Assignment 7 Report

AEROSP 588 – Dr. Joaquim Martins

University of Michigan

December 16, 2025

Austin Leo Thomas

I. Abstract

An optimization study was undertaken to apply numerical optimization techniques to a classical aircraft control problem involving minimization of simulated flight time in an autopilot simulation by varying PID gain values in the digital flight controller. First, an intradisciplinary study was undertaken to explore the efficacy of various optimization techniques in solving the problem with the simulation model as the solve objective function. The DIRECT method proved to be the most effective approach, offering a 33% reduction in flight time when compared to the baseline yielded from manual gain tuning.

The DIRECT method was then applied to a multidisciplinary system adapted from the original problem. A coupled system was introduced, joining the simulation model with a control effort model which determined the control effort required under the current controller configuration and provided feedback to the simulation model to reduce that control effort. The coupled system was solved via Gauss-Seidel iteration and minimized. The DIRECT method successfully yielded a solution which reduced the flight time by 33% while only increasing the control effort by 4.7%, offering a robust solution to the presented multidisciplinary problem.

II. Introduction

A. Background and Motivation

The field of control theory is no stranger to the concept of numerical optimization. Indeed, optimal control is a critical element of modern engineering control systems. The concept of optimality may also be applied to classical control systems through employment of design optimization algorithms such as those defined in *Engineering Design Optimization* [1]. This investigation first endeavored to apply optimization techniques to an aircraft control problem and assess the efficacy of various methods in optimizing the aircraft behavior. The problem scope then expanded to consider a multidisciplinary version of the design wherein the original controls model was coupled with a control effort model.

This investigation served as an extension of a more rudimentary controls project for another course at a previous university. That previous project involved modelling an off-the-shelf drone aircraft in CAD software and running CFD simulations on that model to obtain the vehicle state-space matrices. These matrices were then passed to an autopilot simulation, designed in Simulink, which utilized three separate controllers (for roll, pitch, and yaw control), to guide the aircraft through a series of waypoints in a virtual flight space. The goal of the project was to manually tune seven PID gain values to yield acceptable performance from the aircraft. The metric for success of this project was simply that the aircraft successfully passed through all waypoints. Here, that metric was pushed further as an optimal solution, rather than a passable solution, was found for the system. That is, the investigation sought to find, via optimization techniques, the seven gain values

which minimized the flight time of the aircraft through the virtual waypoints.

This idea was then further pushed by introducing a second discipline to the system. This second discipline was imagined to model the power draw required by the control surfaces of the aircraft. One of the outputs of the autopilot simulation was the aileron, rudder, and elevator deflection angles at each simulation time instance. From this data it was easy to calculate the total control effort of the three aircraft control surfaces, measured as the sum change in deflection angle required by the three surfaces throughout the duration of the flight. This control effort cost was then be used as a metric to scale back the control effort of the aircraft by directly penalizing the input matrices in the state-space models for each control surface. This coupled model was then optimized for total flight time.

Since manual tuning of the controller gains for the autopilot simulation had already been undertaken, a viable solution was immediately available to serve as a jumping-off point for the optimization study. The seven gain values yielded from the manual tuning process were clear choices for an initial point to be passed to optimizers which required one. The flight time found by passing these gains to the simulation also served as an excellent benchmark to measure the efficacy of each optimization method. This baseline flight time was found as $\tilde{f} = 41.279$ seconds. The associated gains are:

$$\widetilde{K}_d = 1 \qquad \widetilde{K}_h = 1.1$$
$$\widetilde{K}_k = 1 \qquad \widetilde{K}_q = 39.6$$
$$\widetilde{K}_t = 6.46 \qquad \widetilde{K}_p = 1.03$$
$$\widetilde{K}_r = -0.12$$

The subscripts of each gain value is associated with a particular state in the classical 12×12 flight dynamics state-space representation. The specific meaning or relevance of each gain value is not pertinent to this investigation and as such is not discussed. The $\sim$ above the variables is used here, and in the future, to reference the baseline design.

Though the design space is 7-dimensional, it is still possible to visualize the optimization path by plotting the path of the aircraft through the virtual flight space. Though this path does not show the flight time or design variables, it does offer a sense of the overall efficiency and quality of the aircraft autopilot controller which the gains affect.

Fig. (1) shows the flight path of the virtual aircraft for the baseline design. It should be noted that while the waypoints are singular points, the path only needs to pass in a certain vicinity of the waypoint to hit is. As such, the flight path may be successful even if it does not directly intersect a waypoint. These trajectory plots offer an intuitive and comprehensible representation of the optimization process, something that is difficult to achieve with higher-dimensional optimization problems.

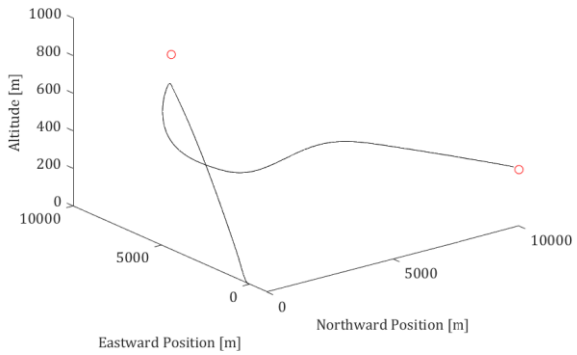

Figure 1. Flight path of the virtual aircraft with the baseline controller design.

