–49. 1, 35, 36

[8] Hunt, R., Hornby, G. S., and Lohn, J. D., 2005. "Toward evolved flight." In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pp. 957–964. 1, 36

[9] chi Wu, J., and Popovic, Z., 2003. "Realistic modeling of bird flight animations." *ACM Transactions on Graphics,* **22**, July, pp. 888–895. 1, 36

[10] George, R. B., 2011. "Design and analysis of a flapping wing mechansim for optimization." Master's thesis, Brigham Young University, August. 2, 13, 36, 38

[11] Singh, B., and Chopra, I., 2008. "Insect-based hover-capable flapping wings for micro air vehicles: Experiments and analysis." *AIAA Journal,* **46**, pp. 2125–2135. 2, 3, 35

[12] Roux, W. J., and Haftka, R. T., 1998. "Response surface approximations for structural optimization." *International Journal for Numerical Methods in Engineering,* **42**(3), June, pp. 517–534. 2

[13] Boker, A. J., J. E. Dennis, J., Frank, P. D., Serafini, D. B., Torczon, V., and Trosset, M. W., 1998. A rigorous framework for optimization of expensive functions by surrogates Tech. rep., ICASE, November. 2, 49

[14] Koziel, S., and Ogurtsov, S., 2011. “Improved variable-fidelity optimization algorithm for simulation-driven design of antennas.” In *Antennas and Propagation (APSURSI), 2011 IEEE International Symposium on*, pp. 2419–2422. 2, 14

[15] Queipoa, N. V., Haftka, R. T., Shyya, W., Goela, T., Vaidyanathana, R., and Tuckerb, P. K., 2005. “Surrogate-based analysis and optimization.” *Progress in Aerospace Sciences,* **41**(1), January, pp. 1–28. 2, 14

[16] Jones, D., 2001. “A taxonomy of global optimization methods based on response surfaces.” *Journal of Global Optimization,* **21**, pp. 345–383. 2

[17] Duffield, M. L., Mattson, C. A., and Colton, M., 2012. “Towards variable fidelity optimization with hardware in the loop for flapping flight.”. 2

[18] Sane, S. P., and Dickinson, M. H., 2001. “The control of flight force by a flapping wing: Lift and drag production.” *The Journal of Experimental Biology,* **204**, pp. 2607–2626. 3, 38

[19] Stanford, B., Beran, P., and Kobayashi, M., 2011. “Aeroelastic optimization of flapping wing venation: A cellular division approach.” *AIAA Journal,* **50**, pp. 938–951. 3, 35

[20] Berman, G. J., and Wang, Z. J., 2007. “Energy-minimizing kinematics in hovering insect flight.” *Journal of Fluid Mechanics,* **582**, pp. 153–168. 3, 35, 37, 38, 39, 41

[21] Walker, W., Patil, M., and Canfield, R., 2012. “Aeroelastic tailoring of flapping membrane wings for maximum thrust and propulsive efficiency.”. 3, 35, 47

[22] Taha, H. E., Hajj, M. R., and Nayfeh, A. H., 2012. “Optimization of wing kinematics for hovering mavs using calculus of variation.”. 3, 47

[23] Peters, H., Goosen, J., and van Keulen, F., 2012. “Flapping wing performance related to wing planform and wing kinematics.”. 3, 35, 36, 47

[24] Rodriguez, J., Renaud, J., and Watson, L., 1998. “Convergence of trust using variable fidelity region augmented lagrangian approximation data.” *Structural Optimization,* **15**, pp. 141–156. 8

[25] Parkinson, A. Optimization-based design BYU Academic Publishing. 8, 9

[26] Box, G. E. P., Hunter, J. S., and Hunter, W. G., 2005. *Statistics for Experimenters: Design, Innovation, and Discovery.*, 2 ed. Wiley, May. 11

[27] Box, G. E. P., and Behnken, D. W., 1960. “Some new three level designs for the study of quantitative variables.” *Technometrics,* **2**(2), November, pp. 455–475. 26

[28] Myers, R. H., and Montgomery, D. C., 1995. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. John Wiley & Sons, Inc. 26

[29] Varon, J., Wenker, O. C., and Robert E Fromm, J., 1997. "Aeromedical transport: Facts and fiction." *The Internet Journal of Emergency and Intensive Care Medicine,* **1**. 35

[30] Green, W. E., and Oh, P. Y., 2006. "A fixed-wing aircraft for hovering in caves, tunnels, and buildings." In *Proceedings of the 2006 American Control Conference*. 35

[31] Deng, X., Schenato, L., Wu, W., and Sastry, S., 2006. "Flapping flight for biometric robotic insects: Part isystem modeling." *Robotics, IEEE Transactions on,* **22**(4), Aug, pp. 776–788. 35

[32] Spedding, G. R., and Lissaman, P. B. S., 1998. "Technical aspects of microscale flight systems." *Journal of Avian Biology,* **29**(4), pp. 458–468. 35

[33] Richter, C., and Lipson, H., 2011. "Untethered hovering flapping flight of a 3d-printed mechanical insect." *Aritficial Life,* **17**, pp. 73–87. 35

[34] Ansari, S. A., Phillips, N., Stabler, G., Wilkins, P. C., Zbikowski, R., and Knowles, K., 2009. "Experimental investigation of some aspects of insect-like flapping flight aerodynamics for application to micro air vehicles." *Experiments in Fluids,* **46**(5), May, pp. 777–798. 35

[35] Madangopal, R., Khan, Z. A., and Agrawal, S. K., 2005. "Biologically inspired design of small flapping wing air vehicles using four-bar mechanisms and quasi-steady aerodynamics." *Journal of Mechanical Design,* **127**, pp. 809–816. 35, 36

[36] Sane, S. P., and Dickinson, M. H., 2002. "The aerodynamic effects of wing rotation and a revised quasi-steady model of flapping flight." *The Journal of Experimental Biology,* **205**, p. 10871096. 35

[37] VandenBerg, C., and Ellington, C. P., 1997. "The three-dimensional leading-edge vortex of a hovering model hawkmoth." *Philosophical Transactions: Biological Sciences,* **352**(1351), pp. 329–340. 35, 36

[38] Wang, Z. J., 2000. "Vortex shedding and frequency selection in flapping flight." *Journal of Fluid Mechanics,* **410**, pp. 323–341. 35

[39] Green, W., and Oh, P., 2005. "A mav that flies like an airplane and hovers like a helicopter." In *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pp. 693–698. 35

[40] Weis-Fogh, T., 1973. "Quick estimates of flight fitness in hovering animals, including novel mechanisms for lift production." *Journal of Experimental Biology,* **59**, pp. 169–230. 36

[41] Wootton, R., 1999. "How flies fly." *Macmillan Magazines Ltd,* **400**, pp. 112–113. 36

[42] Ennos, R., 1989. "The kinematics and aerodynamics of the free flight of some diptera." *Journal of Experimental Biology,* **142**, pp. 49–85. 36

[43] Ellington, C. P., 1999. "The novel aerodynamics of insect flight: Applications to micro-air vehicles." *Journal of Experimental Biology,* **202**, pp. 3439–3448. 36

[44] Sun, M., and Xiong, Y., 2005. “Dynamic flight stability of a hovering bumblebee.” *Journal of Experimental Biology,* **208**, pp. 447–459. 36

[45] Taylor, G. K., and Thomas, A. L. R., 2003. “Dynamic flight stability in the desert locust schistocerca gregaria.” *Journal of Experimental Biology,* **206**, pp. 2803–2829. 36

APPENDIX A. MATLAB CODE FROM FLAPPING FLIGHT OPTIMIZATION

This appendix includes some of the main pieces of code used to complete the flapping flight optimization.