B. Model Structure

The underlying model for the aircraft dynamics upon which this investigation relies was a MATLAB / Simulink flight simulation program. This program included hard-coded state-space matrices, flight parameters, etc. These values were not modified so as to preserve the accuracy of the system physics. The model was modified to allow for integration with a Python-based solving script, including reformatting the model as a MATLAB function which took a matrix of gains as the input and returned key values such as flight time and actuator states. A new variable, β, was also introduced as a scaling parameter on the state-space input matrices for the actuators; this parameter was another input to the function and allowed for model coupling for the MDO system. This primary model is defined mathematically in Eq. (1).

$$f(x,\beta) = \texttt{AutoPilot}(x,\beta) \tag{1}$$

Here, f is the flight time required for the aircraft to reach the second waypoint, x is the vector of design variables (i.e. PID gains), and β is the control scaling parameter which was passed to the simulation model from the control effort model.

From an optimization perspective, this simulation model was a black box. Though the Simulink block diagrams and underlying code could be accessed, the complexity of the system made any attempt at formal gradient estimation impossible. Methods such as complex-step approximation or algorithmic differentiation are incompatible with Simulink modelling while implicit analytic and UDE approaches, though technically feasible, would be extraordinarily difficult to implement. As such, optimization techniques were limited to gradient-free approaches or gradient-based approaches which utilized finite-difference approximation exclusively.

The control effort model was much less complex than the simulation model. This complementary model was based in the cost function described in Eq. (2).

$$\phi = \sum_{i=1}^{3}\sum_{j=1}^{n_f}\left|\theta_i^{(j)} - \theta_i^{(j-1)}\right| \tag{2}$$

Here, ϕ represents the control effort of the control effort (measured in degrees), $\theta_i^{(j)}$ represents the deflection angle of the i^{th} control surface at sampling instance j, and n_f represents the sampling instance at which the aircraft reaches the final waypoint. This cost function is exclusively a function of results from the simulation model and does not explicitly depend on x or β. However, both x and β implicitly affect the value of ϕ greatly.

To couple these two models, the control scaling parameter β was introduced to reduce the control effort of designs which may initially have an extremely high level of control surface actuation during the flight. β was defined according to Eq. (3) and (4).

$$\begin{aligned} &\text{if } \phi > \tilde{\phi}\text{:} \quad \bar{\beta}_k = \max\left(0.5, 1 - \frac{\phi - \tilde{\phi}}{\tilde{\phi}}\right) \\ &\text{else:} \quad \bar{\beta}_k = 1 \end{aligned} \tag{3}$$

$$\beta_{k+1} = \beta_k + \lambda\left(\bar{\beta}_k - \beta_k\right) \tag{4}$$

Here, $\bar{\beta}_k$ is the raw control input scaling parameter, $\tilde{\phi} = 6988°$ is the control effort of the baseline design, β_{k+1} is the β-value for

next scaling parameter, β_k is the current scaling parameter, and $\lambda = 0.5$ is a relaxation parameter which eases convergence and improves solver robustness. The k subscripts here represent the iteration number for an iterative solver.

The coupled system was solved iteratively via a Gauss-Seidel solver, the structure of which is depicted in Fig. (2). For a given set of design variables provided by the optimizer, the Gauss-Seidel solver would run the simulation model with $\beta_0 = 1$ to obtain f- and θ-values,

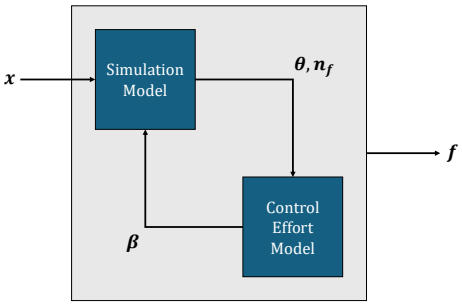

Figure 2. Coupling diagram depicting the two-model MDO system, its input and output, and the coupling variables which relate the models and allow for iterative solution via Gauss-Seidel.

run the control effort model to obtain ϕ, compute the coupling variable β_1, and then iterate. The solver terminated either at an iteration limit of $k = 15$ or at a β-based tolerance, defined in Eq. (5), of $\tau_\beta = 10^{-3}$.

$$\tau_\beta = |\beta_{k+1} - \beta_k| \tag{5}$$

One significant limitation of this system is the sensitive nature of the simulation model. In general, complex control systems are susceptible to erratic and highly noisy behavior when faced with even minor changes in their gain values. As such, safeguard were required to prevent optimization failure in the event that the simulation model failed to converge. The model simulated 100 seconds of flight; if the second waypoint was never reached, the time of flight was assigned to be $f = 100$ seconds so as to encourage the optimizer to deviate from unstable regions. There were also a number of cases where the Simulink run would crash, stall, or return errors. In all such cases, the solution was considered to be very poor as the objective function value was assigned as $f = 100$. In such cases were a set of gains yielded an inviable simulation result, the Gauss-Seidel solver was skipped entirely.

C. Solution Approach

Before considering the multidisciplinary problem, the simpler intradisciplinary case was considered wherein the simulation model represented the entire system. Consideration of this simpler problem would allow for exploration of the design space and determination of reasonable bounds to be placed on the design variables.