A.1 Main Function

```
clc; clear variables; format compact;
tic; %timer for the whole program

%% User Inputs
%enter the name
LabViewOutputFileName='Icecream';
%define the starting physics inputs
x_p_star=[
    %inputs for the sweep (phi)
    90.0; %max sweep in degrees
    %inputs for the pitch (eta)
    85.3; %max pitch in degrees
    -97.9; %phase shift in eta in degrees
    %inputs for deviation (theta)
    8.10;   %max deviation in degrees
    -109.2];%phase shift in theta direction in degrees

%% Set up stuff for the algorithm
global scaling; scaling=1;%Set the starting scaling factor to one.
shift=zeros(length(x_p_star),1); %Set the starting shift to zero.

global HardwareUpperBound HardwareLowerBound
global PhysicsUpperBound  PhysicsLowerBound

[HardwareLowerBound,HardwareUpperBound,PhysicsLowerBound,...
    PhysicsUpperBound]=...
    FormHardwareAndPhysicsBounds(); %form the bounds

%% Find the solution using the physics model
OPTIONS1 = optimset('MaxFunEvals',5000,'TolFun',1e-17,'Display','iter'...
    ,'Algorithm','active-set');
[x_p_star_initial,mu_p_target,junk1,output1]=fmincon(@(x_p_star)...
    Physics(x_p_star,shift,scaling),x_p_star,[],[],[],[],...
    PhysicsLowerBound,PhysicsUpperBound,[],OPTIONS1); %#ok<ASGLU>

gradPStar=FindPhysicsGradient(x_p_star_initial,shift);
```

```

%count the physics model calls
PhysicsModelCalls=output1.funcCount;

% initialize x_p_star
x_p_star=x_p_star_initial;

%% if one of the inputs (x_p_star) is out of bounds, set it equal to...
%  the bound to make it binding
for j=1:length(x_p_star)
    if x_p_star(j)>HardwareUpperBound(j)
        x_p_star(j)=HardwareUpperBound(j);
    elseif x_p_star(j)<HardwareLowerBound(j)
        x_p_star(j)=HardwareLowerBound(j);
    end
end

%convert x_p_star to x_h
x_h=x_p_star;

%% call the hardware at the x_p_star_initial
[mu_h_1]=Hardware(x_h,LabViewOutputFileName); %mu_h_1=lift

%find gradient at x_p_star_initial
[gradH,HardwareModelCalls]=...
    FindHardwareGradient(x_h,HardwareUpperBound,HardwareLowerBound,...
    mu_h_1,1,LabViewOutputFileName);
mu_h_forScaling=mu_h_1; %this is the running max for the scaling

%% make a vector of outputs
    vec_of_x_h(1,1)=1;
    vec_of_x_h(2,1)=mu_h_1;
    for l=3:length(x_h)+2          %x_h at the iteration
        vec_of_x_h(l,1)= x_h(l-2);
    end
    for l=length(x_h)+3:2*length(x_h)+2 %gradH at the iteration
        vec_of_x_h(l,1)= gradH(l-length(x_h)-2);
    end
    vec_of_x_h(l+1,1)=scaling;

%% setup things for the loop
MaxNumberOfLoops=15;
i=2; %the iterator in the while loop
HasConverged=false;
optimizationFunction='SolveWithHybridAlgorithm'; %fmincon or fsolve

%% The loop
while HasConverged==false && i<=MaxNumberOfLoops

    display(i);

    %% find the point where the physics model...
    %predicts What the Hardware model measured

    if strcmp(optimizationFunction,'fmincon')
        %set up the starting point
        fminconStartPoint=x_p_star+[+.1;+.1;+.1;+.1;+.1];
        %find the point
        OPTIONS2 = optimset('MaxFunEvals',50000,'TolFun',1e-3,'Display'...
            ,'iter','MaxIter',5000);
        [x_p_toMatchHardware,exitflag,error2,output2]=...
            fmincon(@(x_p_star) GradientDifference(x_p_star,shift,gradH...
            ,optimizationFunction),fminconStartPoint,[],[],[],[],...
            PhysicsLowerBound,PhysicsUpperBound,@(x_p_star)...
            nonlcon(x_p_star,scaling,shift),OPTIONS2); %#ok<ASGLU>
        %count the physics model calls
        PhysicsModelCalls=PhysicsModelCalls+output2.funcCount;
    elseif strcmp(optimizationFunction,'SolveWithHybridAlgorithm')
        [x_p_toMatchHardware,error2,exitflag,output2]=...
            FindMatchingGradientWithHybrid(shift,gradH); %#ok<NASGU,ASGLU>
    elseif strcmp(optimizationFunction,'fsolve')
        fsolveStartPoint=x_p_star+[+.01;+.01;+.01;+.01;+.01];
        OPTIONS3 = optimset('TolFun', 1e-7,'Display','notify','MaxIter'...
            ,5000,'MaxFunEvals',5000,'MaxTime',120);
        [x_p_toMatchHardware,error2,exitflag,output2]=...
            fsolve(@(x_p_star) GradientDifference(x_p_star,shift,gradH...
            ,optimizationFunction),fsolveStartPoint,OPTIONS3); %#ok<ASGLU>
        PhysicsModelCalls=PhysicsModelCalls+output2.funcCount;
    elseif strcmp(optimizationFunction,'SolveWithOmniscience')
        [x_p_toMatchHardware,error2,exitflag] =...
            FindMatchingPointWithOmniscence(x_p_star_initial,shift,gradH...
            ,PhysicsUpperBound,PhysicsLowerBound); %#ok<NASGU>
    elseif strcmp(optimizationFunction,'SolveWithGeneticAlgorithm')
        [x_p_toMatchHardware,error2,exitflag,output2]=...
            FindMatchingGradientWithGA(shift,gradH); %#ok<NASGU,ASGLU>
    else
        diplay('enter a valid optimiztion function.');
    end

    %% Calculate stuff for the algorithm
    %find the physics objective at x_p_toMatchHardware
    mu_p=Physics(x_p_toMatchHardware,shift,scaling);

    %update the shift
    shift=shift+(x_p_star-x_p_toMatchHardware);

    %update x_p_star
    x_p_star=x_p_star_initial+shift;

    %% calculate more stuff
    %convert x_p_star to x_h
    x_h=x_p_star;

    %call hardware at the new x_h
    [mu_h_1]=Hardware(x_h,LabViewOutputFileName);
    HardwareModelCalls=HardwareModelCalls+1;
    [gradH,HardwareModelCalls]=FindHardwareGradient(x_h,...
        HardwareUpperBound,HardwareLowerBound,mu_h_1,...
        HardwareModelCalls,LabViewOutputFileName);
    ThisShouldBeZero=(gradPStar-gradH)';

    %% Update the scaling
    if mu_h_1<mu_h_forScaling
        mu_h_forScaling=mu_h_1;
    end

    FindNewScalingFactor(mu_h_forScaling,mu_p_target);

    %% add the iteration results to the vector of outputs
    vec_of_x_h(1,i)=i;
    vec_of_x_h(2,i)=mu_h_1;