This study would also allow for determination of the most efficient and accurate optimization method for handling the simulation model. Since the simulation model was the more complex portion of the coupled system, it stood to reason that the best algorithm for solving the simulation model would also be the best algorithm for solving the coupled model. Because the coupled system required Gauss-Seidel iterations within each optimization iteration, the number of function calls required to converge to a solution of the coupled system was expected to be much greater than the number

required to solve only the simulation model. Since the simulation took a non-trivial amount of time to evaluate (typically $1 - 3$ seconds), it was determine the best optimizer for the problem prior to attempting to optimize the coupled system.

The simpler intradisciplinary optimization problem may be expressed as:

$$\begin{aligned} \text{minimize:} \quad & f(x) = \texttt{AutoPilot}(x) \\ \text{by varying:} \quad & x \in \mathbb{R}^7 \end{aligned}$$

Because the model was designed to automatically penalize unstable or non-converging designs with an extremely high objective function value, the problem need not be explicitly constrained. However, this stability constraint is essentially enforced as a built-in penalty method applied directly to the model. Because the constraint violation is made evident by a non-converging Simulink result, there is no need to formally define constraint functions and apply proper interior or exterior penalty methods.

Referring to the definitions in *Engineering Design Optimization*, this problem was classified as a continuous, single-objective, unconstrained, discontinuous, nonlinear, multimodal, nonconvex, and deterministic optimization problem [1]. As such, it was readily approached with gradient-free approaches such as the DIRECT, GPS, and PSO algorithms, as well as the gradient-based BFGS approach with finite-differencing. Though more suitable for unimodal problems, Nelder-Mead is also useful due to its lack of reliance on design variable bounds.

With this considered, the intradisciplinary study was first approached with a Nelder-Mead algorithm in hopes of finding a local minimum in the vicinity of the baseline solution. Then, DIRECT and GPS algorithms were employed to perform a global search across a broader region of the design space. Semi-arbitrary bounds were placed on the design variables at first, and these bounds were expanded as solutions began to indicate they may be needlessly constraining. A PSO algorithm was then applied to assess the performance of a stochastic approach on solving the problem. Finally, a BFGS algorithm was employed to attempt gradient-based optimization via finite-difference estimation.

The more complex multidisciplinary optimization problem may be expressed as:

$$\begin{aligned} \text{minimize:} \quad & f(x) = \texttt{MDO}(x) \\ \text{by varying:} \quad & x \in \mathbb{R}^7 \end{aligned}$$

Here, the `MDO()` function is the MATLAB / Python function which Fig. (1) represents. It is readily apparent that the forms of the two problems are identical; the only difference is the function which is being optimized. Indeed, all the added complexity of the multidisciplinary system is embedded in the `MDO()` function: the coupling variables and Gauss-Seidel solver are all handled within the objective function, invisible to the optimizer. As a result, the same approaches taken for solving the intradisciplinary problem are valid for the multidisciplinary problem: both problems have the same classification and the same form. The only difference, from an optimization perspective, is that the multidisciplinary solution requires many more function calls to the two models to successfully optimize.

III. Intradisciplinary Design

A. Deterministic Gradient-Free Methods

The first approach for solving the intradisciplinary problem was the Nelder-Mead algorithm, a deterministic and gradient-free optimization tool which performs a local search in the vicinity of the provided starting point. SciPy's `minimize` function was employed to orchestrate the optimization procedure. Convergence tolerances were provided to the optimizer on both the objective function and the design variables themselves; those tolerances were $\tau_f = 10^{-2}$ and $\tau_x = 10^{-3}$, respectively. Physically, this meant that if the optimum flight time began to vary by less than 10 milliseconds or all gain values began to vary by less than 10^{-3}, the optimizer would terminate. These tolerances were selected heuristically after some trial and error, with an effort to choose values with provided an accurate solution without becoming so tight as to forbid convergence given the noisy design space. The optimum flight time was found to be 35.163 seconds. The optimizer converged after 122 iterations and 300 function evaluations. The optimum gain values for this solution were found as:

$$K_d^* = 1.326 \qquad K_h^* = 1.341$$
$$K_k^* = 1.045 \qquad K_q^* = 18.890$$
$$K_t^* = 8.767 \qquad K_p^* = 1.095$$
$$K_r^* = -0.074$$

This solution represented a significant improvement in the control response of the aircraft, constituting a 14.8% reduction in flight time with relatively little computational cost. However, examination of the optimum gain values found by the Nelder-Mead solver revealed little variation between the baseline and optimum gain values. This was indicative that this was indeed only a local minimum in the immediate vicinity of the baseline solution. Given the extremely multimodal nature of complex control system, it was assumed that other, potentially superior minima would be found by a global search. The flight path of this solution is shown in Fig. (3).

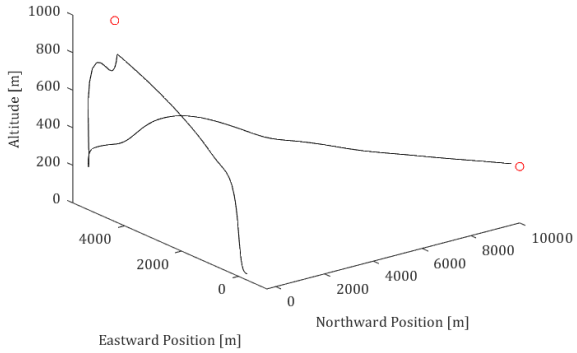

Figure 3. Flight path of the virtual aircraft with the Nelder-Mead optimum controller design.

The next algorithm which was considered was the DIRECT method. This was a global search method and as such required definitions of design variable bounds. These bounds were initially defined to have lower and upper limits of $\pm 10^{-3}$ to enforce that positive and negative gains, respectively, do not drive to zero or flip signs (which would guarantee instability). The other bound was set to be 150% of the baseline value. Thus the initial problem bounds were:

$$K_d \in [10^{-3}, 2.5] \qquad K_h \in [10^{-3}, 2.75]$$
$$K_k \in [10^{-3}, 2.5] \qquad K_q \in [10^{-3}, 99]$$
$$K_t \in [10^{-3}, 16.5] \qquad K_p \in [10^{-3}, 2.575]$$
$$K_r \in [-0.3, -10^{-3}]$$

The DIRECT method was then applied with the same τ_f and τ_x tolerances as were employed for the Nelder-Mead approach. The first optimum solution had a K_k^*-value of 2.485, right at the edge of that variable's upper bounds. As such, the bound for that variable was expanded to $K_k \in [10^{-3}, 5]$ and the process was repeated. This next optimum solution had a K_h^*-value of 2.700 and a K_d^*-value of 2.480, both also very near their upper bounds. Those variables' bounds were likewise doubled to become $K_h \in [10^{-3}, 5.5]$ and $K_d \in [10^{-3}, 5]$. The process was repeated and on the third run the bounds of K_t were expanded to $K_t \in [10^{-3}, 32.3]$. Thus the final design variable bounds were set as:

$$K_d \in [10^{-3}, 5] \qquad K_h \in [10^{-3}, 5.5]$$
$$K_k \in [10^{-3}, 5] \qquad K_q \in [10^{-3}, 99]$$
$$K_t \in [10^{-3}, 32.3] \qquad K_p \in [10^{-3}, 2.575]$$
$$K_r \in [-0.3, -10^{-3}]$$

With these new bounds, the DIRECT method yielded much improved optimum flight time of 27.666 seconds. The solution converged after 25 iterations requiring 627 function evaluations. The optimum gains for this solution were:

$$K_d^* = 2.439 \qquad K_h^* = 3.239$$
$$K_k^* = 4.249 \qquad K_q^* = 46.241$$
$$K_t^* = 14.157 \qquad K_p^* = 0.525$$
$$K_r^* = -0.042$$

Unlike the Nelder-Mead optimum, which remained quite close to the baseline design, the gains in the DIRECT method optimum were far removed from the baseline. This was an expected result given that the DIRECT method offers a global search. The reward for this expanded search was a much better optimum function value constituting a 33.0% improvement in flight time over the baseline. This improvement is reflected in the flight path, shown in Fig. (4), which is clearly a more direct and efficient path when compared to the previous two solutions.

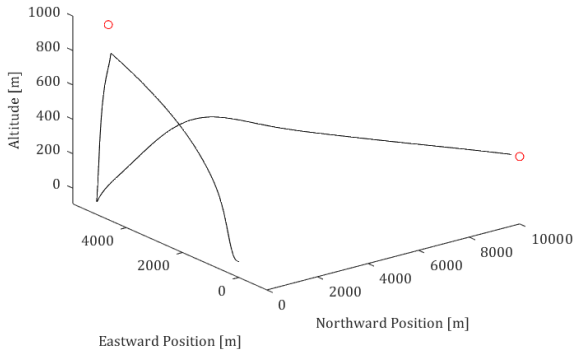

Figure 4. Flight path of the virtual aircraft with the DIRECT method optimum controller design.

This is, of course, not at all guaranteed to be a global optimum, though it is clear from the flight path plot that there is little efficiency loss due to unnecessarily winding trajectories. It is also worth noting that, while effort was taken to expand the design variable bounds as much as needed, there is no guarantee that some better design doesn't lie outside the bounds set for this study. Attempts were made to set the bounds to be much larger, with gains allowed to range in magnitude up to 10^2; however, such methods struggled to converge under the DIRECT method since most points tested by the optimizer were unstable. The result was essentially geometric division of the design space with no mathematical order, as nearly every point probed by the optimizer returned the same objective function value. Thus it was deemed necessary that some

realistic bounds be placed on the gains to minimize the evaluation of unstable designs, even if those bounds were somewhat arbitrary.

The next approach considered employed a GPS algorithm to perform a different type of global search. The same bounds were provided to the GPS algorithm as for the DIRECT method, and no cause was presented to expand those bounds. Unlike the Nelder-Mead and DIRECT method solutions, convergence in the GPS solution was never reached, despite the same values for τ_f and τ_x being provided to the algorithm. Even more interesting was that there was not a direct correlation between the number of function evaluations allowed by the optimizer and the flight time decreasing. Indeed, the GPS algorithm was ran several times with varying limits enforced on the maximum allowable number of function evaluations, and at some point the algorithm demonstrated diminishing returns: past a certain point, increasing the number of function evaluations produced a worse result.

The results for the GPS study are summarized in Table (I). The exact reasoning for this behavior was not able to be determined. Initially is was considered that the poor optimizer performance was the result of noise, though no other optimizers displayed such behavior. The nature of GPS should in theory preclude this type of behavior for any reason other than noise. The most reasonable explanation is that, while the Nelder-Mead and DIRECT algorithms were courtesy of SciPy (`minimize` and `direct`, respectively), the GPS algorithm was an NLopt package. It is possible that the SciPy modules handled the noisy objective function better than the NLopt module was able to.