    for l=3:length(x_h)+2         %x_h at the iteration
        vec_of_x_h(l,i)= x_h(l-2);
    end
    for l=length(x_h)+3:2*length(x_h)+2 %gradH at the iteration
        vec_of_x_h(l,i)= gradH(l-length(x_h)-2);
    end
    vec_of_x_h(l+1,i)=scaling;
    vec_of_x_h(l+2,i)=error2;

    %% Check for convergance and exit the loop if it has converged

    %if the last three objective values are equal
    %the amount of change in objective that is considered zero
    convergenceTolerance=.001;
    if i>=4 && ...
            sqrt((vec_of_x_h(2,i-1)-vec_of_x_h(2,i-2))^2)<...
            converganceTolerance &&...
            sqrt((vec_of_x_h(2,i-1)-vec_of_x_h(2,i-3))^2)<...
            converganceTolerance
        HasConverged=true;
        display('Converged by Repeated Objective Values');
        display(sprintf(['The Solution is: ' num2str(mu_h_1)]));
        display(sprintf(['The Hardware Was Called ',...
            num2str(HardwareModelCalls),' Times.']));
        break
    end

    %check if the gradient equals the gradient at the physics solution
    gradMagCutoff=.00000001;

    if norm(gradPStar-gradH)<gradMagCutoff;
        HasConverged=true;
        display('The Gradient Equals the Gradient at the Physics Solution.');
        display(['The Solution is ' num2str(mu_h_1)]);
        display(sprintf(['The Hardware Was Called ',...
            num2str(HardwareModelCalls),' Times.']));
    end

    %% get ready for the next iteration
    %increment i for the next run of the loop
    if HasConverged == false
```

```
        i=i+1;
    end
end

%calculate and print the performance results
if HasConverged
    design=bbdesign(20);
    bbHardwareCalls=1*length(design(:,1))*1;%1 DOEs,1 flap per test
    display(sprintf(['A Box Behnken would take ' num2str(bbHardwareCalls)...
        ' calls.']));
    CallSavings=100*(1-HardwareModelCalls/bbHardwareCalls);
    display(sprintf(['This is ', num2str(CallSavings), ...
        ' percent less than five box behnken experiments.']));
else
    display('The Convergance Criteria Were Not Met.');
end

%plot resutls
subplot(2,1,1)
plot(vec_of_x_h(1,:),vec_of_x_h(2,:))
xlabel('iteration');
ylabel('lift');
subplot(2,1,2)
plot(vec_of_x_h(1,:),vec_of_x_h(14,:))
ylabel('GA fitness');

plot(vec_of_x_h(1,:),-1.*vec_of_x_h(2,:),'ro')
xlabel('Iteration');
ylabel('Lift (n)');
hline=refline(0,-mu_p_target);
legend('Progression of Intelligent DOE','\mu_{p,i}');

%% clean up the workspace
clear output j junk1 OPTIONS2 OPTIONS1 k Domain
clear HardwareRange PhysicsRange error2 showPlotOfEachItertion
clear CallSavings MaxNumberOfLoops SaveMovie ShowMovieAfterRun
clear bbHardwareCalls scrsz surfaceHardware surfacePhysics
clear x1axis x2axis writerObj PlotHardware PlotPhysics design
clear output1 output2 l junk0 OPTIONS3 exitflag mu_h_2
clear optimizationFunction ub lb LabViewOutputFilename
clear gradMagCutoff
```