Examination of Table (I) reveals that the lowest optimum flight time found by the GPS algorithm was 38.996 seconds, achieved with 100 function calls. Though not quite was good as the DIRECT method solution, this result still constitutes a 29.8% reduction in flight time relative to the baseline solution. Interestingly, despite how similar the optimum function values found by the DIRECT and GPS methods were, the optimum gain values found by each differed substantially, particularly with respect to K_q, K_t, and K_p. The optimum gains found by the GPS algorithm are:

Table I
Summary of GPS Results

Function Calls	Optimum Flight Time [s]
5	34.128
10	34.302
30	35.295
50	32.956
100	28.996
150	29.690
200	37.896
250	30.206
500	31.711
700	32.386

$$K_d^* = 2.130 \qquad K_h^* = 2.231$$
$$K_k^* = 3.951 \qquad K_q^* = 28.193$$
$$K_t^* = 20.744 \qquad K_p^* = 0.110$$
$$K_r^* = -0.036$$

Comparison of the GPS optimum design flight path, shown in Fig. (5), and the DIRECT method optimum design flight path, shown in Fig. (4), reveals that although the trajectories are very similar, the control behavior of the GPS design seems to be slightly less controlled and more erratic, likely lending to the slightly longer flight time. These plots illuminate the differences in the control behavior of both designs which is seen in the large variance

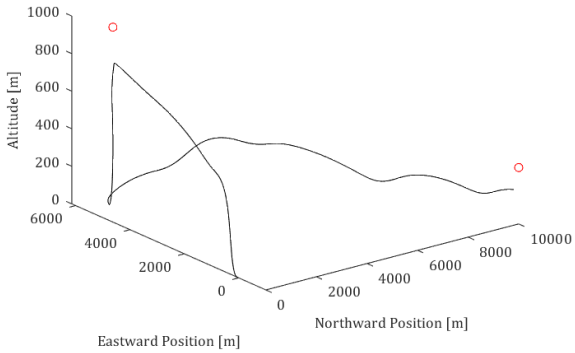

Figure 5. Flight path of the virtual aircraft with the GPS algorithm optimum controller design.

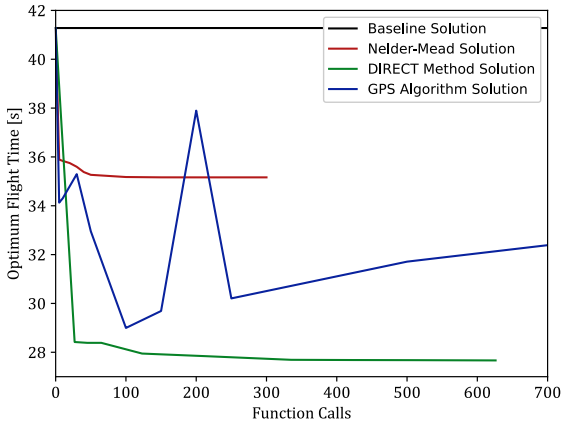

Figure 6. Performance of the Nelder-Mead, DIRECT method, and GPS algorithms as a function of the maximum number of function calls available to the optimizer. The baseline result is included for reference.

between the gain values: clearly both methods converged to difference local minima within the design space which only coincidentally had similar optimum objective function values.

Since a major goal of this study was to identify which algorithm would be the most efficient and accurate for use in the multidisciplinary study, the performance of each of these deterministic and gradient-free methods was considered as a function of the number of function calls the optimizer was allowed to make. A well-suited algorithm would be able to reach a result very near its optimum value with far fewer function calls than was required to truly converge within the defined function and design variable tolerances. A plot of the optimum flight time found by each algorithm as a function of the number of allowed function calls made is shown in Fig. (6).

Examination of Fig. (6) confirms the analysis already done on these results. The GPS algorithm performed the worst of the three approaches, offering a highly erratic results which tended to get worse with increasing function calls. Both the Nelder-Mead and DIRECT methods offered excellent optimization performance even at low numbers of function calls. However, the DIRECT method was clearly the best of the three: it rapidly approaches its optimum value and found the most optimum solution of the three methods. The DIRECT method offered a highly accurate result even with limited function calls. It was also highly consistent with its behavior and never regressed in its solution as the number of allowed function calls increased.

B. Stochastic Gradient-Free Methods

Another approach to a gradient-free global optimization search is the evolutionary algorithm, a class of stochastic, global, gradient-free algorithms which tend to be well-suited to highly noisy and erratic functions due to their tendency to explore regions of the design space which other algorithms may not venture into. For the purposes of this study, a particle swarm optimization (PSO) approach was taken.

Following heuristic rules for evolutionary algorithms, a constant swarm size of 14 (twice the size of the design space) was assigned for each optimization run [1]. The stochastic nature of PSO algorithms necessitates the collection of a statistical sampling of the results for each configuration of the optimizer. Due to the high computational cost of polling the simulation model, a limited $n = 5$ sample was taken for each optimization configuration.

The optimizer used was courtesy of the PySwarm `pso` module. The only convergence criteria supported by this optimizer is user-enforced iteration limits. As such, samples were collected with iteration limits of 5, 10, and 20. Larger samples (or even larger swarm sizes) would in theory allow for better exploration of the design space. However, because statistical sampling is required, there must be a trade-off between the number samples taken per configuration and the number of configurations attempted. Further, because of this statistical sampling, the design space ought to be amply explored even with low swarm sizes and iteration limits.