A.2 Physics Model

```
function [ Lift ] = Physics( x_p,shift,scaling )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

%% user inputs
x_p_shifted=(x_p-shift).*pi./180;

```

```
phi_m    =x_p_shifted(1);%max sweep in rad
eta_m    =x_p_shifted(2);%max pitch in rad
phi_eta  =x_p_shifted(3);%phase shift in eta in rad
theta_m  =x_p_shifted(4);%max deviation in rad
phi_theta=x_p_shifted(5);%phase shift in theta direction in rad

% phi_m=90*(pi/180);  %max sweep in rad
% theta_m=8.1*pi/180; %max deviation in rad
% eta_m=85.3*pi/180;  %max pitch in rad
theta_0=2.67*pi/180;  %constant offset in theta direction n rad
eta_0=-90*pi/180;     %constant offset in eta direction n rad
% phi_theta=-109.2*pi/180; %phase shift in theta direction in rad
% phi_eta=-97.9*(pi/180);  %phase shift in eta in rad

%% define some parameters
cBar=18.26; %mean chord length in mm
R=51.9; %wing radius in mm
numberOfSlices=5; %number of chords that will be used

rho_f=1.29/(1000^3); %density of air in kg/mm^3
b=.01; %thickness of wing in mm
mu1=.2;
mu2=.2;
ct=1.678;
cr=pi;
cd_of_0=.07;
cd_of_piOver2=3.06;
f=19; %flapping frequancy in Hz

K=.796;     %shape of phi 0<K<1
C_eta=.711; %shape of eta 0<C_eta<inf
N=2;        %harmonic of theta

Mwing=47e-6; %mass of wing in kg
m=1648e-6;   %mass of insect in kg

%% calculate vetors of r values and c(r) values
dr=R/numberOfSlices; %width of slice in mm
rValues=(dr/2:dr:R-dr/2); %r cordinate of center of each slice
c_of_rValues=(4*cBar/pi).*sqrt(1-(rValues.^2)./(R^2));

%% calculate wing kinematics
dt=1/(50*f);
t=(0:dt:1/f)'; %time in seconds from 0 to one period

%sweep in rad
phi=(phi_m/asin(K))*asin(K*sin(2*pi*f.*t));
phiDot=2*phi_m*K*cos(2*pi*f.*t)*pi*f./sqrt(1-K^2.*sin(2*pi*f.*t).^2)./...
    asin(K); %in rad
phiDotDot=-4*phi_m*K*sin(2*pi*f.*t)*pi^2*f^2./sqrt(1-K^2.*sin(2*pi*f.*t)...
    .^2)./asin(K)+...
    4*phi_m*K^3*cos(2*pi*f.*t).^2*pi^2*f^2./(1-K^2.*sin(2*pi*f.*t).^2)...
    .^(3/2)./asin(K); %in rad