The same design variable bounds were used for the PSO procedure as was used for the DIRECT method and GPS runs. As before, no optimum were found to lie on the edge of any of these bounds, so no cause was presented to expand the design space. The results of the statistical study are summarized in Table (II).

Table II
PSO Statistical Study Results

Iterations	μ_{f^*} [s]	σ_{f^*} [s]	Best f^* [s]
5	32.882	1.869	29.545
10	31.349	3.167	27.886
20	31.914	2.926	27.648

Examination of Table (II) reveals that the 5-iteration optimization runs performed only marginally worse than the 10- and 20-iteration runs, which performed nearly identically. Additionally, it is seen that the 10- and 20-iteration runs have significantly greater standard deviations when compared to the 5-iteration run, indicating that the increased number of iterations does indeed allow for better exploration of the design space. However, based on these results there is little reason to believe results would significantly improve if the number of iterations was increased further.

The best PSO solution came, expectedly, in one of the 20-iteration runs. The solution yielded an optimum flight time of 27.648 seconds and required 294 function calls. This result was actually marginally better than the solution found by the DIRECT method, also constituting a 33.0% reduction in the flight time compared to the baseline solution. The solution also required far fewer function evaluations (294 calls compared to the 627 calls made by the DIRECT solution). Though this result may seem to indicate that the PSO approach was superior to the DIRECT method, it must be recognized that the PSO algorithm necessitates sampling. In reality, 1470 function calls were required to complete the 5-run statistical batch of 20-iteration PSO runs, making the PSO approach far less computationally efficient when compared to the deterministic DIRECT method. The optimum gain values for this PSO solution are:

$$K_d^* = 2.439 \qquad K_h^* = 3.239$$
$$K_k^* = 4.249 \qquad K_q^* = 46.241$$
$$K_t^* = 14.157 \qquad K_p^* = 0.525$$
$$K_r^* = -0.042$$

These gains are, in fact, identical to the optimum gains found by the DIRECT method. Indeed, this best-case PSO solution and the DIRECT method solution are the same local (or perhaps global) minima in the design space. This revelation further bolstered the claim that the DIRECT method was superior to the PSO approach: the DIRECT method converged, and would always converge, to this best-case solution which the PSO algorithm found only once in 15 separate searches. Clearly the stochasticity of the PSO algorithm did not improve the solution efficiency. The flight path for the best PSO solution is shown in Fig. (7). Expectedly, it is identical to the DIRECT method solution flight path shown in Fig. (4).

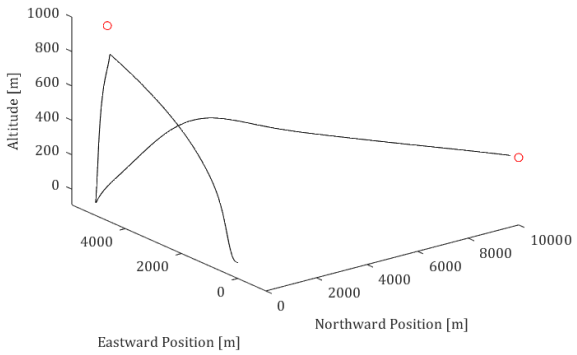

Figure 7. Flight path of the virtual aircraft with the best-case PSO optimum controller design.

C. Gradient-Based Methods

Finite-difference gradient approximations are generally very poor for noisy and discontinuous functions. As such, little hope was held for convergence of a gradient-based approach. Attempts were made with a BFGS solver (courtesy of SciPy's `minimize` module), but none bore fruit. Despite many attempts to tune the finite-differencing approach taken by the optimizer, the BFGS algorithm continuously found numerical instability in the gradient estimation step that prevented it from successfully executing. For all manually-set step sizes, the same error persisted:

```
Desired error not necessarily
achieved due to precision loss.
```

Research into the matter revealed that this is a common issue when passing very noisy and erratic functions to gradient-based optimizers (be they from SciPy or other sources) and that, in general, encountering this error despite manually tuning parameters indicated that the function is ill-suited to gradient-based optimization.

This outcome was not entirely unexpected given the nature of the simulation model. The idea of running step size studies to force a successful gradient-based approach was considered but not undertaken due to the extreme computational and time cost associated with such an approach. As such, after much troubleshooting and attempted tuning, efforts to configure a gradient-based approach were abandoned.

D. Discussion

Though the gradient-based approach to solving the intradisciplinary study was a failure, the gradient-free approaches proved very successful. All four methods yielded significantly improved results over the baseline, with two algorithms converging to the same optimum within the bounded design space. Such behavior from global search algorithms bolsters a strong case that the shared optimum is a global optimum within the design space.

Regardless, the study succeeded in its primary goal of identifying the best algorithm to be applied to the multidisciplinary problem. The DIRECT method proved both highly accurate, even with limited function calls, and highly efficient, requiring fewer function calls than the stochastic PSO approach while obtaining the same result. With this in mind, the DIRECT approach was the obvious choice for application to the multidisciplinary problem tackled next.

Table (III) offers a concise summary of the best results of each of the solutions found to the intradisciplinary problem.

Table III
Summary of Intradisciplinary Solutions

Method	f-Calls	f^*
Nelder-Mead	300	35.163
DIRECT	627	27.666
GPS	100	28.996
PSO	294	27.648

IV. Multidisciplinary Design

A. Gauss-Seidel Performance

Before the coupled system was optimized, consideration was given to the Gauss-Seidel coupled solver and its performance. As discussed before, the purpose of the coupled system is to drive the solver to solutions which not only improve the flight time of the aircraft but are able to do so while increasing the control effort of the aircraft control surfaces by as little as possible. This is accomplished not through the evaluation of multiple objectives – control effort is not being minimized – but rather through clever definition of a system which, when iterated through a Gauss-Seidel solver, innately reduces the control effort by scaling back the state-space input matrices directly. When such a coupled system is then optimized for flight time, the solver naturally must drive towards a solution which minimizes both flight time and control effort.

To better understand the coupled solver, the DIRECT method optimum solution to the intradisciplinary problem was passed to the Gauss-Seidel solver. Prior to Gauss-Seidel iteration, the solution had a control effort value of $\phi = 14229°$ (104% greater than the baseline value of $\tilde{\phi} = 6988°$. When passed to the coupled solver, the Gauss-Seidel solution converged to a new optimum flight time of 29.609 seconds (1.943 seconds slower than the non-coupled solution). This flight time was achieved with a converged scaling parameter of $\beta = 0.567$, representing 43.3% reduction in control effort (from $14229°$ to $8068°$).

This example illustrates the premise of the coupled model. When a set of gains are passed to the model, a β-value is returned which drastically reduces the control effort of the aircraft control surfaces while only marginally compromising the optimality of the flight time.

B. Optimization Results

All parameters for the DIRECT method optimizer remained the same for the multidisciplinary solution as for the intradisciplinary solution: the convergence criteria was a hypervolume less than 10^{-16} (the default for SciPy's DIRECT method algorithm) and the design variables bounds remained unchanged.

The algorithm was allowed to run until it converged after 26 iterations and 3975 function evaluations to an optimum flight time

of 27.529 seconds. The β-value of the final solution was 0.983. The control effort value of the solution was $\phi = 7332°$, only 4.7% greater than the baseline control effort. The optimum gain values for this solution are:

$$K_d^* = 2.439 \qquad K_h^* = 3.370$$
$$K_k^* = 4.229 \qquad K_q^* = 15.279$$
$$K_t^* = 5.384 \qquad K_p^* = 0.525$$
$$K_r^* = -0.083$$

This result is remarkable for a number of reasons. Firstly, it is seen that this optimum flight time was in fact 0.137 seconds lower than that found in the intradisciplinary study. Introducing coupling and seeking a solution with a limited impact on the control effort revealed a solution which was better than that found when control effort was not accounted for. Despite the fact that lessening control effort typically results in a less efficient response, in this case, with these gain values, reducing the magnitude of the state-space input values by just 1.7% results in better performance than can be achieved with the full control capacity of the aircraft. Further, this optimum design is entirely separate from those found in the intradisciplinary study: the combination of gains found in the MDO solution were not found by any of the intradisciplinary optimizers.

Once the converged optimum was identified, the optimizer was run several more times with manual limits placed on the number of function calls the optimizer could make. This allowed for assessment of the behavior of the coupled solver throughout the optimization process. The results of that investigation are summarized in Table (IV) and plotted in Fig. (8) though (10).

Table IV
MDO Results Summary

f-Calls	f^* [s]	ϕ [°]	β
209	30.700	10010	0.568
369	30.287	10367	0.516
808	29.735	10445	0.505
1426	28.104	7222	0.967
2897	27.544	6988	1.000
3975	27.529	7332	0.983

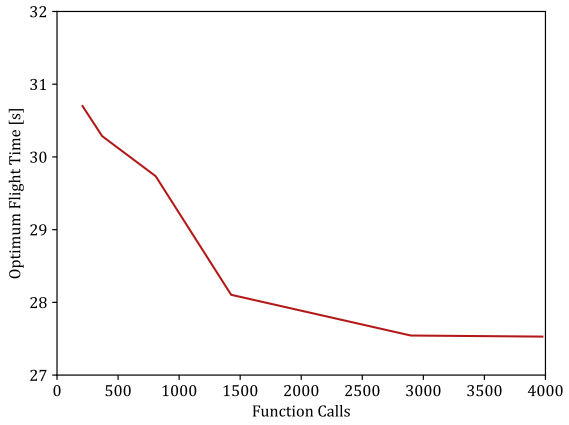

Figure 8. Plot of the optimum flight time found by optimizing the coupled MDO model with the DIRECT algorithm, as a function of the number of function calls allowed.

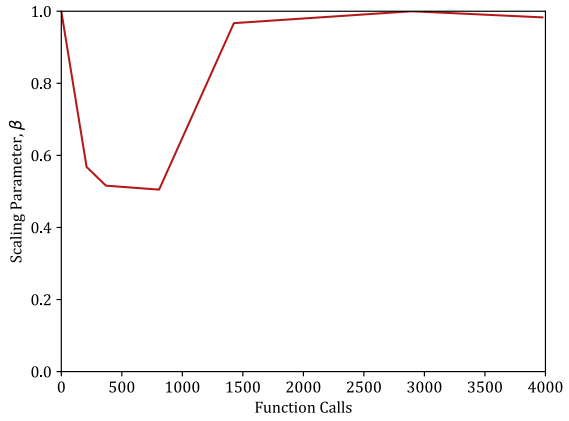

Figure 10. Plot of the β-value required by the optimum solution to the MDO problem, as a function of the number of function calls allowed.