```

```
%pitch in rad
eta=eta_m*tanh(C_eta*sin(2*pi*f.*t+phi_eta))./tanh(C_eta)+eta_0;
etaDot=1/tanh(C_eta).*(2.*eta_m.*(1-tanh(C_eta.*sin(2*pi*f.*t+phi_eta))...
    .^2).*C_eta.*cos(2*pi*f.*t+phi_eta).*pi.*f);

%deviation in rad
theta=theta_m.*cos(2*N*pi*f.*t+phi_theta)+theta_0;
thetaDot=-theta_m*2*N*pi*f.*sin(2*N*pi*f.*t+phi_theta);
thetaDotDot=-theta_m*(2*N*pi*f)^2.*cos(2*N*pi*f.*t+phi_theta);

%% loop through each chord
totalLift=0;
for j=1:length(rValues)

    r=rValues(j); %radius to current chord in mm
    c_of_r=c_of_rValues(j);  %chord length in mm

    %% calculate intermediate things
    m11=.25*pi*rho_f*b*b; %kg/mm
    m22=.25*pi*rho_f*c_of_r^2; %kg/mm
%     I_a=(1/128)*pi*rho_f*(c_of_r^2+b*b)^2; %kg*mm

    %% calculate velocities
    vxPrime=r*(phiDot.*cos(theta).*cos(eta)+thetaDot.*sin(eta)); %in mm/s
    vyPrime=r*(thetaDot.*cos(eta)-phiDot.*cos(theta).*sin(eta)); %in mm/s

    axPrime=r.*((phiDotDot.*cos(theta)+thetaDot.*(etaDot-phiDot.*...
        sin(theta))).*cos(eta)+(thetaDotDot-etaDot.*phiDot.*cos(theta))...
        .*sin(eta)); %in mm/s^2
    ayPrime=r.*((thetaDot.*(etaDot-phiDot.*sin(theta))-phiDotDot.*...
        cos(theta)).*sin(eta)+(thetaDotDot-etaDot.*phiDot.*cos(theta))...
        .*cos(eta)); %in mm/s^2

    %% calculate alpha(t)
    alpha=zeros(length(t),1); %initialize alpha
    for i=1:length(t)
        R1=[cos(eta(i)),sin(eta(i));-sin(eta(i)),cos(eta(i))];
        R2=[-sin(phi(i)),cos(phi(i)),0;-sin(theta(i))*cos(phi(i)),...
            -sin(theta(i))*sin(phi(i)),cos(theta(i))];
        Vel=pinv(R1*R2)*[vxPrime(i);vyPrime(i)];

        if Vel(2)>=0 %when forward velocity in aerodynamic frame >=0
            alpha(i)= eta(i)-atan(Vel(3)/Vel(2)); %upstroke
        else
            alpha(i)=-eta(i)+atan(Vel(3)/Vel(2)); %downstroke
        end
    end

    %% calculate viscosity and curculation terms
    Fvx=.5.*rho_f.*c_of_r.*(cd_of_0.*cos(alpha).^2+cd_of_piOver2.*...
        sin(alpha).^2).*sqrt(vxPrime.^2+vyPrime.^2).*vxPrime.*dr;%kg*mm/s^2
    Fvy=.5.*rho_f.*c_of_r.*(cd_of_0.*cos(alpha).^2+cd_of_piOver2.*...
        sin(alpha).^2).*sqrt(vxPrime.^2+vyPrime.^2).*vyPrime.*dr;%kg*mm/s^2

    Tau=-.5.*ct.*c_of_r.*sqrt(vxPrime.^2+vyPrime.^2).*sin(2.*alpha)+...
        .5.*cr.*(c_of_r.^2).*etaDot;  %in mm^2/s

    %% forces in wing frame
    dFxPrime=((((c_of_r/(cBar*R))*Mwing+m22).*vyPrime.*etaDot-rho_f.*...
        Tau.*vyPrime-m11.*axPrime).*dr-Fvx)/1000; %in N (kg*m/s^2)
    dFyPrime=(((-(c_of_r/(cBar*R))*Mwing+m11).*vxPrime.*etaDot+rho_f.*...
        Tau.*vxPrime-m22.*ayPrime).*dr-Fvy)/1000; %in N (kg*m/s^2)

    %% rotate forces back into labratory frame
    labForces=zeros(3,length(t));
    for i=1:length(t)
        R1=[cos(eta(i)),sin(eta(i));-sin(eta(i)),cos(eta(i))];
        R2=[-sin(phi(i)),cos(phi(i)),0;-sin(theta(i))*cos(phi(i)),...
            -sin(theta(i))*sin(phi(i)),cos(theta(i))];

        labForces(:,i)=pinv(R1*R2)*[dFxPrime(i);dFyPrime(i)]; %in N
        vectorMag=sqrt(labForces(1,i)^2+labForces(2,i)^2+labForces(3,i)^2);
        labForces(:,i)=labForces(:,i)./vectorMag; %normalize
    end

    totalLift=totalLift+sum(labForces(3,:)).*dt;

end

%% calculate lift
Lift=2*totalLift; %there are two wings
LiftToMassRatio=Lift/(m*9.81);
Lift=-scaling*Lift;
% display(LiftToMassRatio);
%
% subplot(3,1,1)
% plot(t,phi,':',t,eta,'--',t,theta);
% legend('\phi','\eta','\theta');
% ylabel('rad');
%
% subplot(3,1,2)
% plot(t,alpha);
% ylabel('\alpha (rad)');
%
% subplot(3,1,3)
% plot(t,vxPrime,':',t,vyPrime)
% legend('v_x''','v_y''');
% xlabel('Time (s)');
% ylabel('mm/s');


end

```

A.3 Find Point with Matching Gradient

```
function [x_p_toMatchHardware,error2,exitflag,output2] =...
    FindMatchingGradientWithHybrid(shift,gradH)
%This is the main function for a genetic algorithm

%bring in global variables
global PhysicsUpperBound PhysicsLowerBound

PUB=PhysicsUpperBound;
PLB=PhysicsLowerBound;
ShiftVector=6.*[1;%x1 in degrees
                1; %x2
                1;
                1;
                1];

PhysicsUpperBound=-ShiftVector+[41.6217165013473;71.7495633590574;...
    -156.404542398670;15;-109.827704076146]+1+2.*shift+0;
PhysicsLowerBound=-ShiftVector+[41.6217165013473;71.7495633590574;...
    -156.404542398670;15;-109.827704076146]-1+2.*shift-0;

%define an initial population
populationSize=40; %use a multiple of 4
numberOfGenerations=1000;

population=zeros(length(PhysicsUpperBound),populationSize);

for i=1:populationSize
    for j=1:length(PhysicsUpperBound)
        population(j,i)= PhysicsLowerBound(j)+rand(1)*...
            (PhysicsUpperBound(j)-PhysicsLowerBound(j));
    end
end

%evaluate the fitness of each gene
populationFitness=zeros(1,populationSize);
for j=1:populationSize
    populationFitness(j)=...
        GradientDifference(population(:,j),shift,gradH,'fmincon');
end

progressOverTime=ones(numberOfGenerations,1).*min(populationFitness);
for i=1:numberOfGenerations

    %determine which genes will become parents
    [parents,parentFitness]=performSelection(population,populationFitness);

    %perform crossover
    [children] = performCrossover(parents);

    %perform mutation;
    mutantChildren=performMutation(children,i,numberOfGenerations);

    %evaluate the fitness of each child
    childFitness=parentFitness.*0;
    for j=1:populationSize/2
        childFitness(j)=GradientDifference(mutantChildren(:,j),shift,...
            gradH,'fmincon');
    end

    population=[parents,mutantChildren];
    populationFitness=[parentFitness,childFitness];
    progressOverTime(i)=min(populationFitness);

    figure(1)
    subplot(2,1,1)
    plot(populationFitness,'*');
    title(sprintf(['Generation ' num2str(i) ' of ' ...
        num2str(numberOfGenerations)]));
    refline(0,min(populationFitness));
    ylim([0 2]);
    subplot(2,1,2)
    plot(progressOverTime);

end

%find the most fit of the last population
[x_p_toMatchHardware,error2]=findTheWinner(population,populationFitness);
exitflag=0;
output2=0;

PhysicsUpperBound=PUB;
PhysicsLowerBound=PLB;

useGradientMethods=true;
functionToUse='fsolve';

if useGradientMethods==true
    if strcmp(functionToUse,'fsolve')
        fsolveStartPoint=x_p_toMatchHardware;
        %find the point
        OPTIONS3 = optimset('TolFun', 1e-9,'Display','iter','MaxIter',...
            5000,'MaxFunEvals',1000,'TolX',1e-15);
        [x_p_toMatchHardware,error3,exitflag,output2]=...
            fsolve(@(x_p_toMatchHardware) GradientDifference(...
            x_p_toMatchHardware,shift,gradH,'fsolve'),fsolveStartPoint...
            ,OPTIONS3);
        error2=sqrt(error3(1)^2+error3(2)^2+error3(3)^2+error3(4)^2+...
            error3(5)^2);
    elseif strcmp(functionToUse,'fmincon')
        fminconStartPoint=x_p_toMatchHardware;%set up the starting point
        %find the point
        OPTIONS2 = optimset('MaxFunEvals',5000,'TolFun',1e-7,'Display',...
            'iter','MaxIter',5000,'TolX',1e-12);
        [x_p_toMatchHardware,error2,exitflag,output2]=fmincon(@(...
            x_p_toMatchHardware) GradientDifference(x_p_toMatchHardware,...
            shift,gradH,'fmincon'),fminconStartPoint,[],[],[],[],...
            PhysicsLowerBound,PhysicsUpperBound,[],OPTIONS2); %#ok<ASGLU>
    end
end
end

```