Examination of Table (IV) and Fig, (8) and (9) reveal several more interesting aspects of this MDO solution. It is seen from the numerical results that, although the optimizer is not directly minimizing the control effort of the design, the nature of the model leads to ϕ driving downwards alongside the flight time f. Of interest as well is that, although the Gauss-Seidel solver initially converges to relatively small β-values (associated with state-space enforced control effort reduction rather than a controller which inherently requires low control effort), by the time the optimizer converges, the scaling parameter is near unity. This is very desirable behavior from the coupled model, as the ideal solution to the MDO problem would be a controller capable of time- and control-efficient flight without forcibly handicapping the controller by scaling back its input values.

It is also worth considering the difference between finding an intradisciplinary optimum and then passing those gains to the coupled solver to reduce the control effort, as was done in *Section IV(A)*. Table (V) summarized the differences in the results found by applying true MDO versus passing an intradisciplinary result to the MDO coupled solver (so-called pseudo-MDO).

Table V
Comparison of True Versus Pseudo-MDO

Value	True MDO	Pseudo-MDO
f^* [s]	27.529	29.609
ϕ [°]	7332	8068
β	0.983	0.567
f-Calls	3466	635

The results shown in Table (V) perfectly illustrate the need for a true coupled MDO optimization process. When an optimum solution is found for a single model (in this case, the simulation model) and is then passed to a coupled solver for iterative refinement, the results are limited. The coupled solver here can only affect the solution by varying the control input scaling parameter, essentially forcing the control input to be reduced by handicapping the controller. This pseudo-MDO approach leads to a result with a higher flight time and control effort value when compared to the true MDO result. Further, it is highly desirable from an engineering standpoint to make full use of a system's control capability. By applying true MDO approaches, a solution was found which minimized the core objective function and satisfied the realistic engineering considerations embedded in the control effort model. Coupling the two models allowed for a clean solution which did not requires the trade-offs associated with multi-objective optimization and Pareto fronts.

Of course, there are drawbacks to running an MDO algorithm, namely in computational cost. Because the introduction of a Gauss-Seidel solver necessitates multiple function evaluations per function call made by the optimizer, the computational cost of the algorithm scales by the average number of function calls made by the Gauss-Seidel solver. Table (V) shows that the computational cost of obtaining the true MDO solution is nearly six times that of finding the pseudo-MDO solution, which requires only one execution of the Gauss-Seidel iterative solver. For this particular problem, however, the true MDO solution proves significantly better than the alternative, and the tradeoff in computational cost is considered worthwhile.

Finally, it is worth considering the flight path taken by the MDO-optimal aircraft design and how it varies from its intradisciplinary counterpart. This plot is shown in Fig. (11).

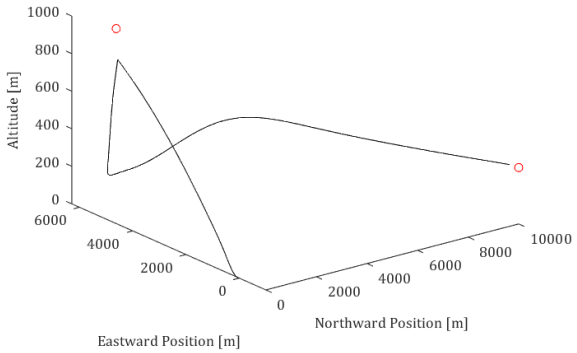

Figure 11. Flight path of the virtual aircraft with the MDO optimum controller design.

Comparison of Fig. (11) to the flight path of the intradisciplinary DIRECT method optimum flight path shown in Fig. (4) reveals several striking similarities. Indeed, the only significant difference between the two paths is that the MDO controller results in the plane terminating its dive following the first waypoint slightly earlier: the intradisciplinary optimum aircraft dives further than the multidisciplinary aircraft does. This may be readily understood to be a result of the introduction of the control effort model and the scaling parameter feedback. Because the dynamics of the aircraft under the coupled solver may be limited by β, and are generally encouraged to utilize less control effort, a deep dive (requiring a sharp upwards recovery and greater control surface activation) is less feasible. As such, the MDO optimum aircraft cut the dive off early to reduce its total control input. As it turns out, cutting the dive off early was indeed an improvement over the deep dive irrespective to the control effort; the intradisciplinary optimizers did not, however, discover this optimum within the design space, proving that MDO solutions may prove useful even from an intradisciplinary view.

V. Final Remarks

As a whole, the investigation was wildly successful in identifying optimum controller configurations for both the coupled and uncoupled systems. The intradisciplinary study proved successful in exploring the capabilities of several different gradient-free optimizers to navigate a noisy, discontinuous, and higher-dimensional design space. The introduction of a second model based in real engineering principles proved useful in the exploration of methods of multidisciplinary design optimization when applied to practical engineering problems. Several interesting and unexpected insights were found which proved useful to developing a greater understanding of the simulation model, coupled model, and of course the algorithms with which these models were solved.

VI. Intradisciplinary Design

As required by the syllabus, a disclosure is included here regarding AI use on this assignment. U-M GPT was used as an aide when writing the Python and MATLAB scripts employed in this assignment.

VII. References

[1] J. R. Martins and A. Ning, Engineering Design Optimization, Cambridge: Cambridge University Press, 2020.